THE UNIVERSITY OF
WARWICK

Library Declaration and Deposit Agreement

1. STUDENT DETAILS

Please complete the following:

Full name: ………………………………………………………………………………………….

University ID number: ………………………………………………………………………………

2. THESIS DEPOSIT

2.1 I understand that under my registration at the University, I am required to deposit my thesis with the University in BOTH hard copy and in digital format. The digital version should normally be saved as a single pdf file.

2.2 The hard copy will be housed in the University Library. The digital version will be deposited in the University's Institutional Repository (WRAP). Unless otherwise indicated (see 2.3 below) this will be made openly accessible on the Internet and will be supplied to the British Library to be made available online via its Electronic Theses Online Service (EThOS) service.
[At present, theses submitted for a Master's degree by Research (MA, MSc, LLM, MS or MMedSci) are not being deposited in WRAP and not being made available via EthOS. This may change in future.]

2.3 In exceptional circumstances, the Chair of the Board of Graduate Studies may grant permission for an embargo to be placed on public access to the hard copy thesis for a limited period. It is also possible to apply separately for an embargo on the digital version. (Further information is available in the *Guide to Examinations for Higher Degrees by Research*.)

2.4 *If you are depositing a thesis for a Master's degree by Research, please complete section (a) below. For all other research degrees, please complete both sections (a) and (b) below:*

(a) Hard Copy

I hereby deposit a hard copy of my thesis in the University Library to be made publicly available to readers (please delete as appropriate) EITHER immediately OR after an embargo period of ……..................... months/years as agreed by the Chair of the Board of Graduate Studies.

I agree that my thesis may be photocopied. YES / NO (*Please delete as appropriate*)

(b) Digital Copy

I hereby deposit a digital copy of my thesis to be held in WRAP and made available via EThOS.

Please choose one of the following options:

EITHER My thesis can be made publicly available online. YES / NO (*Please delete as appropriate*)

OR My thesis can be made publicly available only after…..[date] (Please give date)
YES / NO (*Please delete as appropriate*)

OR My full thesis cannot be made publicly available online but I am submitting a separately identified additional, abridged version that can be made available online.
YES / NO (*Please delete as appropriate*)

OR My thesis cannot be made publicly available online. YES / NO (*Please delete as appropriate*)

3. GRANTING OF NON-EXCLUSIVE RIGHTS

Whether I deposit my Work personally or through an assistant or other agent, I agree to the following:

Rights granted to the University of Warwick and the British Library and the user of the thesis through this agreement are non-exclusive. I retain all rights in the thesis in its present version or future versions. I agree that the institutional repository administrators and the British Library or their agents may, without changing content, digitise and migrate the thesis to any medium or format for the purpose of future preservation and accessibility.

4. DECLARATIONS

(a) I DECLARE THAT:

- I am the author and owner of the copyright in the thesis and/or I have the authority of the authors and owners of the copyright in the thesis to make this agreement. Reproduction of any part of this thesis for teaching or in academic or other forms of publication is subject to the normal limitations on the use of copyrighted materials and to the proper and full acknowledgement of its source.

- The digital version of the thesis I am supplying is the same version as the final, hard-bound copy submitted in completion of my degree, once any minor corrections have been completed.

- I have exercised reasonable care to ensure that the thesis is original, and does not to the best of my knowledge break any UK law or other Intellectual Property Right, or contain any confidential material.

- I understand that, through the medium of the Internet, files will be available to automated agents, and may be searched and copied by, for example, text mining and plagiarism detection software.

(b) IF I HAVE AGREED (in Section 2 above) TO MAKE MY THESIS PUBLICLY AVAILABLE DIGITALLY, I ALSO DECLARE THAT:

- I grant the University of Warwick and the British Library a licence to make available on the Internet the thesis in digitised format through the Institutional Repository and through the British Library via the EThOS service.

- If my thesis does include any substantial subsidiary material owned by third-party copyright holders, I have sought and obtained permission to include it in any version of my thesis available in digital format and that this permission encompasses the rights that I have granted to the University of Warwick and to the British Library.

5. LEGAL INFRINGEMENTS

I understand that neither the University of Warwick nor the British Library have any obligation to take legal action on behalf of myself, or other rights holders, in the event of infringement of intellectual property rights, breach of contract or of any other right, in the thesis.

Please sign this agreement and return it to the Graduate School Office when you submit your thesis.

Student's signature: ....…… Date: ....

Efficient Learning Methods to Tune Algorithm Parameters

by

Jawad A. El-Omari

Supervised by Professor Juergen Branke

A thesis submitted in the partial fulfilment of the requirements
for the degree of Doctor of Philosophy in
Operational Research and Management Sciences

University of Warwick, Warwick Business School

October 2013

Table of contents

Chapter 1: Introduction

Chapter 2: Literture Review

Chapter 3: Theory and Methodology

Chapter 4: Experiments, Results, and Analysis
Meta-Optimization with a Flexible Budget

Chapter 5: Experiments, Results, and Analysis
Computational Budget Allocators

Chapter 6: Conclusion

References

Appendix

Acknowledgement

To Lama, I could not have done this without you… I wish you were here today

List of figures

Chapter 1: Introduction

Chapter 2: Literature Review

Chapter 3: Theory and Methodology

Chapter 4: Experiments, Results, and Analysis
Meta-Optimization with a Flexible Budget

Chapter 5: Experiments, Results, and Analysis
Computational Budget Allocators

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy in Operational Research and Management Sciences. It has been composed by myself and has not been submitted in any previous application for any degree at any other university. All the work presented here, including the data and analysis, was carried out by the author.

Parts of this thesis have been published:

1. Branke, J. and Elomari, J. A., 2012. Meta-optimization for parameter tuning with a flexible computing budget. In*:* Soule, T. ed. *Genetic and Evolutionary Computation Conference.* Philadelphia, 7-11 July. New York: ACM.
2. Branke, J. and Elomari, J., 2013. Racing with a fixed budget and a self-adaptive significance level. In*:* Pardalos, P. and Nicosia, G. eds. *Learning and Intelligent OptimizatioN.* Catania, 7-11 Jan.

List of tables

Chapter 2: Literature Review

Chapter 3: Theory and Methodology

Chapter 4: Experiments, Results, and Analysis
Meta-Optimization with a Flexible Budget

Chapter 5: Experiments, Results, and Analysis
Computational budget allocators

Chapter 6: Conclusion

Abstract

This thesis focuses on the algorithm configuration problem. In particular, three efficient learning configurators are introduced to tune parameters offline. The first looks into meta-optimization, where the algorithm is expected to solve similar problem instances within varying computational budgets. Standard meta-optimization techniques have to be repeated whenever the available computational budget changes, as the parameters that work well for small budgets, may not be suitable for larger ones. The proposed Flexible Budget method can, in a single run, identify the best parameter setting for *all* possible computational budgets less than a specified maximum, without compromising solution quality. Hence, a lot of time is saved. This will be shown experimentally. The second regards Racing algorithms which often do not fully utilize the available computational budget to find the best parameter setting, as they may terminate whenever a single parameter remains in the race. The proposed Racing with reset can overcome this issue, and at the same time adapt Racing's hyper-parameter α online. Experiments will show that such adaptation enables the algorithm to achieve significantly lower failure rates, compared to any fixed α set by the user. The third extends on Racing with reset by allowing it to utilize all the information gathered previously when it adapts α, it also permits Racing algorithms in general to intelligently allocate the budget in each iteration, as opposed to equally allocating it. All developed Racing algorithms are compared to two budget allocators from the Simulation Optimization literature, OCBA and CBA, and to equal allocation to demonstrate under which conditions each performs best in terms of minimizing the probability of incorrect selection.

List of abbreviations

Abbreviation	Description	Page of definition
ACO	Ant Colony Optimization	1
AL	Area Lost	48
ANN	Artificial Neural Networks	25
ANOVA	Analysis of Variance	8
AOS	Adaptive Operator Selection	13
AOTA	Adaptive Online Time Allocator	35
AP	Adaptive Pursuit algorithm	14
A-Race	A Racing algorithm based on a normal-model ANOVA test	31
A-Race_1WUB	A Racing algorithm based on a normal-model 1-way unbalanced ANOVA test	66
A-Race_2Way	A Racing algorithm based on a normal-model 2-way ANOVA test	66
A-RaceR_2Way	Normal model-based 2-way ANOVA Race with Reset	75
A-RaceRR_1WUB	Normal model-based 1-way ANOVA Race with Reset and Resample (unbalanced)	75
ARIMA	Autoregressive Integrated Moving Average	24
ARMA	Autoregressive Moving Average	24
AUC	Area Under the Curve	47
avBSF	best-so-far average utility	47
BasicILS	Basic ILS (a version of the ParamILS algorithm)	31
BP	Back Propagation	25
CA	Credit Assignment	13
CBA	Correlated Budget Allocation	33
CE	Cross Entropy	34

CHeSC	Cross-domain Heuristic Search Challenge	21
CL-PSO	Comprehensive Learning Particle Swarm Optimization	15
CMA-ES	derandomized Evolution Strategy with Covariance Matrix Adaptation	78
DbV	Difference based Velocity update strategy	15
DE	Differential Evolution	25
DMAB	Dynamic Multi Armed Bandit	17
DP	Dynamic Programming	18
E[OC]	Expected Opportunity Cost	101
EA	Evolutionary Algorithm	1
EBA	Equal Budget Allocation	74
EbV	Estimation based Velocity update strategy	15
ES	Evolutionary Strategy	20
FocusedILS	Focused ILS (a version of the ParamILS algorithm)	31
f_r	Failure rate	101
F-Race	A Racing algorithm based on Freidman's test	29
F-RaceR	Non-parametric 2-way Freidman ANOVA Race with Reset (balanced)	75
GA	Genetic Algorithm	22
GE	Grammatical Evolution	25
GP	Genetic Programming	25
HRTS	Hamming Reactive Tabu Search	20
I\F-Race	Iterated *F*-Race	30
ILS	Iterated Local Search	18
KW	Kruskal-Wallis test	8
KW-Race	A Racing algorithm based on Kruskal-Wallis test	70
KW-RaceRR	Non-parametric 1-way Kruskal-Wallis ANOVA Race with Reset and Resample (unbalanced)	75

KW-RaceRR_OCBA	A hybridization of the *KW*-Race and OCBA algorithms	75
L	Length	47
LL	Lower Level algorithm	44
LSPI	Least Squares Policy Iteration	19
LUS	Local Unimodal Sampling	25
MAB	Multi Armed Bandit	15
MCHH	Markov Chain Hyper-heuristic	20
MMAS	Max-Min Ant System	26
OCBA	Optimal Computing Budget Allocation	33, 8
OPSO	Optimized Particle Swarm Optimization	24
OS	Operator Selection	13
ParamILS	Parameter Iterated Local Search	31
PCS	Probability of Correct Selection	51
PD	Pareto Dominance	15
PICS	Probability of Incorrect Selection	101
PM	Probability Matching	14
PM-AdapSS-DE	Probability Matching Differential Evolution with Adaptive Strategy Selection algorithm	13
P_{min}/P_{max}	Minimum/maximum selection probability	14
PR	Pareto Rank	15
PS	Parameter Setting	74
PSO	Particle Swarm Optimization	15
QAP	Quadratic Assignment Problem	26
REVAC	Relevance Estimation and Value Calibration	37
RL	Reinforcement Learning	19
RTS	Reactive Tabu Search	19
SaDE	Self-adaptive Differential Evolution	13

SAPS	Scaling and Probabilistic Smoothing	20
SAT	Boolean satisfiability problem	15
SD	Steepest Descent	18
SMAC	Sequential Model-based Algorithm Configuration	37
SMBO	Sequential Model Based Optimization	35
SPO	Sequential Parameter Optimization	36
STGP	Strongly Typed Genetic Programming	26
TB-SPO	Time Bounded Sequential Parameter Optimization	36
TS	Tabu Search	1
TSP	Traveling Salesman Problem	23
UCB1	Upper Confidence Bound algorithm	17
VNS	Variable Neighborhood Search	18

CHAPTER 1 Introduction

1.1. Motivation and problem statement

This thesis focuses on the algorithm configuration problem. Many search algorithms, and metaheuristics in particular, have several parameters that affect their performance in terms of solution quality and running time. Population size in Evolutionary Algorithms (EA), tabu list length in Tabu Search (TS), and evaporation rate in Ant Colony Optimization (ACO) are examples of numerical parameters, while parent and survival selection types in EAs are examples of categorical parameters. Incorrectly setting these parameters can, in extreme cases, cause the algorithm to behave greedily, converging early to a local optimum. Or, it may behave similar to a random walk, where it does not converge to any solution in the allowed time limit. In either case, computational effort is wasted on poor quality solutions (De Jong, 2007). Ideally, the algorithm should explore as much as possible before converging to high quality solution(s). See Figure 1.1.

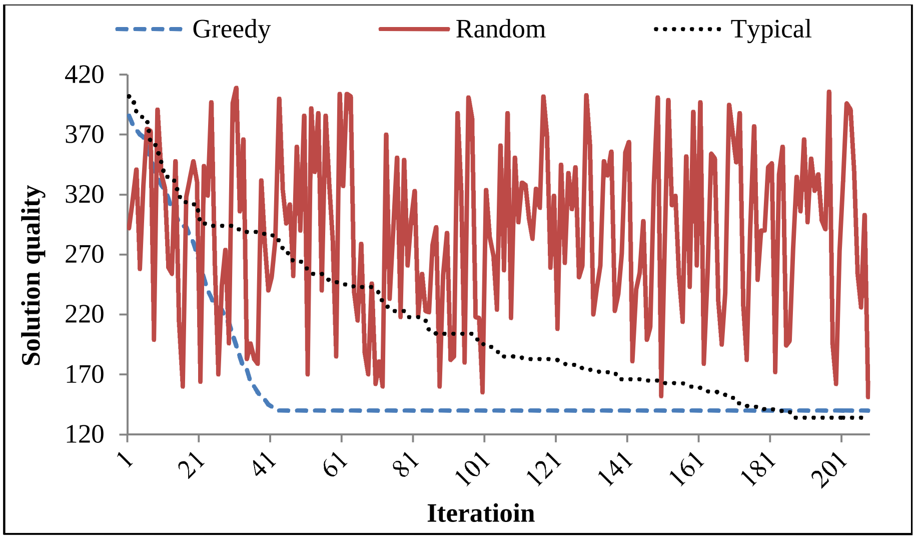


Figure 1.1. Effect of different parameter settings on an algorithm’s performance applied to a minimization problem.

Informally, the algorithm configuration problem is defined as: finding parameter values which enable the algorithm to reach its best possible performance within the available computational budget. Specifying what is "*best*" depends on the application, it could be, for instance, the solution quality obtained within a time limit (or computational budget), the running time needed to reach a target solution quality, or the percentage of how often a stochastic algorithm reaches a target solution quality under a budget constraint.

Hutter (2009, pp. 9-12) formally defines the problem as a 6-tuple $\langle A, \Theta, \Pi, \kappa_{max,} o, \tau \rangle$ where: A is a stochastic algorithm, Θ is the parameter space which includes the solution to the algorithm configuration problem, Π is the set of problem instances, often generated from a distribution D, κ_{max} is the maximum cap time an algorithm is allowed to run, o is the observed performance of the algorithm, and τ is a statistical measure (e.g. mean or median) of the observed performance that is to be optimized.

1.2. Solution approaches

Setting the parameters of an algorithm is not trivial. There are many challenges that require innovative methods to address them. This section first presents the major challenges, and then moves to a broad description of the solution approaches.

First, an algorithm can have categorical and/or numerical parameters (continuous and/or discrete), some of which can be conditional (i.e. they exist only if others do). For example, crossover rate and tournament size are only relevant for an EA if a crossover operator and a tournament selection are used, respectively. The parameter space is made up of all possible combinations of parameter values, and its landscape (defined later) is likely to be discontinuous and highly multi-modal, such that exhaustive search may not be feasible (Eiben and Smit, 2011).

Second, evaluating a parameter's quality, utility hereafter, on stochastic algorithms may require several replications, which raises the question of how many replications are needed to properly distinguish between the competing parameters. Moreover, utility values depend on the budget available for assessment. A greedy parameter setting can seem superior under a small budget compared to an explorative one. The situation may be reversed if more time is allowed for the latter, see Figure 1.2. This presents the challenge of specifying the budget needed to evaluate a parameter setting.

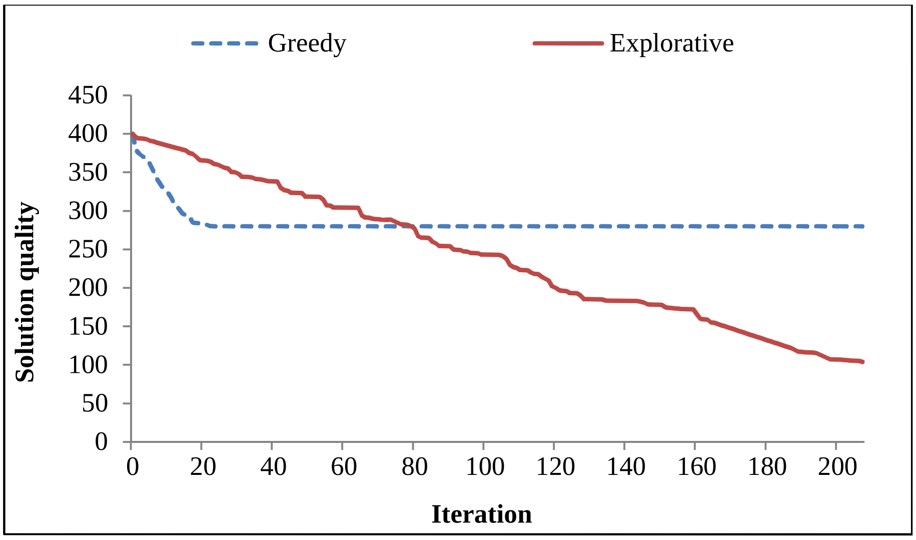


Figure 1.2. Different parameter settings are suitable for different budgets.
For a minimization problem, a greedy parameter is suitable for small budgets, but not larger ones.

Third, parameter settings which work well on one problem domain, routing for example, may not work for another, like cutting stock. The same issue extends across instances of the same domain if they differ greatly, or even within the same instance depending on its landscape. A problem domain encloses all instances (realizations of the problem) that share common similarities. Measuring similarities depends on the problem itself. For example, in a vehicle routing problem the number of nodes, their distribution (scattered, clustered, or random), and the number of vehicles can be used to distinguish between various instances. Smith-Miles and Lopes (2011) present several measures to characterize many real-world optimization problems, and assess their difficulty.

If the problem is to be solved only once, it is desirable to find a parameter setting which works best for that specific problem. Conversely, if similar problems, or instances, are to be solved repeatedly, a parameter setting which performs well overall might be preferred. Eiben and Smit (2011) refer to the earlier as a *specialist*, and the latter as a *generalist*. They also point out that finding a good generalist is by itself a multi-objective optimization problem, with each problem, or instance, being one objective.

Solving the algorithm configuration problem requires another algorithm, working at a higher level, and capable of searching the parameter space. EAs are an example. To distinguish between both, the higher level algorithm will be referred to as the *configurator* and the lower level algorithm as the *target algorithm*, see Figure 1.3 .The solutions of the configurator are parameter settings suitable for the target algorithm. Some configurators generate/modify parameter settings (e.g. through evolution as in EAs), while others only select the best out of a given set. Note that the configurator is likely to be stochastic as well, indicating that it, too, may require multiple replications.

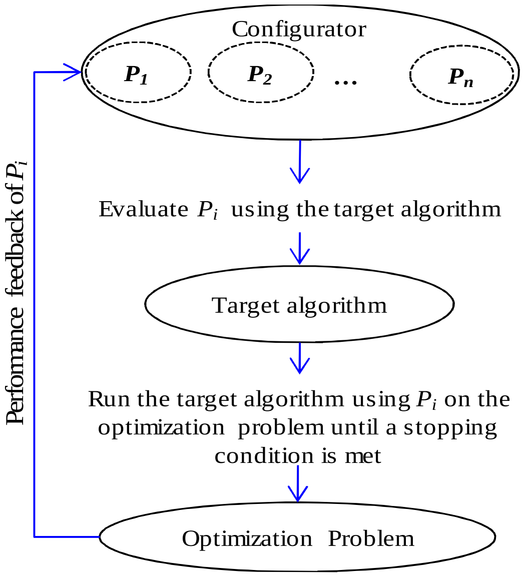

Configurator
P_1
P_2
...
P_n
Evaluate P_i using the target algorithm
Target algorithm
Run the target algorithm using P_i on the optimization problem until a stopping condition is met
Optimization Problem
Performance feedback of P_i

Figure 1.3. Configuring parameters with a higher level algorithm.

Assessing a parameter's utility requires running the target algorithm on the optimization problem until a stopping condition, for that parameter, is met. A stopping condition for a parameter can be, for example, a pre-specified number of iterations, the continued application until no improvement is observed, or the continued application until the target algorithm terminates. A performance measure is then fed back to the configurator and used to determine the utility. Note that more than one performance measure can be reported and combined in calculating a utility (e.g. running time and/or solution quality).

The stopping condition for a parameter determines the frequency of feedback to the configurator. One extreme is to report the performance only after the target algorithm terminates, in which case multiple assessments require multiple complete runs of the target algorithm on different instances of the optimization problem. The other extreme is to report performance after one or more applications of a parameter (an *operator*) before the target algorithm terminates, in which case the configurator gets feedback while the optimization problem is being solved.

The former case will be referred to as *offline tuning* as parameter settings are learnt after the problem is solved across many instances, and remain fixed. It is suitable for finding a generalist based on a training set of instances that is assumed to be large enough and representative, such that the best parameter setting found during training will also perform best on other, unseen, test instances. The latter case will be referred to as *online control* as parameter settings are learnt, and hence change, while solving the problem. It is suitable for finding a specialist.

Using online or offline rests on the application; does the problem require a specialist or a generalist? It remains unclear, however, if a generalist can perform just as well as a specialist, or if there is much to gain from online control over offline tuning. This issue will be further addressed in Chapter 2, where a comprehensive overview of the state-of-the-art configurators is presented, alongside a unique classification.

Finally, the term *Hyper-heuristics* was introduced just over a decade ago to refer to high-level methods specializing in generating or selecting lower-level heuristics, or heuristic components. The heuristics can be simple crossover or local search operators for example. They can also be parameter settings for algorithms. Hyper-heuristics have little or no problem domain knowledge, only information about the heuristics themselves; thus, hyper-heuristics should be able to find solutions for a wide range of problem instances (Cowling *et al.*, 2001). Constructive hyper-heuristics combine heuristic components to produce a new heuristic that is potentially able to solve a number of problem instances. Selective hyper-heuristics, however, select and apply a heuristic to the current solution in order to improve it. Comprehensive surveys on the topic are found in Burke *et al.* (2013) and in Chakhlevitch and Cowling (2008).

Almost any configurator can be viewed as a hyper-heuristic. In this work, though, the terms configurator, parameter setting/operator, and target algorithm will be used to describe a framework for the algorithm configuration problem.

1.3. Contributions

The contributions are methodological. In specific, new *efficient* configurators are introduced to tune algorithm parameters. This Section briefly describes each contribution, the motivation behind it, and a summary of its experimental assessment results.

1.3.1. Meta-Optimization with a Flexible Budget

The first contribution concerns offline tuning, where the configurator *modifies* its solutions to reach better ones. Such configurators are also known as *meta-optimization* methods. Meta-optimizers are computationally intensive as evaluating a single parameter setting requires a complete run of the target algorithm on the optimization problem. Also, several replications, for both levels, may be required for stochastic algorithms. Matters become worse if the computational budget for the target algorithm varies, because this will require a re-run of the whole meta-optimizer for each potential new budget. Figure 1.4 displays the performance of an algorithm, applied to a minimization problem, under three different parameter settings. It is clear that the best parameter setting depends on the length of the lower level run.

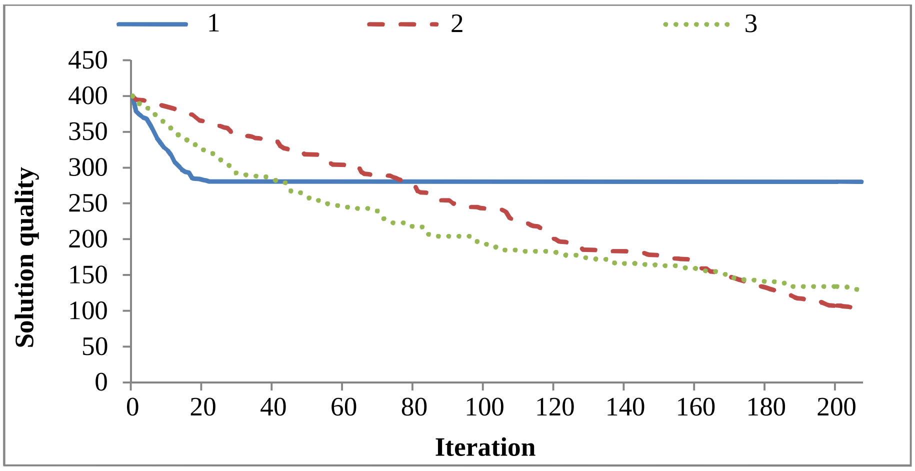


Figure 1.4. The performance of an algorithm, applied to a minimization problem, under three different parameter settings, each is suitable for a different computational budget.

This work proposes a new meta-optimizer capable of finding the best parameter settings for any computational budget less than a specified maximum in a *single* run, thus saving a lot of time. The algorithm is called Meta-Optimization with a Flexible Budget, or *Flexible Budget* for short. It makes use of the entire convergence curve of a parameter setting to calculate a rank-based utility.

It will be shown later that, for the experiments conducted here, the Flexible Budget method always finds the best parameter settings for any computational budget less than a specified maximum, while saving about 60%-70% of the computational effort required by the repeated application of the Fixed Budget method, *without* compromising solution quality. However, some experiments with a small budget using the quad function (defined later), showed minor solution quality degradation. Further details are in Chapter 4. Initial results of this method were presented in Branke and Elomari (2012).

A real-world application of the Flexible Budget is found in shipping. For next-day parcel deliveries, the daily work load varies, which, in turn, affects the time needed to prepare the parcels (e.g. inspection, weighing, sorting.). Such variability disturbs the running time available to a routing algorithm dispatching vehicles on the next day. Obviously, with only one to two hours at hand, the algorithm may perform better with a greedy parameter setting, while for longer running times, more explorative parameters are desired.

1.3.2. Racing with a self-adaptive significance level

The second contribution also falls under offline tuning where the configurator *selects* the best out of a set of parameter settings. There are a number of algorithms that specialize in efficiently allocating a computational budget (e.g. a training set) among several competing systems, the characteristics of which are unknown beforehand, such that the true best is selected when the budget is consumed. These algorithms guarantee a correct selection in the limit (i.e. if the budget is infinite), and converge to it at varying rates.

The term "*system*" is originally used in the Simulation Optimization literature to refer to the performance of a simulated system (e.g. a production plant), which is usually modeled with a probability distribution. Here, the term will be used to speak of the performance of an algorithm, or a parameter setting over various problem instances.

Racing is one such algorithm. It works by discarding inferior systems, or parameter settings, as soon as there is enough statistical evidence that they are significantly worse than the current best. In every iteration, the surviving systems are sampled once and the tests are run again. This continues until one system remains, or if the entire budget is consumed, in which case the system with the best performance measure is chosen as the winner. The algorithm's performance (e.g. the probability of selecting the best) is affected by the significance level α set by the user. A very high α results in less conservative tests and increases the probability of discarding the true best as the systems are dropped off more quickly. A very low α means more conservative tests, where the systems are hardly discarded and the algorithm samples all systems almost equally.

Note that Racing may terminate before the entire budget is consumed. In situations where there is a fixed budget constraint, such that there is no advantage of terminating the algorithm beforehand, a new Racing algorithm, *Racing with reset*, is introduced. Its basic idea is as follows: whenever a winner is identified and the budget is not entirely consumed, Racing with reset rolls back to the iteration where the first dropout occurred, lowers the significance level, and runs the tests on *all* systems. Consequently, if the best system was incorrectly discarded before, it now has another chance to survive in the race. The process is repeated as many times as needed until the budget is consumed. Racing with reset is thus able to consume any given budget and at the same time automatically adapt α.

It will be shown later that Racing with reset, at a relatively high α (say 0.3), always reaches a lower probability of incorrect selection (defined later), than its standard version with the best fixed α set by the user. Moreover, if the variances of the parameter settings' performance distribution are quite close, or they are highly correlated, its probability of incorrect selection converges to zero faster than one of the best budget allocators from the Simulation Optimization literature, namely the Optimal Computing Budget Allocation (OCBA) algorithm, which addresses a problem similar to the focus of this thesis. See Chen and Lee (2010). There are, of course, situations where Racing with reset is inferior (e.g. if the variances are exponentially increasing), such situations will be detailed in Chapter 5. Initial results of Racing with reset were presented in Branke and Elomari (2013).

1.3.3. One-way Racing with an intelligent budget allocation

All Racing algorithms currently used in the literature rely on a two-way Analysis of Variance (ANOVA) test, so as to account for the effect of the parameter setting (first factor), the effect of the problem instance (second factor), and their interaction. This, in turn, dictates having an equal number of samples for each of the competing systems whenever the tests are run, More importantly, every time a system is discarded, the remaining systems *must* all be sampled the same number of times (once is the default). This has two disadvantages, first, it does not allow for a more intelligent way of allocating the budget in each iteration. Second, it does not allow Racing with reset to use all previously collected data if the algorithm terminates before the budget is consumed.

A new Racing with reset algorithm is introduced here, which implements a one-way statistical test, the Kruskal-Wallis test (Kruskal and Wallis, 1952). The algorithm will be called *KW*-RaceR and can handle unequal sample sizes. To better allocate the budget in every iteration, instead of sampling all surviving systems equally, OCBA is run to determine the distribution of that iteration's budget, utilizing all previous knowledge. This combination allows for a different exploration vs. exploitation balance, compared to that obtained with equal allocation (the default). It will be shown that such a combination causes *KW*-RaceR to perform very similarly to OCBA.

1.4. Organization of the thesis

This thesis consists of six main chapters. Every chapter begins with an introduction section stating its objectives and expectations, and finishes with a conclusion section summarizing the main issues and findings. It is possible to grasp the main ideas of this thesis just by reading these two sections of each chapter.

The most relevant work to the developed methods are reviewed in Chapter 2, with the main objectives of positioning the before mentioned contributions within the literature, and highlighting some areas that received little or no attention, and are believed to represent future research topics. The theory and methodology of each contribution are detailed in Chapter 3. Chapter 4 empirically validates the first contribution by comparing it to the current methods in the literature and over a variety of scenarios. Chapter 5 does the same for the second and third contributions. Finally, Chapter 6 summarizes what was presented and proposes future research topics and extensions to this work.

CHAPTER 2 Literature Review

2.1 Introduction

The algorithm configuration problem has been recently advanced by different research communities, and now there exists a wealth of literature describing various solution approaches. Many of these approaches can be seen as extensions, or variations, of a few ideas, and can hence be grouped under a common umbrella. This chapter identifies two such key ideas, online control and offline tuning. It then organizes the current literature accordingly, along with a constructive critique when possible. This classification will assist in understanding how the current state-of-the-art methods have developed, and it helps position the work presented in this thesis.

The chapter is organized as follows: Section 2.2 describes the basis for the classification and the applications of each class. Section 2.3 reviews the literature on online control. Section 2.4 reviews the literature on offline tuning. Section 2.5 presents a few experimental studies comparing both classes. Section 2.6 points to some research areas which received little, or no, attention so far. Finally, Section 2.7 concludes this chapter.

2.2 Classification

Broadly speaking, algorithm configurators can be seen as online controllers, or offline tuners. The difference being whether parameters change while the target algorithm is solving the optimization problem, according to feedback from the search, or they remain fixed. In both cases these parameter settings have to be learnt; online methods do it on-the-fly, which makes them more likely to find specialized parameter settings for specific instances, while offline methods make use of a training set of problem instances, which makes them more likely to find generalized parameter settings suitable for instances similar to the training set.

Eiben *et al.* (1999) distinguished between three types of online control: deterministic, self-adaptive, and adaptive. Deterministic methods first learn a schedule of parameter-changes based on a training set, and then apply these changes while the target algorithm is running; hence, there is no feedback from the specific instance being solved. Still, they can be considered as online controllers as the parameters change. Self-adaptive methods are best seen in an EA context, where the parameter settings are encoded with the solutions and evolve to better values. Whitacre *et al.* (2006) pointed out that such methods increase the size of the search space, making the problem even harder to solve. Finally, adaptive methods use performance feedback gathered during the search to determine which operator to apply next.

A different sub-division of online control is proposed in this thesis, that is: single step look-ahead methods, where the operator which is best for the next immediate move is preferred, or multi-step look-ahead methods, where the operator which is best *n*-steps ahead is preferred. The latter assumes that while the immediate rewards of a chosen operator may be less than those of the others, the gain expected *n*-steps ahead outweighs such smaller losses.

Offline methods are sub-divided according to whether, or not, a model is built to guide the search for the best parameter setting. The model represents a relationship between the performance of an algorithm and a particular parameter setting. Model-free offline tuners are further sub-grouped into meta-optimizers, or computational budget allocators that efficiently distribute a training budget among the competing parameters. Figure 2.1 shows the proposed categorization.

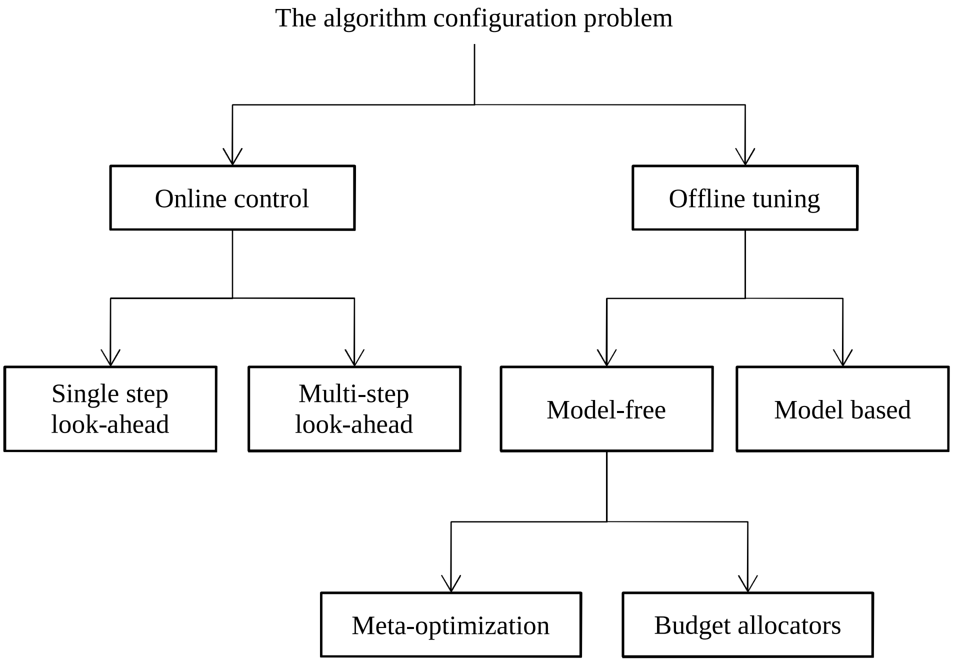


Figure 2.1. The proposed categorization of algorithm configurators.

The application determines whether to use online or offline. If one is interested in solving a single instance, a dynamic instance, or instances that are quite different from one another, then online control is expected to discover a specialist for that particular instance, without the need for the expensive training overhead required by offline tuning, which may not be feasible to begin with. On the other hand, if one is interested in solving many, similar, instances, then it may be meaningful to invest once in offline tuning and find a generalist that works well on many instances. In addition, offline tuning has the advantage of synthesizing algorithm components, or even entire algorithms through Genetic Programming or Grammatical Evolution.

2.3 Online control

2.3.1 Single step look-ahead

Adaptive Operator Selection (AOS) methods are composed of two stages: Credit Assignment (CA) and Operator Selection (OS). AOS applies an operator, from a set of operators, to the target algorithm, observes its effect, assigns a reward to that operator based on its performance, and finally selects an operator for the next iteration. The underlying assumption is that the credit assigned to an operator, up until the current iteration, is indicative of its future performance. Different CA and OS methods are available, and any combination can be used.

The most basic CA method is based on solution quality observed during the last iteration. A slightly advanced version considers the average quality, or a weighted average, over the last *n* iterations. Whitacre *et al.* (2006) were among the first to introduce the idea that operators which produce rare, but large, improvements could be more beneficial than those performing well on average. They setup an experiment to select the best out of ten EA operators using five CA methods based on average performance, and two based on extreme values. Results showed that the two extreme-value methods were significantly better than the rest. However, they pointed out that if extreme improvements are very rare, it might be better to focus on operators with steady improvements, especially towards the end of the run.

Extreme values were not always beneficial, for example Gong *et al.* (2010) applied four CA methods to a Probability Matching Differential Evolution with Adaptive Strategy Selection algorithm (**PM-AdapSS-DE)**, they were: absolute reward, average normalized reward, extreme absolute reward, and extreme normalized reward. PM-AdapSS-DE was compared to a uniform selection DE and the Self-adaptive Differential Evolution algorithm (**SaDE**) of Qin *et al.* (2009). PM-AdapSS-DE was able to outperform the others in terms of solution quality and convergence speed when using average rewards instead of extreme ones.

An operator can be credited for more than one performance measure. Maturana and Saubion (2008) combined quality variation, diversity variation, and application time in a single normalized credit, averaged over the last ω iterations. Their method, **Compass**, introduces its own hyper-parameters: the compass angle θ which determines how much weight the user assigns to quality variation or diversity variation, and the window size ω. Comparing Compass to Adaptive Pursuit (Thierens, 2005), APGAIN (Wong *et al.*, 2003), and random selection, the authors showed the superior performance of Compass for larger population sizes. For smaller ones, however, Adaptive Pursuit was able to better control the diversity and was eventually superior. Compass has been further extended to use extreme values for ω (Maturana *et al.*, 2009a, Fialho *et al.*, 2008), and to dynamically adapt θ during the search (Di Tollo *et al.*, 2011).

As for OS, a basic approach, known as **Probability Matching** (PM), assigns to each operator a selection probability equal to the operator's credit divided by the sum of the credits of all the operators. These probabilities are updated after each application of an operator. To guarantee that no operator receives a 0 (respectively, 1) selection probability in the long run, a minimum selection probability P_{min} is usually enforced → $P_{max} = 1 – P_{min}(k - 1)$, where k is the number of operators. Hence, all PM-based techniques have the additional hyper-parameter P_{min} that needs to be set or controlled.

PM techniques dominated the literature for some time, despite their limitation (Thierens, 2005). When the number of operators is small and their credits are quite close, PM will assign almost similar selection probabilities to all operators, making it hard to capitalize on the best. This was first observed in Thierens (2005), where an alternative method, **Adaptive Pursuit** (AP), was proposed. In AP, the best operator gets its selection probability increased while all the others get their selection probabilities decreased, yet remaining within the [P_{min}, P_{max}] interval. AP introduces the hyper-parameter β to control the greediness of the algorithm. AP outperformed PM and random selection on a test set simulated from a non-stationary process. It was unclear, though, if this performance scales with the number of operators. An extended AP, which pursues a set of operators, is in Drugan and Thierens (2011).

A Self-adaptive Particle Swarm Optimization algorithm (**SLPSO**) was proposed in Wang *et al.* (2011), where PM was used to select between four velocity update strategies: Comprehensive Learning PSO (CL-PSO) which has exploration abilities, a modified version of CL-PSO which has exploitation abilities (CL-PSO-pbest), Difference based Velocity update strategy (DbV) for rotated optimization functions, and Estimation based Velocity update strategy (EbV) for unimodal optimization functions. SLPSO performed significantly better compared to 8 other PSO algorithms on 26 optimization functions with characteristics, such as rotation, noise, ill-conditioning, and multi-modality. The authors did not force a P_{min} and they did not comment on how this might cause the search to “ignore” exploring all the strategies.

Maturana *et al.* (2009b) added a new level, **BLACKSMITH**, to select and/or create new operators for an EA. For credit assignment the authors used three performance metrics: the Pareto Dominance (PD) which measures the number of operators dominated by another operator, the Pareto Rank (PR) which measures the number of operators which dominate a given operator, and Compass (discussed earlier). These CA methods were combined with several OS methods, such as: random selection, PM, and Multi-Armed Bandit (explained shortly) to select from a set of 307 operators for the Boolean satisfiability problem (SAT). The authors found that PD-PM and PR-random offer the best combination of CA and OS in terms of solution quality.

A challenge in online control is to determine when to stop applying an operator. Ideally, one would like to stop when there is little chance of improving performance. Techniques from Optimal Stopping and Multi Armed Bandit (**MAB**) can be used to address this issue, see Hill (2011) and Gittins *et al.* (2011) for introductions on each respectively.

Knowing when to stop does not say much about which operator to use next; nonetheless, it has shown to improve performance of the target algorithm.

Bontempi (2011) attempted to determine the maximum number of function evaluations allocated to a local search operator using the Odds Algorithm of Bruss (2000), which returns a step value *s* such that if one stops at the first success encountered after *s*, the probability that this is the last success is maximized. A success in this case is reaching a local optimum. Compared to a fixed strategy (i.e. a fixed number of function evaluations allocated to the local search operator), the Odds algorithm found better local optima.

A MAB consists of a number of arms *K* each having a stationary binary reward (0 or 1). After each arm is pulled the player gains an expected reward μ_k and, in a full information game, is able to observe the expected rewards that could have been gained by pulling the other arms; hence, a regret can be calculated as $r = \mu_{k^*} - \mu_k$, where k^* is the arm with the best reward. A player's goal is to minimize the regret (or maximize the reward) over a number of *n* trials.

An optimal policy to the MAB problem was presented in Gittins (1979), which uses a set of indices, known as Gittins Indices, that are calculated at each trial for each arm, and the arm with the largest index is pulled. Further details about how to calculate these indices can be found in Gittins *et al.* (2011). One condition under which this policy is optimal, is the assumption of an infinite time horizon (i.e. an arm can be pulled indefinitely), which may not apply for parameter configuration.

Many variants of the MAB problem have been studied, examples include: Dayanik *et al.* (2008) where the arms are not always available and may break down (repair is possible at a cost), dynamic reward distribution (Whittle, 1988, Weber and Gideon, 1990, Bertsimas and Niño-Mora, 2000, Kleinberg *et al.*, 2010), continuous arms (Bubeck *et al.*, 2011), and correlated arms (Rusmevichientong and Tsitsiklis, 2010). In the latter study, the authors utilized the correlation to find a policy which scales better, in terms of regret, with the number of arms (possibly infinite), compared to other traditional policies that assume independence.

A commonly used MAB algorithm is the Upper Confidence Bound (**UCB1**) (Auer *et al.*, 2002, Auer and Ortner, 2010), which selects at each time *t*, the arm with the highest

$$q_{i,t} + \sqrt{\frac{2 \log \sum_{j=1}^{K} n_{j,k}}{n_{i,t}}}, \qquad 2.1$$

where the first term is the average empirical reward of arm *i* up until time *t*, and $n_{i,t}$ is the number of times an arm has been selected up to time *t*. The first term favors exploitation while the second favors exploration.

Viewing the operators as arms of a MAB, Dacosta *et al.* (2008) introduced a new OS method using the UCB1 algorithm. Realizing it cannot be used directly, because operators have non-stationary reward distributions, and their rewards are not binary, the authors used a distribution-change detection test, known as the Page-Hinkley test (Hinkley, 1971), such that when the test is triggered the quantities in 2.1 are reset to their initial values. Meaning, the algorithm “forgets” all it has learnt previously, allowing it to adapt to the new distribution. They also introduced a scaling factor *C* that is multiplied by the second term of 2.1 to normalize the different rewards assigned to the operators. This Dynamic MAB (**DMAB**) outperformed PM, AP, and UCB1 (the static version) when tested on the same dataset used in Thierens (2005).

Any combination of CA and OS can be used. A comprehensive comparison between various combinations can be found in Fialho (2010), and Fialho *et al.* (2010), where the authors also discuss the robustness of these methods against their hyper-parameters.

2.3.2 Multi-step look-ahead

Single-step look-ahead methods can be seen as greedy approaches to the problem, as an operator which seems inferior at the moment may lead to greater gains later on. Looking two steps ahead (i.e. considering the reward of operator *x* if it is applied after another operator *y*), Cowling *et al.* (2001) introduced a choice function that selects the next operator based on a weighted average of past performance, time elapsed since last application, and the pervious operator used. The first and third measures favor exploitation, while the second favors exploration. This differs from the Compass algorithm in that it considers the sequence in which operators were applied. The choice-function-based method performed significantly better than random and greedy selection on the sales summit scheduling problem.

Generalizing the choice function idea, Burke *et al.* (2007) used a TS algorithm to find the best sequence of operators constructing a solution for a timetabling problem. Their algorithm starts with a set of operators, applied in sequence to construct a complete solution. The set is then perturbed, checked against the tabu list, and evaluated based on solution quality. The process continues for a pre-determined number of iterations. Compared to nine other specialized algorithms, the authors showed that their algorithm never outperformed the best, but its results were within range of the others. This work was extended in Qu and Burke (2008) to include other algorithms at the upper level, namely: Iterated Local Search (ILS), Variable Neighborhood Search (VNS), and Steepest Descent (SD). ILS and VNS showed to be superior to the other methods when tested on the same data set.

Another multi-step look-ahead approach is Dynamic Programming (DP), in which a non-myopic control policy can be learnt to account for the future effects of selecting an operator. While this seems appealing, there are three difficulties: first, fully describing the state space is not trivial, especially if it is continuous. Second, the action-reward relationship maybe complex and could require approximation with a learning function. Third, the policy itself has to be learnt from a training set, which makes this approach an online deterministic one and can restrict its applicability to similar test instances. Such difficulties have been addressed by a few researchers, but generally speaking a DP-based approach has not yet been fully developed, and the experimental studies conducted thus far indicate that learning an optimal, or near optimal, policy is a hard task.

Nareyek (2004) proposed ten reinforcement learning strategies to update the weights of a set of heuristics, from which selection is made probabilistically. Five were used in case of solution quality improvement, and the remaining in case of a decline. Different combinations of improving and non-improving strategies were tested on the Orc Quest problem and a modified Logistic Domain problem[1]. The authors concluded that, generally, a low update rate of weights is better in case of an improvement, whereas a high rate is better in case of a decline.

In Battiti and Campigotto (2008), the authors modeled the algorithm configuration problem as a Markov Decision Process, and used the Least Squares Policy Iteration (LSPI) algorithm, adopted from Reinforcement Learning (RL), to learn a near optimal control policy through a training set. The policy was used to learn how to change the prohibition length parameter for the Reactive Tabu Search (RTS) algorithm of Battiti and Protasi (1997). Comparing RTS with LSPI-RTS revealed that LSPI-RTS could not outperform its non-adaptive counterpart; however, it was not significantly worse. Comparison to three other SAT solvers showed superior performance of LSPI-RTS.

Extending the above work in Battiti and Campigotto (2011), the authors compared three LSPI-based algorithms to their original versions using default, and tuned, parameter settings. Tuning was carried out with ParamILS (Section 2.4.1.2 later) of Hutter *et al.* (2009a). The algorithms were: Hamming Reactive Tabu Search (HRTS), Adaptive WalkSAT, and Scaling and Probabilistic Smoothing (SAPS) together with its Reactive version (R-SAPS). When testing against default parameter settings, the LSPI-based HRTS,

[1] Detailed description of both problems can be found in Nareyek, A., 2001. *Constraint-based agents: an architecture for constraint-based modeling and local-search-based reasoning for planning and scheduling in open and dynamic worlds.* 1st ed. Berlin: Springer.

SAPS, and R-SAPS showed superior performance on a set of benchmark random MAX-3-SAT problems; however, the LSPI-based Adaptive WalkSAT performed worse. Repeating the same comparison on other structured MAX-SAT industrial instances, the control policy degraded the performance of SAPS and R-SAPS, while the other two algorithms performed just as well or slightly better. When testing against tuned parameter settings, LSPI-based SAPS was not significantly worse than SAPS$_{offline}$, and the LSPI-based HRTS was either significantly worse than HRTS$_{offline}$ or just as good. The authors contribute the inferior performance of the LSPI-based methods to its inability to converge to a single policy during training.

Mcclymont and Keedwell (2011) attempted to learn the best sequence of applying a number of operators. Their method, Markov Chain Hyper-heuristic (**MCHH**), uses Markov chains to model the transition probabilities between operators, and RL to update the transition probabilities online based on the operators' performance. MCHH was applied within a (1+1)-Evolutionary Strategy (ES) to determine which variation operator to select next. The authors compared a (1+1)-ES using MCHH to random selection and a prohibition-based algorithm proposed by Burke *et al.* (2005). They found their algorithm to be superior to random selection, and it performed just as well as the prohibition-based one. However, the authors noted that while their method converges at a reasonable rate, higher population diversity is still needed.

Gaspero and Urli (2012) created three RL-based heuristic selection policies: tabular RL, where the possible actions and rewards are relatively few and can be read from a table; Multi-layer Perceptron RL, where the actions and rewards are related through a function that is approximated with Artificial Neural Networks; and Eligibility Traces RL, where each action on the trajectory to a reward is assigned a share of that reward, not just based on the last action. Each of the policies introduced its own hyper-parameter which the authors set using the offline tuner *F*-Race (Section 2.4.1.2 later). However, all three policies performed rather poorly when tested on the Cross-domain Heuristic Search Challenge[2] (CHeSC) test-bed, see Burke *et al.* (2011), due to the limited number of training instances available to learn the policies.

A slightly different approach to determine which operator(s) to apply for the next *n*-steps is landscape analysis, which seeks to characterize the topology surrounding the current solution. Proper characterization of the landscape can offer guidance to whether the algorithm should diversify or intensify its search, and select its operators accordingly. Introductions to the topic can be found in Merz and Freisleben (2000), Kallel *et al.* (2001), and Battiti *et al.* (2009).

Ideally, the algorithm should use problem independent measures to characterize the landscape. One such measure is ruggedness, first introduced by Weinberger (1990), which calculates the autocorrelation of a time series of fitness values generated by a random walk algorithm. A related measure is the correlation length, which estimates the largest distance, or time lag, between two points at which the value of one point can still provide information about the expected value of the other one (Hordijk, 1996). Put differently, it estimates the largest time lag *t* for which one can still expect some correlation between two points, *t* steps apart.

Another common measure is the fitness distance correlation, first introduced by Jones (1995), which measures the extent to which solution quality values are correlated with the distance to the global optimum. Additional measures include: the number of global and local optima, the distribution (uniform or random) of the global and local optima, and the structure of the basins of attraction (Weinberger, 1991). These measures have also been extended to multi-objective optimization applications, see Deb (1999), and Knowles and Corne (2002).

[2] http://www.asap.cs.nott.ac.uk/external/chesc2011/

Problem specific measures have also been created. Smith-Miles and Lopes (2011) review such measures for assignment, routing, knap-sack, bin packing, graph coloring, and timetabling problems. Unfortunately, real optimization problems are highly dimensional and are very complex to analyze or measure. This limits landscape analysis to simple and artificial landscapes such as the *N*-*k* landscape (Kauffman and Levin, 1987). Early attempts to measure the relationship between solutions within an *N*-*k* landscape (using the auto-correlation function and the auto-correlation length) can be found in Weinberger (1990), and Hordijk (1996). Recently, ideas based on complex network analysis, which not only relates solutions within the same landscape, but also relates different landscapes to each other, can be found in Ochoa *et al.* (2008) Tomassini *et al.* (2008), and Ochoa *et al.* (2011).

2.4 Offline tuning

2.4.1 Model-free methods

2.4.1.1 Meta-optimization

Model-free tuners can work by introducing a higher (meta-) level search algorithm, operating on the parameters' search space. The meta-level is another optimization algorithm, where a single solution (parameter setting) is evaluated by running the lower-level algorithm on the optimization problem, until a stopping condition is met. Performance is then fed back to the meta-level to improve the current solutions. See Figure 2.2.

Early attempts to meta-optimization can be traced back to Mercer and Sampson (1977), who used a Genetic Algorithm (GA) to find best mutation and crossover rates for another lower-level GA. However, due to the computational limitations at that time, their experiments were small in scale and constituted of a single run of the lower-level. The same approach was conducted in Grefenstette (1986), but on a slightly larger scale (20 generations at the meta-level and 2000 function evaluations at the lower-level). Selection at the meta-level was based on average solution quality obtained from the lower-level.

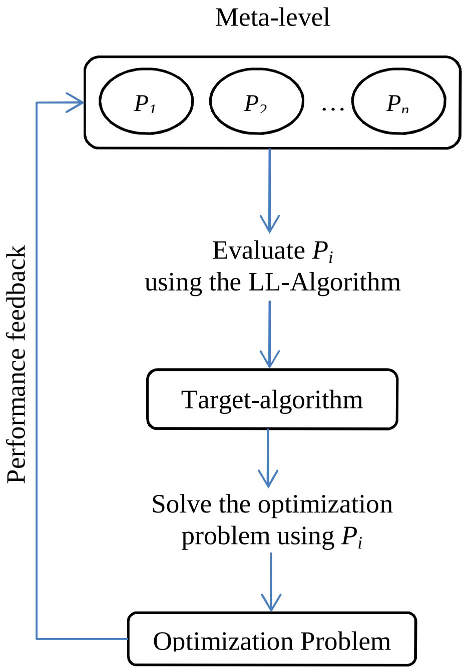


Figure 2.2. Meta-optimization framework.

Later, Shahookar and Mazumder (1990) used a meta-GA to find best values for mutation, crossover, and inversion rates required for a lower-level GA solving the Standard-cell Placement Problem. They compared the tuned GA to the TimberWolf placement software package (Sechen and Sangiovanni-Vincentelli, 1985), and showed that the tuned GA was able to reduce the number of evaluated configurations. A similar approach to tuning ACO parameters using a meta-GA can be found in Botee and Bonabeau (1998) and in White *et al.* (1998), where the authors compared the tuned ACO to another using the default parameters published in the literature, on a set of Traveling Salesman Problem (TSP) instances, and a set of path finding instances. Results favored the tuned ACO in terms of convergence speed in both studies.

Meta-optimization methods performed well on dynamic problems as well. Stanhope and Daida (1998) used a meta-GA to find optimal crossover and mutation rates for a lower-level GA, solving a simple dynamic fitness function changing at pre-determined rates, which the authors created. Their experiments showed that as the change in the fitness function increases, the meta-GA converges to higher mutation rates and lower crossover rates, and vice-versa. This supports the general notion that for highly dynamic problems, utilizing information learnt from the past is not the best policy.

A slightly different application can be seen in Cortez *et al.* (2001), where the authors used a meta-GA to determine the best coefficients of a lower-level Autoregressive Moving Average (ARMA) model. The Root Mean Squared Error was used as a performance measure of the ARMA model. They compared the tuned ARMA model to Exponential Smoothing, tuned using grid search, and an un-tuned Autoregressive Integrated Moving Average (ARIMA) model. Tests on a number of forecasting instances with different characteristics (seasonal and trended, seasonal, trended, and non-linear) showed that Exponential Smoothing performed best on seasonal instances. However, the tuned ARMA performed best on non-linear instances especially when trend components are present.

Stephenson *et al.* (2003) used a meta-EA to combine/create new compiler heuristics, from a set of available heuristics. The idea was to evolve a general purpose heuristic applicable to a wide range of problems. They tested their method on three case studies from computer architecture, and were able to at least match the performance of human-generated heuristics, and in some cases achieved considerable speedups. Still, the authors noted two drawbacks in the approach: first, over-training the meta-optimizer caused degradation in performance on the test cases; even though they at least matched the human-generated ones. Second, there is a limit to the diversity of test cases to which the general compiler heuristic can be applied to.

Particle Swarm Optimization was used as a meta-optimizer in Meissner *et al.* (2006). The authors introduced the concept of Optimized Particle Swarm Optimization (OPSO), to optimize the free parameters of a PSO applied to neural network training. The tuned PSO was compared to a standard PSO and the Constriction type PSO (Clerc and Kennedy, 2002), both un-tuned, on a set of five artificial fitness functions. Results indicate that the tuned PSO outperformed the other methods in terms of solution quality and running time.

Pedersen and Chipperfield (2008c) tuned the parameters of Differential Evolution (DE) algorithms which optimize a number of Artificial Neural Networks (ANN) weights. The meta-algorithm in their study was the Local Unimodal Sampling (LUS) (Pedersen and Chipperfield, 2008a). Briefly, the method adds a random number to the current solution to obtain a new solution. The random number is initially drawn from the search-range, which decreases as the search progresses. Four DE variants were compared to a Back Propagation (BP) algorithm, which is a gradient-based method as opposed to DE. The results were in favor of BP, even though the differences in performance were minor. This could be attributed to the small training set the authors used to tune the DE algorithms; one training instance vs. four test instances. It is also possible that BP is simply superior for these kinds of problems, as it is gradient-based.

Meta-optimization has been extended to evolve certain algorithm components (e.g. pheromone update rules in ACO, or survival selection methods in EAs) using Genetic Programming (GP) (Koza, 1992), or algorithms as a whole using Grammatical Evolution (GE) (O'neill and Ryan, 2003).

With primitives like selection operators, variation operators, and replacement methods, Oltean (2005) applied linear GP to evolve new EAs for various optimization functions. They also evolved EAs for the TSP and the Quadratic Assignment Problem (QAP). The evolved EAs were compared to standard GA and ES algorithms from the literature, and performed just as well, or worse than, the standard algorithms over most instances. Superior performance was observed on only few instances. Likewise, Ross (2002) evolved three main algorithm classes for the TSP using GP, they were: hill climbing, annealing, and genetic algorithms. A new crossover operator was discovered as well. The evolved algorithms had different structures than their counterparts in the literature, and performed just as well, but not better. In both studies, the parameters of the evolved algorithms were not tuned. Instead, the default values of the competing algorithms were used, which may have degraded their performance.

Poli *et al.* (2005) evolved new position, velocity, and acceleration equations for PSO using GP. The evolved equations outperformed the ones in the literature, in terms of distance to the optimal solution, when tested on the Rastrigin and City Block Sphere functions. Similarly, Runka (2009) created new probabilistic decision formulas for ACO, they were of the type: roulette wheel, greedy, and tournament selection of size seven. None of them was an exact match to the default formulas in the literature, although tournament selection was the closest. Comparing the performance of ACO using the new formulas to that when using the default, the average performance, in terms of closeness to the optimal solution, was significantly better for the developed roulette wheel and greedy formulas, on two-thirds of the test instances, and for the remaining one-third the performance was either the same or worse. Tournament selection performed worse on all instances. It is worth noting that the developed formulas were parameter-free as GP set those parameters as part of its evolution.

Tavares and Pereira (2011) used Strongly Typed Genetic Programming (STGP) to evolve pheromone update strategies for ACO. The evolved strategies were competitive to the default, human-designed, ones used in the Max-Min Ant System (MMAS) when tested on the QAP. The authors also carried out a cross-problem validation for the evolved strategies; in specific, pheromone update strategies evolved using a TSP training set were tested on a QAP test set and vice-versa. Over most QAP instances, the strategies evolved for the TSP were competitive to the MMAS strategy; yet, they were inferior to specifically evolved strategies for the QAP. In the reverse experiment, strategies evolved for QAP perform worse than the MMAS strategy when tested on the TSP. All evolved strategies used un-tuned parameters.

Tavares and Pereira (2012) extended their previous work by evolving complete ACO algorithms using GE. The algorithm components were taken from human-designed ACO algorithms. Two important results came out of this study: first, none of the evolved algorithms exactly matched any of the currently used ACO variants. Nonetheless, three algorithms were similar to Ant Colony System, one was similar to Rank-based Ants System, and one was similar to Elitist Ants System. Second, the meta-algorithm did not converge to a single ACO algorithm, but some components were selected more often than the others. This is probably due to the small training set used (only three TSP instances). Upon testing, the best evolved algorithms performed significantly better than the human-designed ones, with one exception where the Rank-based Ants System performed best on a single instance. The authors did point out that the language used for the meta-algorithm was restrictive, and relaxing this restriction allows for more innovative algorithms, at the expense of convergence speed.

It should be noted that evolving algorithm components, or entire algorithms, depends largely on the function and terminal sets (in GP), or language (in GE), made available to the meta-level algorithm. Having a general language which includes all possible functions and terminals is not feasible; therefore, knowledge of the algorithm class which needs to be developed, and its application, comes a long way in properly designing the meta-level.

2.4.1.2 Computational budget allocators

Computational budget allocators take, as an input, a number of algorithms k, or the same algorithm with different parameter settings, a problem instance generator I, and a computational budget N. The objective is to allocate N among the $\langle algorithm, instance\rangle$ pairs, such that the best algorithm is correctly identified when the budget is consumed. Running algorithm i on an instance j returns a utility U_{ij}, and aggregating all the utility values gives one measure, which can be used to compare between the different algorithms and select the best.

Each algorithm is first run on the same subset of problem instances n_0 to obtain an estimate of its performance. Then, the budget allocator determines how to distribute the remaining budget, or a portion of it as allocation is usually made sequentially. For stochastic algorithms, replicating an $\langle algorithm, instance \rangle$ pair may be necessary; however, Birattari (2005) showed that performing a single on each instance (hence testing on more instances under the same budget) produces a smaller variance of the aggregated measure compared to replicating. While the allocation itself happens online, the algorithm solving the problem instance does not adapt itself during the search, this is why these methods are classified as offline tuners. See Figure 2.3.

Problem instance or sample number	Parameter settings or systems				
	1	*2*	*3*	*4*	*5*
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	1	1	1
4	1	1	1	1	1
5	1	1	1	1	1
6	2	3	2	3	5
7	2	3		3	5
8	2	3		4	5
9	2	4		4	5
10		4		4	5
…		…		…	…
m					

Figure 2.3. An example of allocating a budget between five parameter settings with n_0 = 5. The numbers indicate the iteration in which the runs were made. For instance, parameter settings 1 and 3 were run in the second iteration with parameter setting 1 being tested on instances 6-9, while parameter settings 3 was run on instance 6.

Racing is a group of algorithms which follow the above mentioned framework. Maron and Moore (1994) introduced the first Racing algorithm to solve the model selection problem in Machine Learning. Racing applies a two-stage statistical test, to identify algorithms/parameter settings whose performances are significantly worse than the current best, and discard them from the race. The first stage only detects if at least one algorithm is significantly different from the rest, using tests such as ANOVA, for normally distributed data, or ranked-based ANOVA (e.g. Friedman's *F*-test or the Kruskal-Wallis test), for non-normal data. Racing algorithms are named according to the corresponding ANOVA test (e.g. *F*-Race).

If the first test is significant, the second stage is carried out to identify those inferior algorithms, using tests such as the paired *t*-test, for normally distributed data, or any non-parametric post-hoc test for non-normal data, see Conover (1999, pp. 369-371) and Sheskin (2004, pp. 830-850) for various suggestions. The race ends when one algorithm remains, or the computational budget is consumed. As Racing is one of the main focal points of this thesis, its technical details will be elaborated on in Chapter 3. For now, the focus is on its application to tune algorithm parameters.

Rossi-Doria *et al.* (2003) compared the performance of five different metaheuristics on timetabling problems. To allow for a fair comparison, all algorithms had their parameters tuned with *F*-Race. Results showed that even for a very specific problem domain, no one algorithm was able to outperform the rest on all instances. A similar application of *F*-Race can be found in Paquete and Stutzle (2006) where the authors tuned the parameters of five metaheuristics for the QAP, and compared the results to those obtained by the same algorithms with default parameter settings. Results favor the tuned algorithms in almost all cases.

Balaprakash *et al.* (2009) improved the performance of an estimation-based local search algorithm, by combining heuristically two variance reduction techniques: Importance Sampling and Adaptive Sampling. The authors applied Iterated *F*-Race (Birattari *et al.*, 2009) to tune the parameters of several Importance Sampling variants. In the same fashion Racing algorithms have been used to tune different algorithms applied in various fields, examples include: Blum and Socha (2005) in neural network training, Chiarandini and Stutzle (2007) in Graph coloring, Gaspero *et al.* (2007) in portfolio selection, and Di Gaspero and Roli (2008) and Lenne *et al.* (2008) in bioinformatics.

Racing has indirectly been used in parameter tuning, specifically for selection within metaheuristics. Yuan and Gallagher (2007) used *F*-Race to reduce the computational cost of running a (1+λ)-ES tuning a number of numerical and categorical parameters of a GA. Instead of evaluating all λ individuals equally to find the best one, *F*-Race was used to efficiently allocate the computational budget to the most promising individuals. Likewise, *F*-Race was combined with ACO (Birattari *et al.*, 2007), Mesh Adaptive Direct Search (Yuan *et al.*, 2010), Bound Optimization By Quadratic Approximation, Derandomized Evolution Strategy with Covariance Matrix Adaptation, and Uniform Random and Iterated Random Sampling (Yuan *et al.*, 2012).

F-Race can only select from the initial set of parameter settings provided to it. If a better parameter setting exists for that algorithm, and it was not included in the initial set, it will not be discovered. To overcome this issue, Chiarandini *et al.* (2006) proposed inserting a new parameter settings into the race in each iteration. This parameter setting is first tested on as many instances as the others, and then the statistical tests are carried out. Another exploration mechanism can be found in Birattari *et al.* (2010), where the authors created an iterative version of *F*-Race (*I\F*-Race). Each iteration is a single race, and with each iteration the initial set of parameter settings is biased towards the best. In their implementation, biasing was done in a manner analogous to an Estimation of Distribution Algorithm (EDA) working at a higher level.

Yuan and Gallagher (2004) compared a Racing algorithm using a normal-model ANOVA, *A*-Race, with *F*-Race. They observed better performance for *F*-Race in terms of identifying the best parameter setting (which was determined by exhaustive experimentation), and the number of survivors at the end of the race. A possible explanation for the poor performance of *A*-Race is that the assumptions required by a normal-model based ANOVA were violated (e.g. normality, independence, and homoscedasticity). Moreover, *F*-Race implements a blocking design, and it was not clear if the same was used for *A*-Race. More details in Chapter 3.

A more extensive comparison of different budget allocators was carried out by Caelen and Bontempi (2005). Five methods, from different research communities, were experimentally evaluated. They were: a two-stage selection from simulation optimization (Dudewicz and Dalal, 1975), the MAB, Equal Allocation, Greedy Allocation, and F-Race. While no single allocator outperformed the rest in all test cases (model selection), the following general observations were made: the two-stage selection method performed worst in most cases due to its independence-of-models assumption, MAB performed best in most cases when the available budget is low, and finally F-Race performed best when the available budget is high.

Another budget allocator is Parameter Iterated Local Search (**ParamILS**) (Hutter *et al.*, 2007b, Hutter *et al.*, 2009a), which uses ideas from the Iterated Local Search algorithm (Lourenco *et al.*, 2002) to search for good parameter settings. It begins with a set of parameter settings s_0 consisting of the algorithm's default, or user specified, and a random sample from the configuration space. Then, s_0 is evaluated on a sub-set I_0 of the training instances, using a part n_0 of the total computational budget N. Depending on n_0 multiple replications maybe carried out. The best parameter setting s_0^* is further improved via local search (e.g. one-exchange neighborhood). ParamILS next enters its main optimization loop, where s_0^* is randomly perturbed and improved via a local search procedure. If the resulting parameter setting improves over s_0^*, it is accepted as the initial starting point for the following iteration. This continues until a termination condition is met. A diversification mechanism of ParamILS is to draw a random parameter setting, with a probability of p, and use it as a starting point for its optimization loop.

ParamILS has two variants: **BasicILS** and **FocusedILS**, which differ mainly in the way the training budget is allocated between the competing parameter settings. While BasicILS allocates the budget equally, FocusedILS does a more efficient job by focusing on more promising configuration, and spending less effort on the poor performing ones. The idea of FocusedILS is similar to that of Racing, except that allocations are determined heuristically rather than statistically. Hutter *et al.* (2007a) used FocusedILS to tune 27 parameters of the SAT algorithm SPEAR (Babic and Hu, 2007), solving a number of real-world bounded model-checking and software verification instances. The tuned algorithm was able to significantly reduce the average run time of the standard, hand-tuned, algorithm by many folds.

Khudabukhsh *et al.* (2009) created *SATenstein*, which is a framework to design stochastic local search algorithms for SAT problems. It has just over 40 parameters which were set using FocusedILS, enabling it to create new SAT solvers outperforming current state-of-the-art algorithms. Later Xu *et al.* (2010) combined SATenstein with the per-instance algorithm selector SATzilla (Xu *et al.*, 2008), in what the authors call *Hydra*. Tuning Hydra with FocusedILS enabled it to outperform 16 state-of-the-are SAT solvers in terms of solution quality, and was, on several occasions, able to do so in less than a third of the time required by its competitors.

Hutter *et al.* (2010b) used FocusedILS to tune the parameters of the complete mixed integer programming solvers LPSOLVE, CPLEX, and GUROBI. They were able to lower the time required to reach optimal solutions, by a maximum of 52 folds, and reduce the gap between the best solution and the optimal one (when there is a time constraint) by a maximum of 45 folds. However, they did note that such gains come at the cost of large training instances.

FocusedILS was extended in Fawcett *et al.* (2009) to take advantage of unimodal parameter response surfaces. Specifically, the local search step now searches through solutions that are only adjacent to the current one. The extended version was used to tune a highly parameterized solver for the course timetabling problem, which the authors created, and outperformed other algorithms in the 2007 International Timetabling Competition[3]. Another extension can be found in **CluPaTra** and **SufTra** (Lindawati *et al.*, 2011,

[3] http://www.cs.qub.ac.uk/itc2007/index.htm

Lindawati *et al.*, 2013), which groups problem instances into clusters with similar search trajectory patterns[4], and then applies ParamILS. The authors showed that this method finds better parameter settings per cluster compared to those obtained without clustering.

The Ranking and Selection literature has numerous algorithms solving problems similar to the algorithm configuration problem. For instance, a number of stochastic simulation systems are made available along with a computational budget. Since evaluating a system is expensive, it is crucial to efficiently allocate the budget such that the true best is selected when the entire budget is consumed. It is beyond the scope of this thesis to review such a huge literature, and the interested reader is referred to the works of Kim and Nelson (2006), Branke *et al.* (2007), and Kim (2013) for an overview. Instead, the focus will be on two methods, namely: the Optimal Computing Budget Allocation (**OCBA**) (Chen, 1995, Chen *et al.*, 1997), and its correlated version Correlated Budget Allocation (**CBA**) (Fu *et al.*, 2004, Fu *et al.*, 2007).

Similar to Racing, OCBA/CBA first estimates the performance of all k systems using $k \cdot n_0$ samples, and then it allocates the remaining $N - k \cdot n_0$ samples all at once, or over several iterations using in each iteration a portion Δ of $N - k \cdot n_0$. Unlike Racing, however, OCBA/CBA never discards a system, it may choose not to sample some systems for a while, but all systems are candidates for sampling at any iteration. The difference between OCBA and CBA is that CBA is designed to account for correlation between the systems, while OCBA assumes independence. Still, both methods require the normality assumption. The detailed workings of both methods will be presented in Chapter 3, and the focus here is on their applications.

OCBA has been developed and applied more than CBA, examples include: comparing alternative system designs in order to select the one with the best performance (Chen *et al.*, 2000, Brantley *et al.*, 2008, Chen *et al.*, 2010), selecting the m-best designs

[4] A search trajectory pattern is defined as the path that an algorithm follows as it iteratively moves from an initial solution to a neighboring one.

(Chen *et al.*, 2008), or selecting the one that optimizes multiple objectives simultaneously (Lee *et al.*, 2004, Loo Hay *et al.*, 2007, Chen and Lee, 2009, Teng *et al.*, 2010). Extending OCBA to handle correlated data, with *known* correlation structure, was done in Fu *et al.* (2004), and Fu *et al.* (2007). The method was further improved in Qu *et al.* (2012) to learn the *unknown* correlation structures. OCBA has also been modified to work with non-normal data in Glynn and Juneja (2004), and with heavy-tailed distributions in Blanchet *et al.* (2008).

Moreover, OCBA has also been integrated into other algorithms to improve their performance. He *et al.* (2010) combined a new variant, OCBA-CE, with the extended Cross Entropy algorithm to select the top *m* designs *and* accurately estimate their performance. Shortle and Chen (2008), Shortleab *et al.* (2012) used OCBA to efficiently allocate a simulation budget such that the variance of the estimate of the probability of rare-event simulations is minimized. A comprehensive overview of these applications, and others, were recently published in the book of Chen and Lee (2010).

Neither OCBA nor CBA have been directly applied to tune parameters, and they were never compared to budget allocators designed for the algorithm configuration problem. However, OCBA was used within the algorithm configurator SPOT, see Section 2.4.2, to make it run more efficiently. OCBA and CBA will be tested against various versions of Racing algorithms to see under which conditions each performs best.

2.4.2 Model Based Methods

Methods under this category work in a sequential manner. First, a set of parameter settings are selected and evaluated to build an initial model of the algorithm’s performance. The model is used to find the best parameter setting observed, or predicted. Second, based on the model, a new set of parameter settings are sampled and evaluated to *update* the current model, and find a new best parameter setting. The process continues until a stopping condition is met. This general framework is known as Sequential Model Based Optimization (**SMBO**).

Coy *et al.* (2001) used a 2-level full factorial designed experiment to construct a linear response surface, representing the performance of two heuristic algorithms for the VRP. Using steepest descent, the authors discovered new parameter settings which enabled both heuristics to outperform other algorithms. However, this approach consisted of a single stage, which means that the initial model built was not updated.

A similar approach, based on Taguchi’s fractional designs combined with local search, was presented in Adenso-Diaz and Laguna (2006). As opposed to the previous approach, **CALIBRA**, updates its model over several stages and fits a non-linear surface when necessary. While CALIBRA showed improved performance, when used to tune six heuristics, compared to their default parameters, it is limited to only a small number of parameter settings, and does not analyze interaction effects, it is, therefore, more effective in situations where the interaction effect is negligible.

Gagliolo *et al.* (2004) created an Adaptive Online Time Allocator (**AOTA**), which uses algorithm runtime and problem features, to build and update a simple linear regression model. Additional runs are allocated to the promising algorithms/parameter settings based on extrapolating their performances on the next instance. AOTA was able to find algorithms which outperformed specialized GAs designed for SAT instances. This work was extended in Gagliolo and Schmidhuber (2006), and Gagliolo and Schmidhuber (2011) by introducing a MAB solver at a higher level, which selects from a set of eleven allocators including uniform and greedy. This method, **GAMBEL**TA, showed superior performance compared to a uniform allocator and an *oracle* which has insight into the runtime distributions beforehand (based on training). However, it was unable to react to any incorrect predictions made by the model.

Sequential Parameter Optimization (**SPO**) differentiates itself from other SMBO approaches by fitting a noise-free Gaussian Process model to the sampled data (Bartz-Beielstein *et al.*, 2005). The best parameter setting, based on the model, is determined by the Expected Improvement criterion, which tries to balance between exploring new options and exploiting the current best. A comprehensive treatment of SPO can be found in Hutter *et al.* (2010a).

Hutter *et al.* (2009b) investigated four design choices in SPO: how to construct the initial model, use of the Expected Improvement criterion, fitting a model to raw data or log-transformed data, and use of an intensification mechanism. They concluded that changing the intensification method and fitting a model to log-transformed data had the largest impact on algorithm performance, in terms of solution quality and running time. They also proposed **SPO+** which replaces the current best solution if its competitor shows improved performance *after* it has been sampled the same number of times. This better distinguishes between competing parameter settings. Still, SPO+ also has a rule to discard a competitor if it fails to show improvement for a pre-determined number of iterations, which may lead to loosing good parameter settings.

Time Bounded SPO (**TB-SPO**) introduces three modifications to the basic method: first, the intensification phase is now limited by a time constraint (Hutter *et al.*, 2010c). Second, to save time, the Gaussian Process model is replaced with an approximation model known as the Projected Process model. Third, it interleaves choosing parameter settings based on the Expected Improvement criterion and random selection. TB-SPO showed significant speedups when compared to the standard SPO tuning a local search algorithm for the SAT problem.

Another extension of SPO, relevant to this work, can be found in Bartz-Beielstein *et al.* (2011), where the authors applied the combination of OCBA with SPO, see Lasarczyk (2007), to intelligently determine the best distribution of the computing budget among the new design points. SPO was used as an optimization algorithm applied to five noisy optimization functions. The authors compared six SPO variants and three optimization algorithms and found that OCBA significantly improves on SPO on three out of the five functions, and did not degrade the performance on the remaining two. It was noted, however, that given the small scale of the experiments, the results are only indicative, and further research is needed.

Hutter *et al.* (2011) introduced Sequential Model-based Algorithm Configuration (**SMAC**), which is based on random forests models. It handles categorical parameter settings and works over a set of problem instances, not just one as in SPO. SMAC performed better when compared to ParamILS and a Gender Based Genetic Algorithm, see Ansotegui *et al.* (2009), when configuring a number of SAT solvers.

Relevance Estimation and Value Calibration (**REVAC**) is an EDA variant which estimates a parameter’s relevance using normalized Shannon entropy (Nannen and Eiben, 2006). Instead of estimating the performance of an algorithm under different parameter values, REVAC calculates the expected performance of a parameter taken from a probability distribution constructed over the parameter space. It then evolves such distributions into ones with decreasing Shannon entropies, such that relevant parameter settings are selected more often and hence evolve to better values. Further details can be found in Smit and Eiben (2010a).

Initial implementations of REVAC were focused on tuning GAs solving a number of continuous optimization functions (Nannen and Eiben, 2007). It was concluded that tuning the mutation rate is more important than the crossover rate, with recommended values of 0.01-0.1 and 0.6-1.0, respectively. Smit and Eiben (2010c) trained a generalist and a specialist using REVAC. Again the algorithm was a GA solving a number of continuous optimization functions. Comparing solution quality of these algorithms to those obtained with default parameter settings, the generalist outperformed the specialist on four out of six functions, but it was inferior to the default on the Sphere function.

REVAC also showed significant gains when tuning the winner of the 2005 CEC competition, even though its default parameters were carefully set by the creators (Smit and Eiben, 2010b). The competing algorithms were presented with a number of continuous optimization functions, and were evaluated on three different performance measures. One outcome of this study is that REVAC found different parameter settings for each performance measure. While this is expected, it raises questions about using one parameter setting for different performance measures, as was done with the winning algorithm.

2.5 Online or offline?

Only few recent studies compared online and offline methods, in hopes to better understand when to apply each. Still, this research area remains scarce and requires more attention. Nareyek (2004), see Section 2.3.2, compared adaptive reinforcement learning strategies, used in finding the best set of weights for a number of heuristics, to fixed strategies obtained through training. They concluded that the fixed ones outperform their adaptive counterparts given sufficient time, and if they are tested on instances similar to the training set. They also remarked that for complex problems, the fixed strategy performs just as well as the adaptive one, even if sufficient time is available.

Pedersen and Chipperfield (2008c), and Pedersen and Chipperfield (2008b) used several DE variants to solve a number of optimization functions, and to find the best weights for a number of ANNs. To allow for a fair comparison, all DEs were tuned by meta-optimization, even though some had adaptive mechanisms to adjust their parameters online. The conclusion was that the adaptive DEs did not have any advantage, in terms of solution quality and convergence rate, over those using the fixed parameter settings obtained from training. Also, the adaptive methods did not change the parameter settings they started with very often.

Battiti and Campigotto (2011), see Section 2.3.2, compared two algorithms using reinforcement learning strategies to adapt their parameters, to the same algorithms using fixed tuned parameter obtained by running ParamILS. Results showed that the fixed parameters performed significantly better on few instances, and were better on all other instances but the improvement was not significant. Again, this indicates that online adaptation has no advantage over carefully tuned fixed parameter settings.

A recent study specifically aiming at comparing online and offline methods can be found in Pellegrini *et al.* (2010). Six parameters for the MMAS algorithm, applied to the TSP, were tuned using five self-adaptive online tuners and one offline tuner, *F*-Race. Four different scenarios were compared: use default parameter settings, tune the parameters offline, control the parameters online, and finally starting with offline-tuned parameters, using *F*-Race, try to find better ones online. Results showed that the improvement in MMAS's performance is higher with offline tuning. Also, online control can find better parameters than what is used in the literature if the computational budget is low. The effect of increasing the number of parameters tuned online was also investigated, starting from one parameter and going up to six parameters. It was shown that the performance of the online methods degrades as the number of parameters increases.

The same authors carried out another analogous study using Memetic Algorithms applied to the QAP (Francesca *et al.*, 2011). Their conclusions were generally the same. The interested reader is also referred to Stützle *et al.* (2011) for an extensive review on the various parameter tuning methods used for ACO algorithms.

Generalizing the results of these studies is not possible, as they apply only to the algorithms and experimental setup chosen. Yet, they are consistent in their findings and do seem to support the counter-intuitive conclusion that online control has no advantage over offline tuning.

2.6 Prospect research

Significant advances have been made to the algorithm configuration problem. However, there remain many areas which received little, or no, attention so far. This section will highlight such areas.

Starting with online control, determining the number of applications and replications required to properly evaluate an operator remains unclear. Evaluating too often (e.g. after a single application and a single replication) may lead to poor evaluations, especially if the algorithm is stochastic. Evaluating after many applications and replications can be expensive, and may not allow the algorithm to react in a timely fashion. Most online methods evaluate an operator after a single application, and hardly ever replicate. Only few methods evaluate an operator over a sliding window, but with no proper justification to the choice of the window size. An example is the configurator Compass discussed earlier.

From the literature reviewed in this chapter, it can be seen that all algorithm configuration methods, online or offline, introduce their own hyper-parameters. In some situations there are just as many hyper-parameters as there are parameters to tune, see Table 2.1 for an example. Some authors claim that the hyper-parameters are robust and are easier to set. Others use a *third* level algorithm to configure the hyper-parameters of the configurator itself. It remains unclear whether tuning/controlling the hyper-parameters will improve the configurator’s ability to find better parameters settings for the target algorithm, ones that could not have been discovered if the hyper-parameters were not tuned or controlled.

Table 2.1. A sample comparison of the number of hyper-parameters required by some online controllers and the corresponding number of parameters to control.

Configurator	Number of Hyper-parameters	Number of parameters
Whitacre *et al.* (2006)	8	10
PM-AdapSS-DE (Gong *et al.*, 2010)	4	6
Compass (Maturana and Saubion, 2008)	3	5
AP (Thierens, 2005)	5	6
SLPSO (Wang *et al.*, 2011)	4	6
DMAB (Dacosta *et al.*, 2008)	4	5

This issue was tackled in Herdy (1992), where the author used three ES algorithms: the first optimizes the sphere function, the second tunes the mutation step size of the first ES, and the third tunes the number of generations of the second ES. All three levels work offline so the whole algorithm can be considered a meta-meta-optimizer. It was empirically shown that the long term performance is improved when tuning the second level compared to setting its parameters by trial and error. One drawback of Herdy's approach is the huge computational overhead added by the third level. This can be improved, for example, if the third level is replaced with an online controller. Moreover, the number of levels that should be added to tune/control the parameters one level below is potentially infinite, and one can argue that the algorithm configuration problem is just shifting from one level to another (De Jong, 2007).

Whether tuning the hyper-parameters is truly needed, and is in fact feasible, algorithm configurators have improved the performance of many algorithms compared to the default settings. With all the available configurators nowadays, it is unjustifiable to hand-tune newly created algorithms, or accept the default when solving problems from different domains.

2.7 Conclusion

This chapter reviewed research relevant to the algorithm configuration problem, where it was broadly classified as online control or offline tuning. Online control was further sub-divided into single step look-ahead methods, where operator selection is done in every iteration, and multi-step look-ahead methods, where an operator is allowed several applications before it is evaluated and a new selection is made. Within offline tuning, model-based methods were distinguished from model-free methods on the basis of whether, or not, a model is built to map the relationship between parameter settings and algorithm performance. Model-free methods were further classified into meta-optimizers or computational budget allocators.

Configuring algorithms online or offline depends on the application. Online control is more suitable when solving a single instance, a dynamic instance, or instances that are quite different from one another. Offline tuning, however, is more suitable when solving many, similar, instances. A few comparative studies have been carried out in this direction, and contrary to the general belief, online control has not shown advantage over offline tuning. Still, these results apply only to the algorithms and problem instances tested, so further research is needed.

Within the above mentioned categorization, the first contribution, the Flexible Budget method, falls under meta-optimization, while the second and third contributions, Racing with a self-adaptive significance level and one-way Racing with an intelligent budget allocation, fall under computational budget allocation. All contributions are offline tuners.

Finally, no single algorithm has outperformed all other algorithms over all problems and performance measures, even if its parameters are properly set. An attempt to find a configurator which tunes, or controls, the parameters of a target algorithm, such that it performs well over a wide range of different problem domains, was carried out in the CHeSC in 2011. The interesting result was that the winning algorithm of Mustafa Misir *et al.* (2012) was never the best on any individual problem domain, but was best, on average, across all problem domains.

CHAPTER 3 Theory and Methodology

3.1 Introduction

This chapter details the workings of each of the aforementioned contributions, together with that of the methods they compare to, or improve upon. The chapter is organized as follows: Section 3.2 explains why a typical meta-optimizer requires several runs to find the best parameter settings for different computational budgets, and how the Flexible Budget method can determine the best parameter settings for any computational budget, in a single run. Computational budget allocators, OCBA and CBA, are discussed in Section 3.3, and Racing algorithms are in Section 3.4. Section 3.5 shows how a Racing algorithm can adapt its parameter, the significance level, and consume any given budget. Section 3.7 introduces one-way Racing and shows how it has an advantage over two-way Racing when used with the reset idea. Finally, Section 3.7 concludes this chapter.

3.2 Meta-optimization

Meta-optimization is an offline tuning method consisting of a meta-level algorithm searching the parameter settings' space of another lower-level (LL-) algorithm, which, in turn, searches the solution space of an optimization problem (e.g. TSP). To simplify the description, both levels are assumed to be EAs, following a basic EA cycle: *initial population* → *parent selection* → *variation operators* → *survival selection* → *new generation*. Solutions of the meta-EA (its individuals) are parameter settings of the LL-EA that can be numerical and/or categorical.

3.2.1 Meta-optimization with a fixed budget

The meta-EA begins with a population of individuals, usually selected at random from the configuration space. It then evaluates each by running the LL-EA, with that fixed parameter setting, on the optimization problem, until a stopping condition is met. This reflects that parameter's performance at only a single point (the end of the run). The utilities of individuals are typically the solution quality of the LL-EA, the running time required to reach a target solution quality, or the number of times a target is reached within a specific time limit. Afterwards, the meta-EA chooses parents from its current population, based on the utilities, and creates a new offspring population by applying variation operators (e.g. crossover and mutation). Out of the two populations, a sub-set is selected to "survive" to the next generation. An example is shown in Figure 3.1.

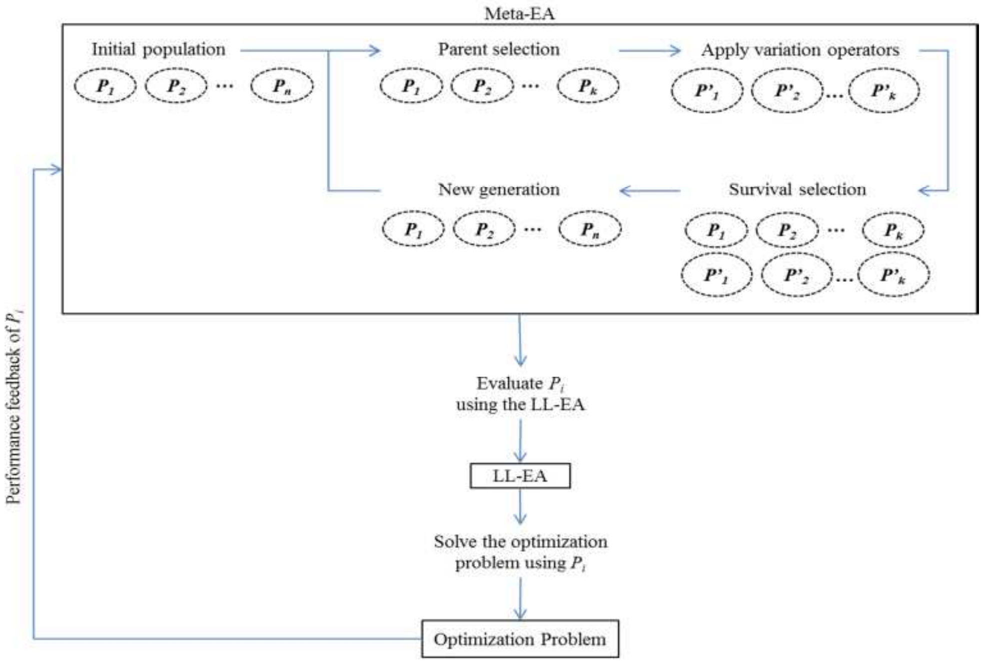


Figure 3.1. A sample meta-EA.

Since the LL-algorithm is likely to be stochastic, obtaining accurate utility values requires multiple runs. The same holds for the meta-level if it is stochastic, it should ideally converge to the same parameter settings, or close enough, over different replications. All of this leads to a very time consuming offline tuner. Matters get worse if the computational budget for the lower level changes often, as optimizing different computational budgets requires a separate run of the meta-optimizer, one run for each potential new budget. For example, different instances of a VRP are solved repeatedly under varying time constraints. Figure 3.2 shows an example of convergence curves for an algorithm, minimizing a multimodal function, under two different parameter settings. Clearly, a greedy parameter setting is suitable for lower budgets, whereas an explorative one is better for higher ones.

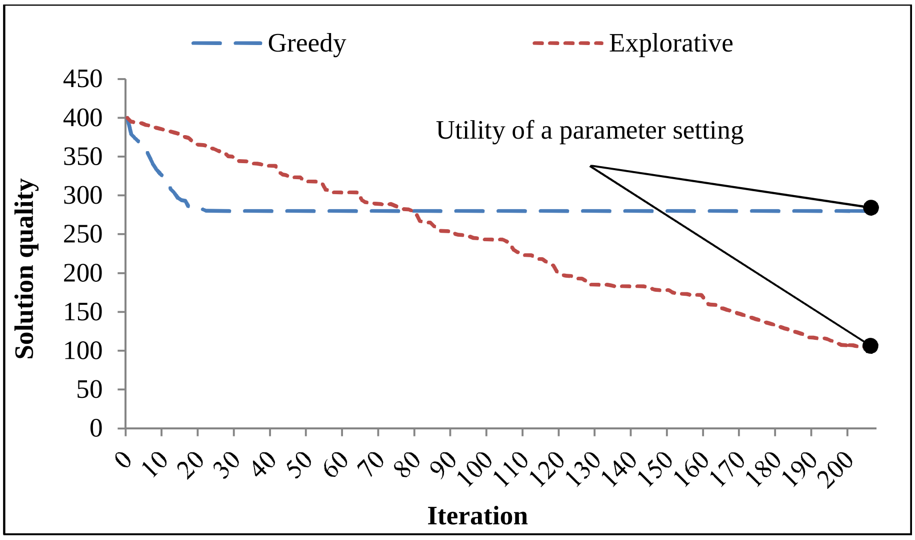


Figure 3.2. Convergence curves of an algorithm, applied to a minimization problem, under two different parameter settings.

3.2.2 Meta-optimization with a flexible budget (1st contribution)

The main idea is to maintain a diverse set of parameter settings suitable for different computational budgets. To achieve this, utilities no longer reflect the algorithm’s performance at a single point (the end of the run), but they are rather composed of each point on the entire convergence curve, building a lexicographic ordering between the parameter settings. This, in turn, will affect the selection at the meta-level, favoring individuals fit for different computational budgets, rather than for a pre-defined fixed budget. The proposed contribution will be referred to as the Flexible Budget method, compared to the traditional Fixed Budget method.

Again, assuming both levels are EAs, the Flexible Budget method works as follows:

1. Randomly initiate a population of parameter settings from their domains.
2. Evaluate each individual at the meta-level by running the lower level algorithm, with that parameter setting, for a specific computational budget n_{max} (maximal computational budget of interest). Replicate k times using common random numbers.
3. Using the lower-level objective function value as a utility, record for each parameter setting the best-so-far average utility at each function evaluation (avBSF).
4. Identify the parameter setting(s) with the best avBSF at each function evaluation of the lower-level. Rank these parameter settings as rank 1.
5. Remove rank-1 parameter settings from the population and repeat as before, classifying the new parameter settings as rank 2.
6. Continue until all parameter settings have been ranked.

   Example: Figure 3.3 shows the convergence curves of an algorithm under three different parameter settings. During the first 160 iterations, the solid line has the best objective value, so it's ranked 1. The dashed line has the best objective function value from iteration 160 onwards, so it is ranked 1 too. Removing rank-1 parameters, one line remains, rank it as 2.

   This ranking scheme is inspired by the non-dominated sorting of Deb *et al.* (2002).
7. To favor between individuals in the same rank, three secondary criteria are proposed:
   - Length (L): select individuals which are in that rank for a longer period.

     Example: In Figure 3.3 if parent selection at the meta-level used a tournament selection of size 2, and the dashed and solid lines were chosen, pick the solid line as it is in the same rank but for a longer period.
   - Area Under the Curve (AUC): select individuals with a smaller (respectively larger) AUC for a minimization (respectively maximization) problem.

     Example: In Figure 3.4, both curves are in the same rank, if one is to be selected at the meta-level, choose the solid line as it has a smaller AUC.

- Area Lost (AL): this measure represents the quantity lost if a particular individual is not selected at the meta-level; hence, prefer individuals with higher AL values.

  Example: In Figure 3.5, both curves are in the same rank, if one is to be selected at the meta-level, disregard the solid line, as one loses less area if it is not chosen.

  The AL measure is relative to the individuals being compared, whereas the AUC measure for a curve is independent of the other curves.

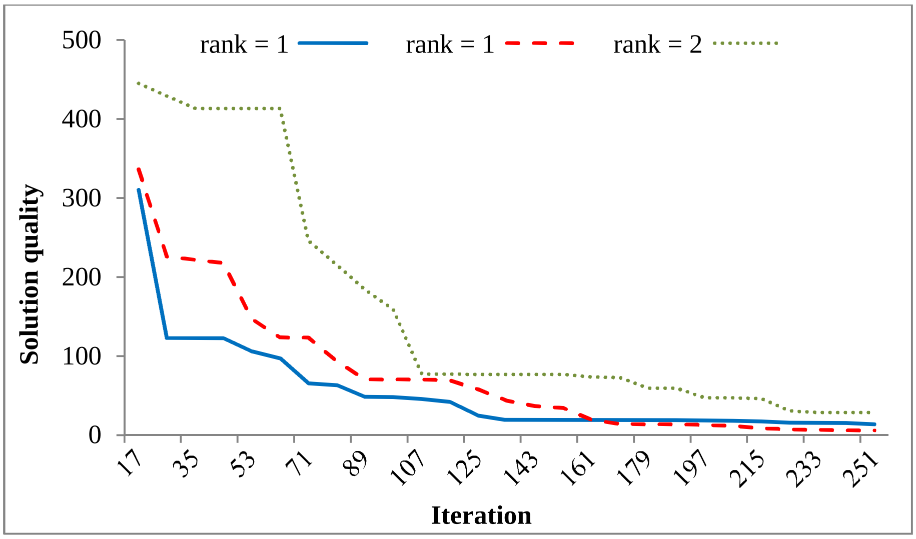


Figure 3.3. Ranking parameter settings using the Flexible Budget method.
Assume a minimization problem.

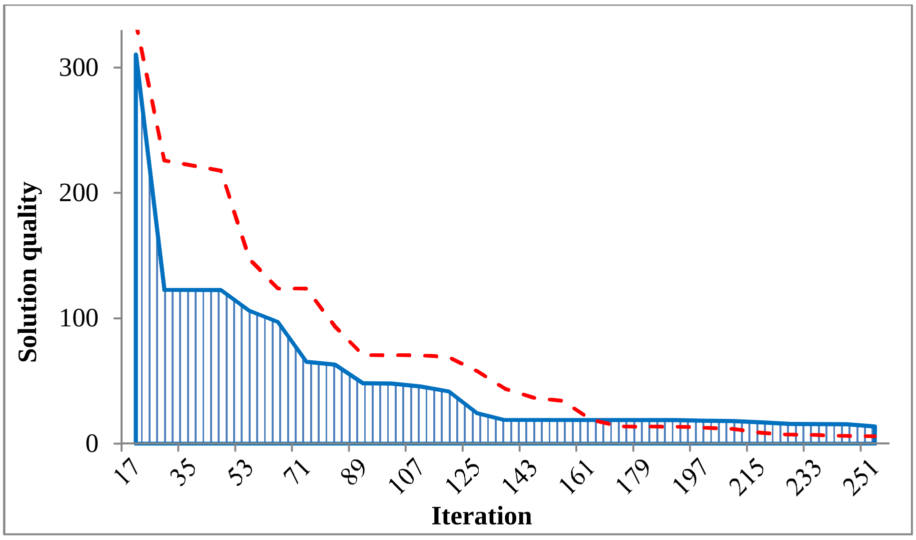


Figure 3.4. Area Under the Curve as a secondary criterion for the Flexible-Budget method.
Assume a minimization problem.

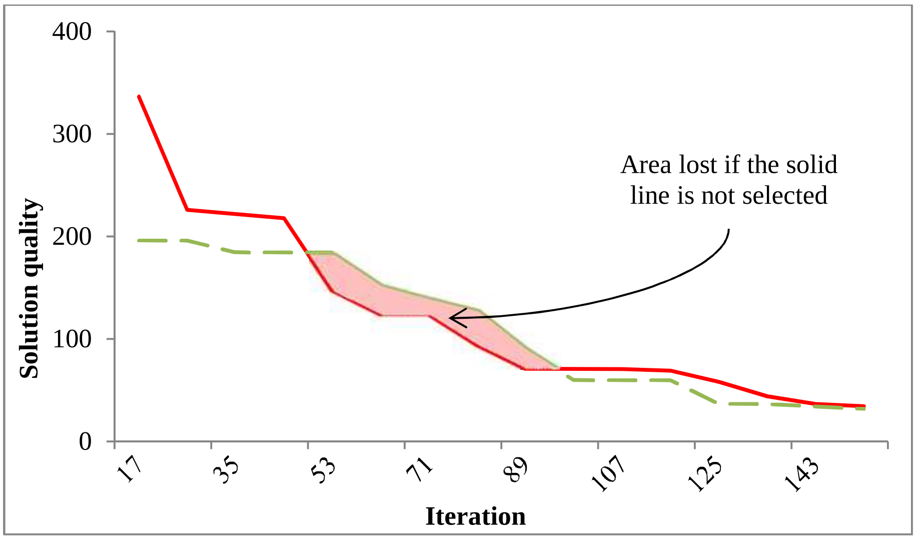


Figure 3.5. Area Lost as a secondary criterion for the Flexible-Budget method. Assume a minimization problem.

8. When the meta-level terminates, report the best parameter setting found at each lower-level function evaluation, as the parameter setting to use when solving a similar problem with only that many function evaluations.

   Example: In Figure 3.6, if a new instance is to be solved with only 100 iterations, the parameter setting which generated the solid line will be used. Likewise, for 50 iterations, select the dashed line.

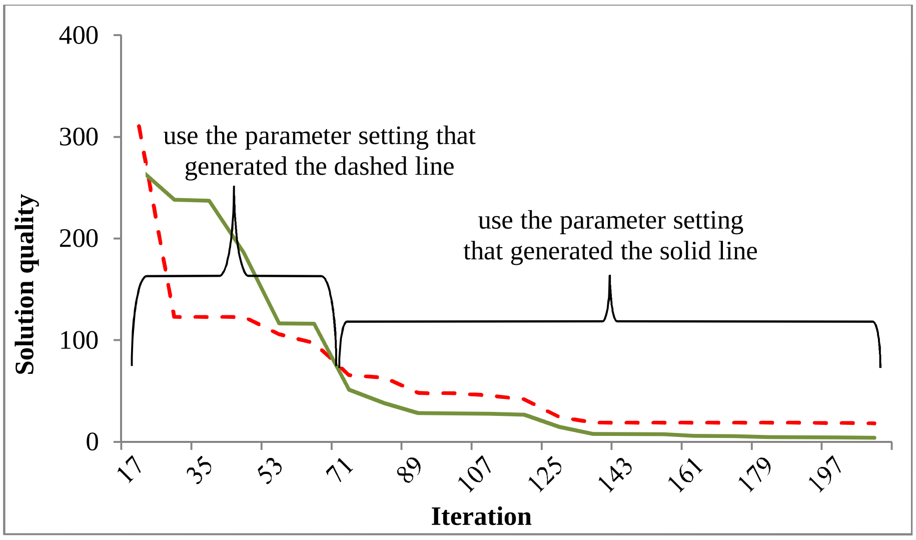


Figure 3.6. Parameter settings suitable for different computational budgets. Assume a minimization problem.

```
Algorithm 3.1: The Flexible Budget method
Input for the meta-EA: metaGen, metaRep, metaPopSize, LL-parameters
Input for the LL-EA: $n_{max}$, k

Initialize meta-EA
randomly choose LL-parameters
from their respective domains
                                                          //replication, generation,
$r \leftarrow 0$; $g \leftarrow 0$; $i \leftarrow 0$      //and individual counters
while r < metaRep                                         //Meta-EA run
  while g < metaGen
    while i < metaPopSize
      run LL-EA using individual i                        //LL-EA run
      store the entire convergence curve
      averaged over k replications
      assign ranks                                        //Rank-based evaluation
      calculate L or AUC                                  //absolute measures
      i++
    end while
    parent selection                                      //using ranks and L, AUC,
    if AL is used                                         //or AL. AL is a relative
      calculate AL for the competing individuals          //measurement
    apply variation operators
    survival selection                                    //using ranks and L, AUC,
    if AL is used                                         //or AL. AL is a relative
      calculate AL for the competing individuals          //measurement
    g++                                                   //next meta-generation
  end while
  r++                                                     //next meta-replication
end while
Output: best parameter settings for any computational budget less than $n_{max}$
```

Figure 3.7. A pseudo-code of Flexible Budge method.

3.3 Budget allocators

Inputs to a computational budget allocator are: a number of algorithms k, or the same algorithm with different parameter settings, a problem instance generator I, and a computational budget N. The objective is to allocate N among the $\langle algorithm, instance\rangle$ pairs, such that the best algorithm is correctly selected when N is consumed. This is the training phase. The working assumption is that the training set is large enough and representative, and that the testing set does not differ much such that the algorithm which performed best during training will also perform best on the testing set.

Running algorithm i on an instance j yields utility U_{ij}, and aggregating all the utilities gives one measure that is used to favor between the competing algorithms. All algorithms are first run on the same subset of problem instances n_0 to estimate their performance, and then the allocator is run to determine how to distribute the remaining $N - k \cdot n_0$ samples. Generally the remaining budget is not allocated all at once, but rather sequentially; every time a portion Δ of the remaining budget is distributed, the aggregated utilities are updated, and the allocator is run again. See Figure 3.8.

The Ranking and Selection literature has many budget allocation algorithms which fit the same setting just described. The parameter settings are replaced with stochastic simulation systems, and the problem instances are replaced with samples drawn from a distribution representing the simulation output. The focus here is on two such algorithms, OCBA and its correlated version CBA.

Problem instance or sample number	Parameter settings or systems				
	1	*2*	*3*	*4*	*5*
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	1	1	1
4	1	1	1	1	1
5	1	1	1	1	1
6	2	3	2	3	5
7	2	3		3	5
8	2	3		4	5
9	2	4		4	5
10		4		4	5
…		…		…	…
m					

Figure 3.8. An example of allocating a budget between five parameter settings/systems with $n_0 = 5$. The numbers indicate the iteration in which a sample is taken. For instance, systems 1 and 3 were sampled in the second iteration with system 1 being tested on instances 6-9, while system 3 was run on instance 6.

3.3.1 OCBA maximizing the probability of correct selection

The main idea of OCBA is to allocate a large share of the total budget to those systems which are critical in detecting the true best. Doing so will decrease the variance of their estimated utility, and increase the Probability of Correct Selection (PCS). Moreover, it will lower the computational effort spent on non-critical systems, which do not contribute much in distinguishing between the competing systems.

The problem of allocating a budget N among k systems, such that the PCS is maximized, can be stated as

$$\max_{n_1,n_2,\ldots,n_k} \quad PCS$$
$$s.t. \sum_{i=1}^{k} n_i = N, \text{and } n_i \in \mathbb{Z}^+, \tag{3.1}$$

where n_i is the number of times system *i* is to be sampled. Chen and Lee (2010, p.41) showed that for the simple case of $k = 2$, a theoretical optimal solution can be found. Let $J_1$and J_2, $\sigma_1$and σ_2 be the means and standard deviations of the competing systems. An optimal allocation is given by

$$\frac{n_1}{n_2} = \frac{\sigma_1}{\sigma_2}, \tag{3.2}$$

which does not depend on the means, only the standard deviations. For the general case ($k > 2$), the formulation of the PCS becomes intractable and is replaced with an approximate probability of correct selection, which is asymptotically maximized when

$$\frac{n_i}{n_j} = \left(\frac{\sigma_i/\delta_{b,i}}{\sigma_j/\delta_{b,j}}\right)^2 \quad i,j \,\epsilon \{1, 2, \ldots, k\} \text{and } i \neq j \neq best \tag{3.3}$$

$$n_b = \sigma_b \sqrt{\sum_{i=1, i\neq b}^{k} \frac{n_i^2}{\sigma_i^2}}, \tag{3.4}$$

where $\delta_{b,l} = \overline{J_b} - \overline{J_l}$ and $\overline{J_b} < \min_{l,l\neq b} \overline{J_l}$, assuming that a system with the lowest mean is the best. For the two system case, 3.3 and 3.4 reduce to 3.2. It should be noted that the constraint requiring n_i to be integers is relaxed, which means that the solution should be rounded such that $\sum_{i=1}^{k} n_i = N$.

Figure 3.9 outlines a pseudo-code of an OCBA algorithm, which allocates Δ additional samples among the *k* systems in every iteration *t*. In particular:

1. Sample each system n_0 times.
2. While the computational budget has not been consumed:
   - Calculate/update the sample averages and standard deviations.
   - Find the system with the best sample average *b*.

- Solve 3.3 and 3.4 to determine how to allocate Δ for the next iteration. That is, the expected number of samples to take from each system at iteration t+1 $\hat{n}_l^{t+1}$. Rounding maybe necessary for this step.
- Sample each system l $max\left(\hat{n}_l^{t+1} - n_l^t, 0\right)$.
- Move on to the next iteration.

```
Algorithm 3.2: OCBA
Input: k, N, Δ, n0                                       //n0 ≥ 5 and N-k.n0 is a
                                                         //multiple of Δ
Initialize
$t \leftarrow 0;\ c \leftarrow 0$                        //iteration and consumed
                                                         //budget counters
sample each system n0 times $n_1^t = n_2^t = \cdots n_k^t = n_0$   //initial performance
                                                         //estimation
$t \leftarrow n_0;\ c \leftarrow n_0 \cdot k$
Main allocation loop
while $c \le N$
  $\bar{J}_l = \frac{\sum_{i=1}^{n_l^t} \omega_{i,l}}{n_l^t}\ \forall l = 1, \cdots, k$   //update the sample
                                                         //average and standard
  $s_l = \sqrt{\frac{\sum_{i=1}^{n_l^t} \left(\omega_{i,l} - \bar{J}_l\right)^2}{n_l^t - 1}}\ \ \forall l = 1, \cdots, k$   //deviation for each.
                                                         //ω_{i,l} is the i^th sample
                                                         //taken from system l
  find $b = arg\,\min_l \bar{J}_l$                       //find the system with
                                                         //the best sample average
  solve 3.3 and 3.4 such that $\sum_{l=1}^{k} \hat{n}_l^{t+1} = \sum_{l=1}^{k} n_l^t + \Delta$   //allocate Δ among all
                                                         //systems. rounding maybe
                                                         //necessary
  sample system l $max(\hat{n}_l^{t+1} - n_l^t, 0)\ \forall l = 1, \cdots, k$
  $c \leftarrow c + \Delta$                              //update consumed budget
  $t \leftarrow t + 1$                                   //next iteration
end while
Output: optimal allocation of $N - k \cdot n_0$
```

Figure 3.9. A pseudo-code of the OCBA algorithm.

3.3.2 CBA

Utility values for parameter settings are usually correlated, for example, if a particular problem instance is hard, parameter settings are expected to perform poorly, or worse than their average performance. OCBA was extended in Fu *et al.* (2004) to account for correlated data. CBA works similar to OCBA, and can be implemented sequentially. Moreover, it produces the same allocation as OCBA for the k = 2 case with zero correlation. For the general case (k > 2) with zero correlation, CBA and OCBA reach quite similar performance, but their solutions are not exactly identical (Fu *et al.*, 2007).

The implementation specifics of CBA are not as direct as those of OCBA, and some of its steps can be done in several ways. It is beyond the scope of this work to go into all of its details, and the interested reader is directed to Fu *et al.* (2007). However, a simplified version is described here, that of Fu *et al.* (2004) to enable the reader to grasp some of the concepts of the algorithm. First, some definitions are required. As before, n_l^t is the number of samples to take from system *l* at iteration *t*, and $\overline{J_l}, s_l$ are the sample average and sample standard deviation of system *l*. Denote n_*^t and n_{**}^t as the number of samples to take from the systems with the best, and second best, sample averages known so far. Furthermore, let C_{ij} be the covariance between systems *i* and *j*. Finally, define the following quantities

$$\widehat{\beta_l} = \left(\overline{J_*} - \overline{J_{**}} \Big/ \overline{J_*} - \overline{J_l} \right)^2 \; \forall l = 1, \dots, k \text{ and } l \neq * \Rightarrow \hat{\beta}_{**} = 1 \qquad 3.5$$

$$\begin{aligned} \widetilde{C_{*l}} &= s_*^2 - 2C_{*l} \\ \widetilde{C_{ll}} &= s_l^2 \end{aligned} \qquad if\; n_*^t \geq n_l \qquad 3.6$$

$$\begin{aligned} \widetilde{C_{*l}} &= s_*^2 \\ \widetilde{C_{ll}} &= s_l^2 - 2C_{*l} \end{aligned} \qquad if\; n_*^t < n_l. \qquad 3.7$$

To begin with, find a value for n_{**}^t that satisfies

$$1 = \sum_{l=1}^{k} \frac{\hat{\beta}_l^2 \tilde{C}_{ll} \tilde{C}_{*l}}{\left(\tilde{C}_{*,**} + \tilde{C}_{**,**}/n_{**}^t - \hat{\beta}_l \tilde{C}_{*l}\right)^2}, l \neq * \qquad 3.8$$

using any numerical method. Note that the summation is over all systems except the current best. Then, all other n_l^t, except n_*^t, are calculated as

$$n_l^t = \frac{\widehat{\beta_l} \widetilde{C_{ll}}}{\tilde{C}_{*,**} + \tilde{C}_{**,**}/n_{**}^t - \hat{\beta}_l \tilde{C}_{*l}}, l \neq * \; l \neq **. \qquad 3.9$$

Finally, n_*^t is calculated via

$$n_*^t = \sqrt{\sum_{l=1}^{k} \left(\left[\widetilde{C_{*l}} \Big/ \widetilde{C_{ll}} \right] {n_l^t}^2 \right)}, l \neq *. \qquad 3.10$$

Implementing a sequential version of CBA follows the same framework as that of OCBA. The major difference is how n_l^t are calculated, and the use/update of a covariance matrix, which is not present in OCBA. A pseudo-code is in Figure 3.10.

```
Algorithm 3.3: CBA
Input: $k$, $N$, $\Delta$, $n_0$                              //$n_0 \geq 5$ and $N$-$k.n_0$ is a
                                                               //multiple of $\Delta$
Initialize
$t \leftarrow 0$; $c \leftarrow 0$                             //iteration and consumed
                                                               //budget counters
sample each system $n_0$ times $n_1^t = n_2^t = \cdots n_k^t = n_0$    //initial performance
                                                               //estimation
$t \leftarrow n_0$; $c \leftarrow n_0 \cdot k$
Main allocation loop
while $c \leq N$                                               //total number of samples
                                                               //is less than N
    $\bar{J}_l = \frac{\sum_{i=1}^{n_l^t} \omega_{i,l}}{n_l^t} \quad \forall l = 1, \cdots, k$    //update the sample
                                                               //average each system
                                                               //$\omega_{i,l}$ is the $i^{th}$ sample
                                                               //taken from system l
    Update the covariance matrix
    find $b = arg \min_l \bar{J}_l$                            //find the system with
                                                               //the best sample average
    solve 3.8-3.10 such that $\sum_{l=1}^{k} \hat{n}_l^{t+1} = \sum_{l=1}^{k} n_l^t + \Delta$    //allocate Δ samples
                                                               //among all systems
                                                               //rounding
                                                               //maybe necessary
    sample system l $max(\hat{n}_l^{t+1} - n_l^t, 0) \ \forall l = 1, \cdots, k$
    $c \leftarrow c + \Delta$                                  //update consumed budget
    $t \leftarrow t + 1$                                       //next iteration
end while
Output: optimal allocation of $N - k \cdot n_0$
```

Figure 3.10. A pseudo-code of the CBA algorithm.

3.4 Racing algorithms

Racing algorithms follow the same framework as that of OCBA/CBA. The major difference, however, is that in each iteration Racing subjects all “surviving” parameter settings to a series of statistical tests, which could identify a subset that is significantly worse than the current best. In that case, the subset is discarded from further consideration, and each surviving parameter setting is tested on the next instance. The statistical tests are carried out over two stages. The first is an ANOVA test that determines if at least one parameter setting performs differently from the rest. The second stage is a number of pair-wise comparisons that pinpoint inferior parameter settings. Figure 3.11 offers a classification of the Racing algorithms presented here.

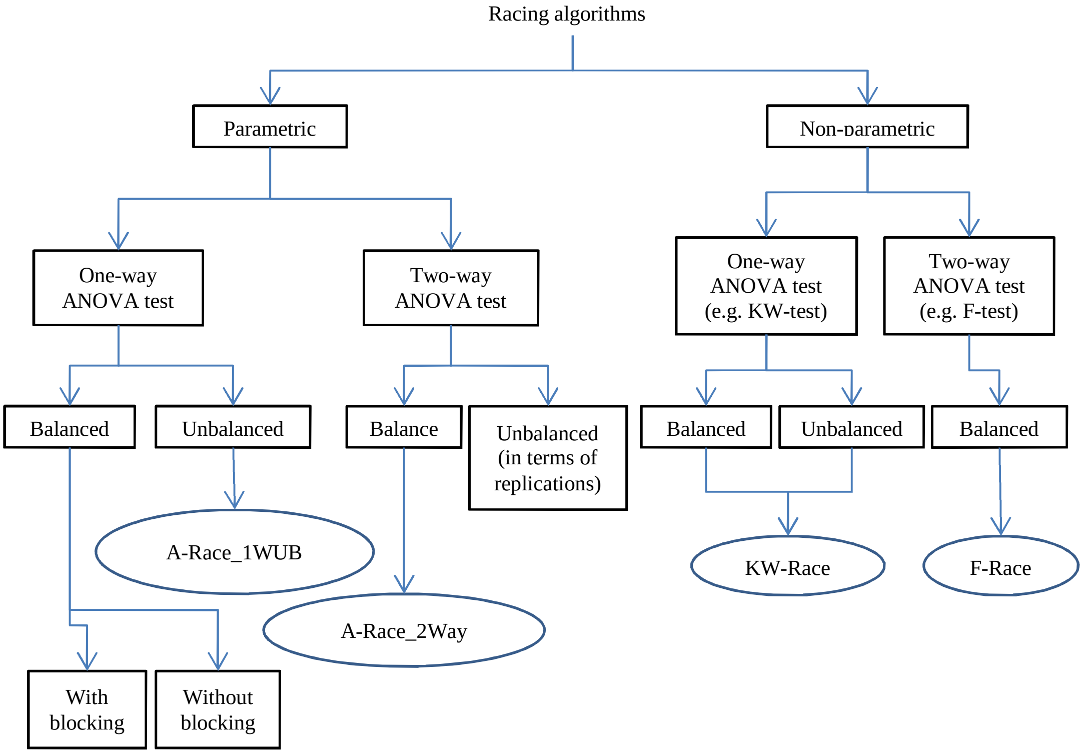


Figure 3.11. Classification of the Racing algorithms used in this thesis.

3.4.1 Parametric Racing

Parametric Racing, *A*-Race in particular, refers to a Racing algorithm which uses a normal-model based ANOVA for the first stage filtering, followed by tests such as: a paired *t*-test, Tukey's honest significance test, Scheffe's test, or Fisher's Least Significant Difference (LSD) to name a few. Montgomery (2001, pp.96-107) presents these and other methods, and provides a comparative discussion. A normal-model based ANOVA, or simply ANOVA, assumes that the error terms (see 3.11-3.13) are normally and independently distributed with mean of zero and unknown but equal variance σ^2. More about the assumptions can be found in Glass *et al.* (1972), and Wilcox (1987). Some of the assumptions may not be satisfied when tuning parameters; still, there are methods to reduce the effect of violating them.

The next section gives a brief background on some ANOVA models that are relevant to *A*-Race, followed by a review of various studies showing the effect of not complying with the assumptions. Finally, the applicability of ANOVA models within a Racing algorithm is discussed.

3.4.1.1 A background on normal-model based ANOVA

The following ANOVA models are discussed here: two-way ANOVA, one-way ANOVA, and one-way ANOVA with blocking. All methods are considered under fixed and random effects. Most of what follows is based on the works of Montgomery and Runger (2011).

Using some terminology from Design of Experiments, an algorithm's utility, its *response*, is affected by two *factors*: the parameter setting *A*, and the problem instance *B*. The specific values for these factors are called *factor levels*, or *treatments* (i.e. a parameter setting), *a* levels for factor *A* and *b* levels for factor *B*. The response for level *i* of factor *A*, level *j* of factor *B*, over replication *k*, is a random variable following the linear model

$$Y_{ijk} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij} + \varepsilon_{ijk} \begin{cases} i = 1,2, \dots , a \\ j = 1,2, \dots , b \\ k = 1,2, \dots , n \end{cases} , \tag{3.11}$$

where μ is the overall mean response, τ_i is the effect of treatment *i* of factor *A*, β_j is the effect of treatment *j* of factor *B*, $(\tau\beta)_{ij}$ is the interaction effect between factors *A* and *B* at levels *i* and *j*, and finally ε_{ijk} is the random error term $\sim\mathcal{N}(0, \sigma^2)$. This is known as a two-way ANOVA model. Table 3.1 shows one possible data arrangement for a two-way ANOVA.

Table 3.1. Data arrangement for a two-way ANOVA model.

Factor *B*						
	Observations	1	2	…	*b*	Sum
Factor *A*	1	$y_{111}, y_{112},\dots, y_{11n}$	$y_{121}, y_{122},\dots, y_{12n}$	…	$y_{1b1}, y_{1b2},\dots, y_{1bn}$	$y_{1..}$
	2	$y_{211}, y_{212},\dots, y_{21n}$	$y_{221}, y_{222},\dots, y_{22n}$	…	$y_{2b1}, y_{2b2},\dots, y_{2bn}$	$y_{2..}$
	⋮	⋮	⋮	⋮	⋮	⋮
	a	$y_{a11}, y_{a12},\dots, y_{a1n}$	$y_{a21}, y_{a22},\dots, y_{a2n}$	…	$y_{ab1}, y_{ab2},\dots, y_{abn}$	$y_{a..}$
Sum		$y_{.1.}$	$y_{.2.}$	…	$y_{.b.}$	$y_{...}$

A one-way ANOVA, on the other hand, accounts for only one factor, the parameter setting, and follows the statistical linear model

$$Y_{ij} = \mu + \tau_i + \varepsilon_{ij} \begin{cases} i = 1,2, \dots, a \\ j = 1,2, \dots, n \end{cases}, \qquad 3.12$$

for which data can be arranged as in Table 3.2. This model is applicable to offline tuning when it is believed that the algorithm's performance depends solely on its parameters, and that the problem instances are quite similar such that their effect can be neglected.

Table 3.2. Data arrangement for a one-way ANOVA model.

		Observations				Sum
Factor *A*	1	y_{11}	y_{12}	…	y_{1n}	$y_{1.}$
	2	y_{21}	y_{22}	…	y_{2n}	$y_{2.}$
	⋮	⋮	⋮	⋮	⋮	⋮
	a	y_{a1}	y_{a2}	…	y_{an}	$y_{a.}$
Overall sum						$y_{..}$

Sometimes there are two factors affecting the response, but only one is of interest, say factor *A*. For instance, the effect of the problem instance is considered negligible and one is interested just in the effect of the parameter setting, so the instance effect needs to be "blocked out". In such cases, a one-way ANOVA can still be used if the effect of the other factor, say *B*, is blocked out. In this case, the statistical linear model is

$$Y_{ij} = \mu + \tau_i + \beta_j + \varepsilon_{ij} \begin{cases} i = 1,2, \dots, a \\ j = 1,2, \dots, b \end{cases}, \qquad 3.13$$

with data arranged as in Table 3.3. It is clear from the model that the interaction effect is unaccounted for, as factor *B* is of no interest.

Table 3.3. Data arrangement for a one-way ANOVA model with blocking.

	Block					Sum
	Observations	1	2	…	*b*	
Factor *A*	1	y_{11}	y_{12}	…	y_{1b}	$y_{1.}$
	2	y_{21}	y_{22}	…	y_{2b}	$y_{2.}$
	⋮	⋮	⋮	⋮	⋮	⋮
	a	y_{a1}	y_{a2}	…	y_{ab}	$y_{a.}$
Overall sum						$y_{..}$

Whether one-way or two-way ANOVA is used, the treatments can be selected in two different ways. One is to have the experimenter specifically choose some factor levels, in which case the treatments are considered *fixed*, and whatever conclusions drawn based on them, cannot be extended to others that were not tested. This is known as the *fixed effects* model. The other is to have the experimenter select a random sample of treatments from a larger population of possible treatments, in which case the treatments are considered *random* variables, and one can extend conclusions to other treatments that were not tested. This is known as the *random effects* model. A two-way ANOVA may have one fixed factor and one random factor, leading to what is known as a *mixed effects* model.

In a fixed effects model of two factors, the treatment effects τ_i, β_j, and their interaction $(\tau\beta)_{ij}$ are seen as deviations from the overall mean μ, such that

$$\sum_{i=1}^{a}\tau_i = 0, \sum_{j=1}^{b}\beta_j = 0, \sum_{i=1}^{a}\tau\beta_{ij} = 0, \sum_{j=1}^{b}\tau\beta_{ij} = 0\,, \qquad 3.14$$

and one is interested in testing the null hypotheses H_0 that all treatment means, and their interactions, are zero, against the alternative hypotheses H_1 that at least one is not. Hence, if H_0 is true, each observation is composed of the overall mean plus a realization of the random error term. That is to say, all observations are drawn from a single normal distribution with parameters μ and σ^2, implying that changing between treatments has no effect on the overall mean response. See 3.15.

$$\begin{array}{ll} H_0\colon \tau_1 = \tau_2 = \ \dots = \tau_a = 0 \\ H_1\colon \tau_i \neq 0 \text{ for at least one } i & \text{no effect for factor A} \\ \\ H_0\colon \beta_1 = \beta_2 = \ \dots = \beta_b = 0 \\ H_1\colon \beta_j \neq 0 \text{ for at least one } j & \text{no effect for factor B} \quad 3.15 \\ \\ H_0\colon \tau\beta_{11} = \tau\beta_{12} = \ \dots = \tau\beta_{ab} = 0 \\ H_1\colon \tau\beta_{ij} \neq 0 \text{ for at least one } ij. & \text{no interaction effect} \end{array}$$

When the factors are random variables, it is assumed that they are independently and normally distributed with means of zero and variances $\sigma_\tau^2, \sigma_\beta^2, \sigma_{\tau\beta}^2, \text{and } \sigma^2$, such that $V(Y_{ijk}) = \sigma_\tau^2 + \sigma_\beta^2 + \sigma_{\tau\beta}^2 + \sigma^2$. However, 3.14 no longer holds here, and it carries little meaning to test hypotheses as in 3.15, since the treatments are only random samples from a larger population. To be able to generalize the results from the treatment sample to the treatment population, a more appropriate set of hypotheses is

$$\begin{array}{ll} H_0\colon \sigma_\tau^2 = 0 & \\ H_0\colon \sigma_\tau^2 > 0 & \text{no effect for factor A} \\ & \\ H_0\colon \sigma_\beta^2 = 0 & \\ H_0\colon \sigma_\beta^2 > 0 & \text{no effect for factor B} \\ & \\ H_0\colon \sigma_{\tau\beta}^2 = 0 & \\ H_0\colon \sigma_{\tau\beta}^2 > 0. & \text{no interaction effect} \end{array} \quad 3.16$$

For a one-way ANOVA model, without blocking, 3.14-3.16 reduce only to a single factor and no interaction term. In case of blocking, 3.14 reduces to $\sum_{i=1}^{a}\tau_i = 0, \sum_{j=1}^{b}\beta_j = 0$, while 3.15 and 3.16 reduce to a single factor with no interaction.

Whether the factors are fixed or random, ANOVA divides the total variability in the data into variability due to treatment(s), variability due to interaction, and variability due to random error. For a two-way model, the total variability (SS_T) is expressed in terms of the sum of squared differences between individual readings and the overall mean.

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n}(y_{ijk} - \overline{y}_{...})^2 . \quad 3.17$$

Variability due to treatment(s) (SS_{Treat}) is expressed as the sum of squared differences between the treatment average and the overall mean.

$$SS_A = bn\sum_{i=1}^{a}(\overline{y}_{i..} - \overline{y}_{...})^2 \quad 3.18$$

$$SS_B = an\sum_{j=1}^{b}\left(\overline{y}_{.j.} - \overline{y}_{...}\right)^2. \quad 3.19$$

Variability due to the interaction between factors (SS_{AB}) can be expressed as

$$SS_{AB} = n\sum_{i=1}^{a}\sum_{j=1}^{b}\left(\left(\overline{y}_{ij.} + \overline{y}_{...}\right) - \left(\overline{y}_{i..} + \overline{y}_{.j.}\right)\right)^2, \quad 3.20$$

and random error variability is expressed as the sum of squared differences between individual readings and the cell average (the intersection of two factor levels in Table 3.1).

$$SS_E = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n}\left(y_{ijk} - \overline{y}_{ij.}\right)^2 \quad 3.21$$

$$SS_T = SS_A + SS_B + SS_{AB} + SS_E. \tag{3.22}$$

In case of a single factor, without blocking, 3.17-3.22 reduce to

$$\sum_{i=1}^{a}\sum_{j=1}^{n}\left(y_{ij} - \bar{y}_{..}\right)^2 = n\sum_{i=1}^{a}\left(\bar{y}_{i.} - \bar{y}_{..}\right)^2 + \sum_{i=1}^{a}\sum_{j=1}^{n}\left(y_{ij} - \bar{y}_{i.}\right)^2 \tag{3.23}$$

$$SS_T = SS_{Treat} + SS_E, \tag{3.24}$$

and in case of blocking they reduce to

$$\sum_{i=1}^{a}\sum_{j=1}^{b}\left(y_{ij} - \bar{y}_{..}\right)^2 = b\sum_{i=1}^{a}\left(\bar{y}_{i.} - \bar{y}_{..}\right)^2 + a\sum_{j=1}^{b}\left(\bar{y}_{.j} - \bar{y}_{..}\right)^2 + \sum_{i=1}^{a}\sum_{j=1}^{b}\left(\left(y_{ij} + \bar{y}_{..}\right) - \left(\bar{y}_{i.} + \bar{y}_{.j}\right)\right)^2 \tag{3.25}$$

$$SS_T = SS_{Treat} + SS_{Block} + SS_E. \tag{3.26}$$

Sometimes, a single factor experiment, without blocking, may have different number of observations under each treatment. This is known as the *unbalanced* experiment. It is computed as

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{n_i} {y_{ij}}^2 - \frac{{y_{..}}^2}{N} \tag{3.27}$$

$$SS_{Treat} = \sum_{i=1}^{a}\frac{{y_{i.}}^2}{n_i} - \frac{{y_{..}}^2}{N} \tag{3.28}$$

$$SS_E = SS_T - SS_{Treat}, \tag{3.29}$$

where, n_i is the number of observations taken from treatment i and $N = \sum_{i=1}^{a} n_i$.

Once the *SS* terms are determined, the factor(s) and their interactions are considered significant if their calculated test statistic, which follows an *F*-distribution, exceeds some critical *f* value at a chosen significance level. Calculation details for a *fixed* effects model are given in Tables 3.4-3.6.

Table 3.4. Test statistic calculations of a fixed effects two-way ANOVA model.

Variation source	*SS*	Degrees of freedom	Mean Squares	*F*-statistic	*f* critical
A treatments	SS_A	$a – 1$	$MS_A = SS_A / a – 1$	MS_A / MS_E	$f_{\alpha,\ a – 1,\ ab(n-1)}$
B treatments	SS_B	$b – 1$	$MS_B = SS_B / b – 1$	MS_B / MS_E	$f_{\alpha,\ b – 1,\ ab(n-1)}$
AB interaction	SS_{AB}	$(a – 1)(b – 1)$	$MS_{AB} = SS_{AB} / (a – 1)(b – 1)$	MS_{AB} / MS_E	$f_{\alpha,\ (a – 1)(b – 1),\ ab(n-1)}$
Error	SS_E	$ab(n – 1)$	$MS_E = SS_E /ab(n – 1)$		
Total	SS_T	$abn – 1$			

Table 3.5. Test statistic calculations of a fixed effects one-way ANOVA model without blocking.

Variation source	*SS*	Degrees of freedom	Mean Squares	*F*-statistic	*f* critical
Treatments	SS_{Treat}	$a – 1$	$MS_{Treat} = SS_{Treat} / a – 1$	MS_{Treat} / MS_E	$f_{\alpha,\ a – 1,\ a(n – 1)}$
Error	SS_E	$a(n – 1)$	$MS_E = SS_E / a(n – 1)$		
Total	SS_T	$an – 1$			

Table 3.6. Test statistic calculations of a fixed effects one-way ANOVA model with blocking.

Variation source	*SS*	Degrees of freedom	Mean Squares	*F*-statistic	*f* critical
Treatments	SS_{Treat}	$a – 1$	$MS_{Treat} = SS_{Treat} / a – 1$	MS_{Treat} / MS_E	$f_{\alpha,\ a – 1,\ (a – 1)(b – 1)}$
Blocks	SS_{Blocks}	$b – 1$	$MS_{Blocks} = SS_{Blocks} / b – 1$		
Error	SS_E	$(a – 1)(b – 1)$	$MS_E = SS_E / (a – 1)(b – 1)$		
Total	SS_T	$ab – 1$			

The error term of a two-way ANOVA has $ab(n – 1)$ degrees of freedom, which means if only one observation per cell is available ($n = 1$), the MS_E cannot be calculated and no further analysis can be conducted unless one assumes that the interaction effect is negligible and uses MS_{AB} instead of MS_E. In this case, a two-way ANOVA model reduces to a one-way ANOVA model with blocking.

Calculations of the *F*-statistic for random factors are a bit different. For a two-way mixed effects model, where factor *B* is random, MS_A is divided by MS_{AB} instead of MS_E. See Table 3.7. However, for a one-way model with blocking, if the treatments and/or blocks are random, then the *F*-statistic will still be calculated as MS_{Treat}/MS_E. The same holds for a one-way model without blocking.

Table 3.7. Test statistic calculations of a mixed effects two-way ANOVA model.
Factor *B* is random and factor *A* is fixed.

Variation source	*SS*	Degrees of freedom	Mean Squares	*F*-statistic	*f* critical
A treatments	SS_A	$a – 1$	$MS_A = SS_A / a – 1$	MS_A / MS_{AB}	$f_{\alpha,\ a – 1,\ (a – 1)(b – 1)}$
B treatments	SS_B	$b – 1$	$MS_B = SS_B / b – 1$	MS_B / MS_E	$f_{\alpha,\ b – 1,\ ab(n-1)}$
AB interaction	SS_{AB}	$(a – 1)(b – 1)$	$MS_{AB} = SS_{AB} / (a – 1)(b – 1)$	MS_{AB} / MS_E	$f_{\alpha,\ (a – 1)(b – 1),\ ab(n-1)}$
Error	SS_E	$ab(n – 1)$	$MS_E = SS_E /ab(n – 1)$		
Total	SS_T	$abn – 1$			

Finally, an unbalanced two-way ANOVA is possible in terms of having an unequal number of replications per cell *n's*, not in terms of having unequal number of observations per factor. Think of trying to conduct a paired *t*-test with unequal sample sizes! This type is not discussed here as it will not be used.

3.4.1.2 Violating normal-model based ANOVA assumptions

The effects of violating the assumptions of ANOVA models, both fixed and random, have been studied extensively, and now there is a general consensus on how the *nominal* Type I and Type II error rates (α and β) could change if one, or more, of the assumptions are not met. This section limits the discussion to the fixed effects one-way ANOVA model, as it is beyond the scope of this work to review the effect on all ANOVA models. In specific, the effect of violating the independence, normality, and equal variances assumptions are investigated. The interested reader is directed to the works of Box and Andersen (1954) and Scheffé (1999) for random effects models, and Bradley (1952) and Elashoff (1969) for fixed effects models.

Before going into the smaller details, two points are worth mentioning: first, while non-parametric versions of ANOVA models are available, one should not rush to applying them simply because some assumptions are violated; instead, one should be concerned about the extent to which the violation(s) will change the nominal error rates (Glass *et al.*, 1972). Second, since many of these studies are computer simulations, which empirically evaluate error rates and then compare them to the nominal ones, some disagreement between different studies is expected.

Violating the independence assumption perhaps has the greatest impact on α and β, compared to the other two assumptions, even if treatments have equal sample sizes n (Glass *et al.*, 1972). It will be shown shortly that equal n's reduces the effect of violating the other assumptions. Cochran (1968) and Scheffé (1999) showed that negative correlation yields more conservative tests (i.e. actual α is less than the nominal α), while a positive correlation yields more liberal tests (i.e. actual α is greater than the nominal α). See Table 3.8.

Table 3.8*. Effect of correlation on the actual α values of a one-way ANOVA. Nominal α = 0.05.

Correlation	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4
Actual α	1.100^{-5}	0.002	0.011	0.028	0.050	0.074	0.098	0.120	0.140

*Reproduced from Scheffe' (1959, p. 339).

Violating the equal variances assumption can have a large effect on the nominal α levels if the n's are *unequal*. Based on the works of Box and Andersen (1954), Pratt (1964), and Hsu and Feldt (1969), Glass *et al.* (1972) made the following remarks on heterogeneous variances and unequal n's: first, actual α levels tend to be higher when small n's come from populations with larger variances. Second, actual α levels tend to be lower when small n's come from populations with smaller variances. Third, heterogeneous variances seem to have little, or no, effect on the actual α levels if the n's are equal. See Tables 3.9-3.11. The third remark holds for β as well (Horsnell, 1953).

Table 3.9. Effect of heterogeneous variances on the actual α values of a one-way ANOVA model with various sample sizes. Nominal α = 0.05.

	σ_1/σ_2						
n_1/n_2	*0.1*	*0.2*	*0.5*	*1*	*2*	*5*	*10*
3*	0.317	0.178	0.098	0.050	0.025	0.008	0.005
1.67*	0.216	0.103	0.072	0.050	0.038	0.031	0.030
1*	**0.072**	**0.058**	**0.051**	**0.050**	**0.051**	**0.058**	**0.063**
1**	**0.050**	**0.050**	**0.050**	**0.050**	**0.050**	**0.050**	**0.050**
2**	0.170	0.120	0.080	0.050	0.029	0.014	NA
5**	0.380	0.220	0.120	0.050	0.014	0.002	NA

* Reproduced from Hsu and Feldt (1969)
** Reproduced from Scheffé (1999)

Table 3.10*. Effect of heterogeneous variances on the actual α values of a one-way ANOVA model with various sample sizes.

Ratio of σ^2's	Sample sizes	Nominal α	Actual α
1:2:3	**5, 5, 5**	0.05	**0.056**
	3, 9, 3	0.05	0.056
	7, 5, 3	0.05	0.092
	3, 5, 7	0.05	0.040
1:1:3	**5, 5, 5**	0.05	**0.059**
	7, 5, 3	0.05	0.110
	9, 5, 1	0.05	0.170
	1, 5, 9	0.05	0.013
1:1:1:1:3	**5, 5, 5, 5, 5**	0.05	**0.074**
	9, 5, 5, 5, 1	0.05	0.14
	1, 5, 5, 5, 0	0.05	0.025

* Reproduced from Box and Andersen (1954)

Table 3.11*. Effect of heterogeneous variances on the actual α values of a one-way ANOVA model with equal sample sizes ($n_1 = n_2 = 10$).

σ_1^2/σ_2^2	Nominal α	Actual α
219/69	0.05	0.050
252/36	0.05	0.054
279/9	0.05	0.066
278/1	0.05	0.062

* Reproduced from Young and Veldman (1963)

Violating the normality assumption, usually measured in terms of skewness and kurtosis, does not seem to have major consequences on the actual α and β values (Lindquist, 1953, Games and Lucas, 1966), see Tables 3.12-3.13. Other researchers concluded the same (Rider, 1929, Egon, 1931, Srivastava, 1959, Young and Veldman, 1963, Cochran, 1968). The effect of using binomial data (i.e. dichotomous variables) with a one-way ANOVA was investigated in Lunney (1970), where he showed that actual α and β values are very close to the nominal ones for *equal n*'s. Reproduced results are in Tables 3.14-3.15.

Table 3.12*. Effect of non-normality on the actual α values for a one-way ANOVA, with different number of treatments and sample sizes.

		Nominal α levels								
Skewness	Kurtosis	*0.50*	*0.25*	*0.20*	*0.10*	*0.050*	*0.025*	*0.010*	*0.005*	*0.001*
0	3.0	0.51	0.25	0.20	0.10	0.056	0.029	0.014	0.007	0.002
0	7.0	0.53	0.28	0.23	0.13	0.078	0.046	0.028	0.015	0.008
0	7.0	0.54	0.29	0.22	0.11	0.066	0.038	0.016	0.010	0.004
0	1.7	0.48	0.25	0.20	0.12	0.061	0.032	0.018	0.009	0.001
0.5	3.7	0.51	0.25	0.20	0.10	0.052	0.028	0.013	0.007	0.001
1.0	3.8	0.50	0.24	0.19	0.10	0.048	0.021	0.008	0.004	0.001
1.0	3.8	0.51	0.26	0.20	0.10	0.048	0.023	0.010	0.007	0.001
1.4	3.6	0.51	0.24	0.19	0.09	0.048	0.026	0.010	0.005	0.002

* Reproduced from Lindquist (1953)

Table 3.13*. Effect of non-normality on the actual β values for a one-way ANOVA, with different number of treatments and sample sizes.

		Nominal β levels								
Skewness	Kurtosis	*0.050*	*0.087*	*0.097*	*0.206*	*0.261*	*0.417*	*0.540*	*0.851*	*0.954*
0.00	2.90	0.059	0.098	0.091	0.200	0.275	0.430	0.519	0.851	0.945
0.45	3.53	0.052	0.096	0.096	0.193	0.266	0.410	0.556	0.848	0.947
0.00	2.91	0.046	0.098	0.098	0.201	0.286	0.518	0.575	0.834	0.946
0.64	3.53	0.054	0.100	0.100	0.188	0.273	0.446	0.563	0.858	0.943
0.00	2.82	0.057	0.097	0.097	0.213	0.268	0.445	0.594	0.828	0.941
2.04	9.54	0.037	0.078	0.078	0.264	0.326	0.541	0.559	0.854	0.910
0.03	2.88	0.057	0.120	0.120	0.325	0.485	0.536	0.657	0.792	0.964
0.00	9.16	0.036	0.109	0.109	0.269	0.338	0.522	0.584	0.874	0.924

* Reproduced from Games and Lucas (1966)

Table 3.14*. Effect of binomial data on the actual αvalues for a one-way ANOVA, with various numbers of treatments and equal sample sizes.

	Nominal α values			
Number of treatments	*0.10*	*0.05*	*0.025*	*0.01*
2	0.101	0.048	0.023	0.010
3	0.099	0.051	0.026	0.011
4	0.098	0.047	0.024	0.010
5	0.099	0.051	0.026	0.010

* Reproduced from Lunney (1970)

Table 3.15[*]. Effect of binomial data on the actual β values for a one-way ANOVA, with different sample sizes and a constant number of treatments.

Number of treatments	α	Sample size	Nominal 1-β values	Actual 1-β values
3	0.01	11	0.8	0.764
3	0.05	11	0.8	0.801
3	0.01	11	0.6	0.607
3	0.05	11	0.6	0.602
3	0.01	21	0.8	0.782
3	0.05	21	0.8	0.846
3	0.01	21	0.6	0.616
3	0.05	21	0.6	0.594

* Reproduced from Lunney (1970)

3.4.1.3 Applicability of ANOVA models to Racing algorithms

Since the sample sizes in Racing algorithms are always equal, the effect of non-normality and unequal variances is greatly reduced, and correlation can be dealt with using blocking or two-way ANOVA. A-Race has been used to tune algorithm parameters by Schaffer *et al.* (1989) and Yuan and Gallagher (2004); however, it is unclear if they used any of the proposed versions here, or if the actual α and β values were empirically calculated and compared to the nominal ones.

With that in mind, the reader should note that Racing is not just a single ANOVA test, it is a series of tests applied to a decreasing number of parameter settings, and an increasing number of sample sizes. In addition, the tests are not independent as Racing is a sequential procedure. This makes it difficult to predict the true α and β values, in both parametric and non-parametric Racing, see Branke and Elomari (2013).

In this work, the ANOVA models used for A-Race are either two-way (A-Race_2Way), with the parameter settings factor treated as fixed and the instances factor treated as random. Or, one-way unbalanced without blocking (A-Race_1WUB), with the parameter settings treated as a fixed factor. In both cases, ANOVA is followed by Fisher's LSD test, which declares any pairs of means μ_i and μ_j to be different if

$$|\bar{y}_{i.} - \bar{y}_{j.}| > t_{\alpha/2,N-a}\sqrt{MS_E\left({}^{1}/_{n_i} + {}^{1}/_{n_j}\right)}, \qquad 3.30$$

where a is the number of levels of factor A, n_k is the sample size of treatment k, and N is the total number of samples for all parameter settings. The right hand side of the inequality is known as Fisher's LSD. As 3.30 does not cap the family-wise error rate (explained later), α is divided by twice the number of comparisons made.

3.4.2 Non-parametric Racing

Switching to a non-parametric Racing algorithm does overcome some of the difficulties discussed previously. Nevertheless, since non-parametric procedures require transforming the data into a rank-based format, some information is sacrificed, leading to less powerful tests (Sheskin, 2004, p.388).

The most widely used non-parametric Racing algorithm, for offline tuning, is *F*-Race and its iterative version *I\F*-Race (Birattari *et al.*, 2010). It uses Friedman's two-way ANOVA by ranks as a first stage filtering, followed by a non-parametric pair-wise comparison test. A one-way ANOVA by ranks is also possible, and is known as the Kruskal-Wallis test. The author is unaware of any implementation of this test into a Racing algorithm, for the purposes of offline tuning. Still, it will be implemented in this work as *KW*-Race. The details of these tests are presented in the next two sections, based on the works of Conover (1999) and Sheskin (2004).

3.4.2.1 *F*-Race

This version of Racing implements Friedman's two-way ANOVA by ranks with no interaction effect (Friedman, 1937). The null hypothesis is: in a set of k *dependent* samples (k > 2), all samples represent medians Θ of the same population, while the alternative hypothesis is: at least two samples represent two medians of two different populations.

$$\begin{gathered} H_0'\colon \theta_1 = \theta_2 = \cdots = \theta_k \\ H_1'\colon Not\ H_0'. \end{gathered} \qquad 3.31$$

If the *F*-test is significant, it means that there is a significant difference between at least two of the sample medians. This, in turn, should trigger a series of follow-up tests to identify them. In conducting the test, the data is assumed to be in a rank-based format, or can be transformed into a rank-based format.

Data arrangement of the *F*-test is shown in Table 3.16. The columns represent treatments, and the rows constitute B mutually independent k-variate random variables $[y_{i1}, y_{i2}, \dots, y_{ik}]$, known as blocks. The individual readings y_{ij} are ranked per block as $r(y_{ij})$, for example, in block 2 if the second treatment has the lowest value, it is ranked 1, the second smallest is ranked 2, and so on. If some y_{ij} values are equal in block *i*, then each of them will get a rank equal to the average of the ranks they would have received if they were not equal, but are still less than y_{ij+1}, see Table 3.17 for an example. Ranking in a reverse order is also possible (i.e. the largest reading gets a rank of 1). Rank totals are then calculated per treatment *j* as

$$R_j = \sum_{i=1}^{B} r(y_{ij}) \ \forall j = 1,2, \dots, k, \tag{3.32}$$

and if H_0 is true, then one expects $R_1 = R_2 = \cdots = R_k$.

Table 3.16. Data arrangement for the *F*-test.

	Treatment			
Block	1	2	…	*k*
1	$r(y_{11})$	$r(y_{12})$	…	$r(y_{1k})$
2	$r(y_{21})$	$r(y_{22})$	…	$r(y_{2k})$
…	…	…	…	…
B	$r(y_{B1})$	$r(y_{B2})$	…	$r(y_{Bk})$
Rank totals	R_1	R_2	…	R_k

Table 3.17. An example of the ranking scheme of the *F*-test.

Original data	Ranked data no adjusted for ties	Ranked data adjusted for ties
[1.1, 2, 2,2, 3.3, 7, 9]	[1,2,3,4,5,6,7]	[1,3,3,3,5,6,7]

Probably the most important assumption of the *F*-test is that the *B* blocks are mutually independent. Other assumptions include: the readings per treatment have been randomly selected, and that they are continuous random variables. The final assumption is not often obeyed though (Sheskin, 2004, p.830).

The test statistic for the *F*-test is approximated with a Chi-square distribution with *k*-1 degrees of freedom. In case there are no, or few, ties, its value is calculated as

$$\chi_r^2 = \frac{12}{Bk(k+1)}\left[\sum_{j=1}^{k}\left(R_j\right)^2\right] - 3B(k+1). \qquad 3.33$$

Daniel (1990, p.226) suggested a correction factor in case of excessive ties

$$C = 1 - \frac{\sum_{i=1}^{s}(t_i^3 - t_i)}{B(k^3 - k)}, \qquad 3.34$$

where *s* is the number of sets of ties, and t_i is the number of tied scores in the i^{th} set. In the example in Table 3.17 there is one set of ties (s = 1) and there are 3 tied scores in that set. A test statistic adjusted for ties is calculated by dividing 3.33 by 3.34.

$$\chi_{rc}^2 = \frac{\chi_r^2}{C}. \qquad 3.35$$

Following a significant *F*-test, pair-wise comparisons are carried out. Daniel (1990, p.231) proposed calculating a minimum difference (CD_F) in rank sums (R_i and R_j), required for treatments *i* and *j* to be significantly different at a specified type I error rate α.

$$CD_F = z\sqrt{\frac{Bk(k+1)}{6}}, \qquad 3.36$$

where *z* can be obtained from normal distribution tables. Note that for the individual comparisons, the maximum family-wise type I error rate α_{FW}, which the user sets, should be adjusted based on the number of comparisons *C*. For a one-tail test *z* is evaluated at α_{FW}/C, and for a two-tail test z is evaluated at $\alpha_{FW}/2C$.

Any absolute difference between treatment ranks sums $\left|R_i - R_j\right|$ that is greater than or equal to CD_F, means that treatments *i* and *j* come from two populations with unequal medians. For a Racing algorithm, not all pair-wise comparisons are conducted, only those with the current best R_b (i.e. $|R_b - R_i|$). If $|R_b - R_i| \geq CD_F$, then parameter setting *i* should be dropped out from the race.

Conover (1999, p.371) proposed another test, where treatments *i* and *j* are considered to represent populations with different medians if the following inequality is satisfied.

$$|R_i - R_j| > t_{1-\frac{\alpha}{2}} \sqrt{\left[\frac{2\left(B \sum_{i=1}^{B} \sum_{j=1}^{k} \left[r(y_{ij})\right]^2 - \sum_{j=1}^{k} R_j^2\right)}{(B-1)(k-1)}\right]}, \qquad 3.37$$

where $t_{1-\alpha/2}$ can be obtained from *t* distribution tables with *(B*-1*)(k*-1*)* degrees of freedom. In this work, *F*-Race is implemented as it is in literature using 3.33 and 3.37.

3.4.2.2 *KW*-Race

This version of Racing implements the one-way ANOVA by ranks test of Kruskal and Wallis (1952). The null hypothesis is: for *k independent* samples (*k > 2*), all samples represent medians *Θ* of the same population, while the alternative hypothesis is: at least two samples represent medians of two different populations.

$$\begin{gathered} H_0'': \theta_1 = \theta_2 = \cdots = \theta_k \\ H_1'': Not\ H_0''. \end{gathered} \qquad 3.38$$

If the *KW*-test is significant, it means that there is a significant difference between at least two of the samples. In that case, follow-up tests should be conducted to identify these populations. Data arrangement is similar to that of an *F*-test, but the ranking procedure is different, see Table 3.18. Instead of ranking the individual readings y_{ij} per row *i*, all y_{ij} values are pooled together, sorted, and ranked such that the lowest value gets a rank of 1, and the highest gets a rank of $n \cdot k$. Ranking can be reversed and ties are dealt with as in the *F*-test. The ranks are then plugged back into their original positions, and treatment averages are calculated as

$$\bar{R}_j = \frac{\sum_{i=1}^{n} r(y_{ij})}{n} \; \forall j = 1,2, \dots, k. \qquad 3.39$$

Table 3.18. Data arrangement for the *KW*-test.

	Treatment			
Sample	1	2	…	k
1	$r(y_{11})$	$r(y_{12})$	…	$r(y_{1k})$
2	$r(y_{21})$	$r(y_{22})$	…	$r(y_{2k})$
…	…	…	…	…
n	$r(y_{n1})$	$r(y_{n2})$	…	$r(y_{nk})$
Rank averages	$\overline{R_1}$	$\overline{R_2}$	…	$\overline{R_k}$

Assumptions for the *KW*-test include independence of the n samples, individual readings are continuous and are drawn at random, and that the populations distributions are identical in shape, but there is no requirement that they follow any specific distribution (Marascuilo and Mcsweeney, 1977, pp.300-303, Daniel, 1990, pp.200-202, Conover, 1999, p.289). The final assumption does indicate that the populations' distributions should have equal dispersion measures (Maxwell and Delaney, 2003, pp.141-142). However, some work suggests that the sampling distribution of the *KW*-test statistic is robust to violating this assumption, mainly due to the ranking scheme used, which tends to reduce the effect of outliers (Sheskin, 2004, p.745).

The test statistic for the *KW*-test is approximated by a Chi-square distribution, with k-1 degrees of freedom. In case there are no, or few, ties, the value of the test statistic is calculated as

$$H = \frac{12}{N(N+1)} \sum_{j=1}^{k} \left[\frac{\sum_{i=1}^{n} r(y_{ij})^2}{n_j} \right] - 3(N+1), \qquad 3.40$$

where N is the total number of observations $n \cdot k$. A correction factor may be used in case of excessive ties, it is calculated as

$$C = 1 - \frac{\sum_{i=1}^{s} (t_i^3 - t_i)}{N^3 - N}, \qquad 3.41$$

where s and t are interpreted as in 3.34 and the corrected test statistic is again obtained by dividing 3.40 by 3.41.

Daniel (1990, p.203) proposed calculating a minimum difference (CD_{KW}) in ranks sums (R_i and R_j), required for treatments i and j to be significantly different at a specified α.

$$CD_{KW} = z\sqrt{\frac{N(N+1)}{12}\left(\frac{1}{n_i}+\frac{1}{n_j}\right)}. \qquad 3.42$$

Where z is obtained from normal distribution tables and α_{FW} is adjusted based on the number of comparisons C. For a one-tail test z is evaluated at α_{FW}/C, and for a two-tail test z is evaluated at $\alpha_{FW}/2C$. Any absolute difference between treatment ranks averages $\left|\bar{R}_i - \bar{R}_j\right|$ that is greater than or equal to CD_{KW}, indicates that they come from two populations with unequal medians. For a Racing algorithm, not all pair-wise comparisons are needed, only those with the current best (i.e. $|\bar{R}_b - \bar{R}_i|$). If $|\bar{R}_b - \bar{R}_i| \geq CD_{KW}$, then parameter setting i should be dropped out from the race.

All Racing algorithms follow the same general framework, the only difference is the set of statistical tests applied. See Figure 3.12.

```
Algorithm 3.4: A Racing algorithm
Input: $k$, $N$, $\alpha$, $n_0$                  //$n_0$ is the number of instances
                                                   //for the initial phase
Initialize
   $t \leftarrow 0$; $c \leftarrow 0$              //iteration and consumed budget
                                                   //counters
   $s \leftarrow \{1,2,\dots,k\}$                  //s is the set of surviving
                                                   //parameter settings
   run all parameter settings on $n_0$             //initial performance estimate of
                                                   //each parameter setting
   $t \leftarrow n_0$; $c \leftarrow n_0 \cdot |s|$
Main allocation loop
 while $c \leq N$ and $|s| > 1$
    $\bar{J}_l = \frac{\sum_{i=1}^{n_l} \omega_{i,l}}{n_l}$  $\forall l \in s$   //update the performance measure
                                                   //$\omega_{i,l}$ is the $i^{th}$ sample/rank of
                                                   //system l
    run an ANOVA test on $s$                       //normal-model based, F-test, or
                                                   //KW-test
    if ANOVA test is significant                   //first stage filtering
    {
      find $b = arg\min_l \bar{J}_l, l \in s$      //find the current best
      pair-wise comparisons with $b$               //select tests corresponding to
                                                   //the chosen ANOVA
      if pair-wise comparison is significant
         $s \leftarrow s \setminus l$              //second stage filtering
    }
    run $s$ on a new instance
    $c \leftarrow c + |s|$                         //update consumed
    $t \leftarrow t + 1$                           //next iteration
 end while
Output: $s$                                        //surviving parameter setting(s)
```

Figure 3.12. A pseudo code of a Racing algorithm.

3.5 Racing with reset (2nd contribution)

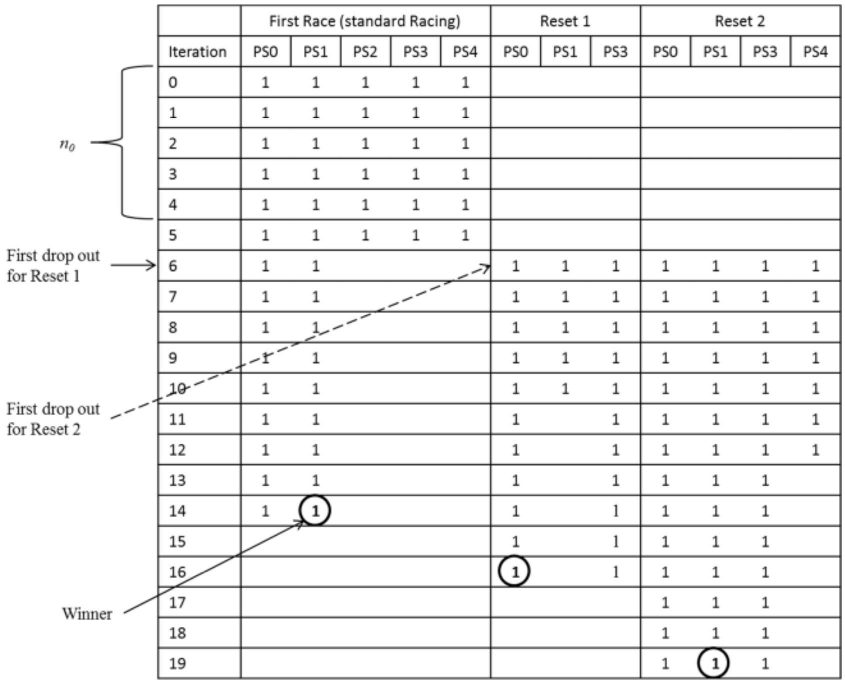
As should be clear by now, Racing may terminate before the permitted budget is consumed. In situations where there is a fixed budget and there is no gain from early termination, the algorithm's failure rate f_r can be reduced by utilizing the "*excess*" budget. f_r is defined as the percentage of runs the algorithm fails to select the true best, either because it was discarded at some point during the race, or the race terminates with more than one parameter setting, and the true best has an inferior aggregated performance.

Starting with a relatively high *initial* significance level α_0 (say 0.3 or 0.2), Racing is run until a single parameter setting remains. As α_0 is quite large, it is likely that the race terminates before N is consumed. If this happens, the algorithm rolls back to the iteration where the first parameter setting was discarded, decreases α_0 by a factor of γ, and then re-runs the statistical tests on *all* the parameter settings. Since α is smaller now, Racing will be less likely to discard parameter settings, and it will either discard all previously dropped parameters, or pick some up. This online adaptation of α allows the true best a chance to get back into the race if it was discarded at some point. Thus, f_r is reduced. See Figures 3.13 and 3.14. This *reset* is done as many times as needed until N is consumed, each time the previous α is discounted by γ. Obviously, all the samples that were collected before are stored in memory and used in later iterations if needed. Only those parameter settings which had previously been discarded, but now survive due to the lower α, are sampled.

As shown in Branke and Elomari (2013), Racing with reset is fairly robust to γ. Tested values of 0.5 and 0.2 showed almost identical performance over a wide range of α_0. If γ is very small, however, say 0.01, α_0 is reduced by a large amount within a few resets, and Racing's performance will approach that of equal allocation. Higher γ values (e.g. $\geq$ 0.8) allow α_0 to gradually approach its optimal value depending on the number of resets permitted within the available budget. Based on the experiments done in this thesis, it is advisable to set γ between 0.2 and 0.7.

```
Algorithm 3.5: Racing with reset
Input: $k$, $N$, $\alpha$, $n_0$                          //$n_0$ is the number of instances
                                                          //for the initial phase
1   Initialize
2   $t \leftarrow 0$; $c \leftarrow 0$                    //iteration and consumed budget
                                                          //counters
3   $s \leftarrow \{1,2,\dots,k\}$                        //s is the set of surviving
                                                          //parameter settings
4   run all parameter settings on $n_0$                   //estimate performance of each
                                                          //parameter setting
5   $t \leftarrow n_0$; $c \leftarrow n_0 \cdot |s|$
6   Main allocation loop
7     while $c \leq N$ or $|s| > 1$
8       $\bar{J}_l = \frac{\sum_{i=1}^{n_l} \omega_{i,l}}{n_l}$ $\forall l \in s$   //update the performance measure
                                                          //$\omega_{i,l}$ is the $i^{th}$ sample/rank of
                                                          //system l
9       run an ANOVA test on $s$                          //normal-model based, F-test,
                                                          //or KW-test
10      if ANOVA test is significant                      //first stage filtering
11      {
          find $b = arg\min_l \bar{J}_l, l \in s$         //find the current best
12        pair-wise comparisons with $b$                  //select tests corresponding to
                                                          //the chosen ANOVA
13          if pair-wise comparison is significant
14            $s \leftarrow s \setminus l$                //second stage filtering
        }
17      run $s$ on a new instance                         //only if it was not run before
18      if $|s| == 1$
19      {
          find first drop out $t_f$
20        $t \leftarrow t_f$
21        $\alpha \leftarrow \alpha \cdot \rho$           //reset idea
22        $s \leftarrow \{1,2,\dots,k\}$
23        go back to 9
        }
25      update $c$
26      $t \leftarrow t + 1$                              //move to the next iteration
27    end while
Output: $s$                                               //surviving parameter setting(s)
```

Figure 3.13. A pseudo code of a Racing algorithm with a reset.


n_0
First drop out for Reset 1
First drop out for Reset 2
Winner

	First Race (standard Racing)					Reset 1			Reset 2			
Iteration	PS0	PS1	PS2	PS3	PS4	PS0	PS1	PS3	PS0	PS1	PS3	PS4
0	1	1	1	1	1							
1	1	1	1	1	1							
2	1	1	1	1	1							
3	1	1	1	1	1							
4	1	1	1	1	1							
5	1	1	1	1	1							
6	1	1				1	1	1	1	1	1	1
7	1	1				1	1	1	1	1	1	1
8	1	1				1	1	1	1	1	1	1
9	1	1				1	1	1	1	1	1	1
10	1	1				1	1	1	1	1	1	1
11	1	1				1		1	1	1	1	1
12	1	1				1		1	1	1	1	1
13	1	1				1		1	1	1	1	
14	1	1				1		1	1	1	1	
15						1		1	1	1	1	
16						1		1	1	1	1	
17									1	1	1	
18									1	1	1	
19									1	1	1	

Figure 3.14. An example of Racing with reset.

The reset differs depending whether the ANOVA test is one-way or two-way. For *F*-Race and *A*-Race_2Way, the number of observations of the competing systems, after the reset, has to be equal as both algorithms use a two-way ANOVA test. *KW*-Race and *A*-Race_1WUB, however, do not have that restriction and can use all the previously collected data. For example, in Figure 3.14, reset 1 occurs at iteration 5, if a two-way test is applied, only readings 0-5 for parameter settings PS0, PS1, and PS3 can go into the test with the new α. A one-way test, however, can utilize all 15 readings of PS0 and PS1, plus the first 6 readings of PS3.

Depending on the number of resets, α may not become small enough for Racing to pick up the true best again. Therefore, another strategy is added to increase the chances of the true best to re-enter the race if dropped. In addition to decreasing α, all previously discarded parameters are sampled once more (reset and re-sample). This provides a better estimate of their aggregated utilities. Re-sampling for two-way Racing algorithms is expected to degrade performance, as they cannot use all previously collected data, and the re-sampling forces them to become similar to Equal Budget Allocation (EBA), an algorithm which will be shown to be inferior in most cases later on. Hence, they will be tested without the re-sampling idea. One-way Racing, however, is not likely to behave like EBA, and will be tested with the re-sampling idea.

Racing algorithms with reset will be designated as: *F*-RaceR, *A*-RaceR_2Way, *KW*-RaceRR, and *A*-RaceRR_1WUB, where the “R” stands for the reset, and “RR” stands for reset and re-sample.

3.6 One-way Racing with intelligent budget allocation (3rd contribution)

Current Racing algorithms rely on a two-way ANOVA test, which requires having an equal number of samples for each of the competing parameter settings every time the test is run. This means that every time a drop out occurs, the remaining must all be sampled the same number of times (once is the default). The disadvantage is that one cannot allocate the additional budget more intelligently in each iteration. For example, 7 parameter settings are in the race, and the statistical tests did not discard any of them. In the next iteration, there is a budget of $\Delta = 7$ samples that should be distributed among the survivors. Instead of sampling each once more, another algorithm can be run to find a better distribution.

To the extent of the author’s knowledge, this idea was never proposed before, as most of the development focused on *F*-Race, which cannot handle unequal sample sizes. The proposed *KW*-Race and *A*-Race_1WUB, however, can. In this work, *KW*-RaceRR is combined with OCBA (*KW*-RaceRR_OCBA) to see if intelligently allocating Δ will provide a better, or worse, exploration vs. exploitation balance, and ultimately improve, or degrade, its performance. The combination is direct, whenever *KW*-RaceRR attempts to allocate Δ, OCBA is run once to try and make a better use out of Δ. It is possible that OCBA elects to allocate Δ equally at some iteration, in which case no benefit is gained from OCBA.

3.7 Conclusion

This chapter focused on the theory and implementation details of the contributions proposed in this thesis. First, the Flexible Budget method that can find the best parameter settings for any computational budget, less than a specific maximum, in a single run, without compromising solution quality. Hence, it saves computational effort. Second, Racing with reset that utilizes any fixed budget to lower its failure rate. This is done by automatically adapting the significance level parameter. And third, a Racing algorithm that can intelligently allocate Δ compared to equally distributing it among the surviving parameters.

Three versions of the Flexible Budget method were presented. They differ in how the utilities of the parameter settings are calculated. The first uses the length of the convergence curve, the second uses the area under the convergence curve, and the third uses the area lost by not selecting a specific parameter setting. These three versions will be compared to the standard Fixed Budget method to experimentally demonstrate their efficiency. That is, finding the best parameter values in a single run, without compromising solution quality. This is done in Chapter 4.

Different Racing algorithms were presented, those which take advantage of normally distributed data (*A*-Race_2Way and *A*-Race_1WUB), and those which do not require the data to follow any specific distribution (*F*-Race and *KW*-Race). Two more algorithms from the Simulation Optimization literature were suggested as offline tuners, namely OCBA and CBA. They mainly differ from Racing algorithms in that inferior parameter settings, or systems, are never discarded, but are rather sampled less often. All budget allocators will be compared in the Chapter 5 to see under which circumstances each performs best.

CHAPTER 4 Experiments, Results, and Analysis

Meta-Optimization with a Flexible Budget

4.1 Introduction

This chapter describes the experiments conducted to validate the first contribution, the Flexible Budget method, and assess its performance against other algorithms currently used in the literature. The chapter is organized as follows: Section 4.2 presents the experimental setup (competing algorithms, performance measures, and training and testing sets), followed by the results and analysis in Section 4.3. Section 4.4 concludes this chapter.

4.2 Experimental setup

4.2.1 Competing algorithms and performance measures

A meta-optimizer with a fixed budget is compared to a meta-optimizer with a flexible budget to test the null hypothesis H_0: the difference in solution quality obtained by both methods is equal to zero for various computational budgets, against the alternative hypothesis H_1: the difference is not equal to zero. To do so, the Flexible Budget is first run *once*, optimizing parameters for any budget less than n_{max}. Then, several computational budgets are selected for the Fixed Budget $\{b_1, \dots, b_k : b_i \leq n_{max} \; \forall \; i = 1, \dots, k\}$, which is run k times optimizing parameters for a specific b_i in each run. Finally, solution qualities $\{q_{b_1}, \dots, q_{b_k}\}_{Fixed}$ and $\{q_{b_1}, \dots, q_{b_k}\}_{Flexible}$ are compared. The expectation is that, over many replications, these quantities will be statistically indifferent, at a given significance level. If this is the case, then the Flexible Budget is shown to have saved a lot of time, depending on k, while maintaining the same solution quality. See Figure 4.1.

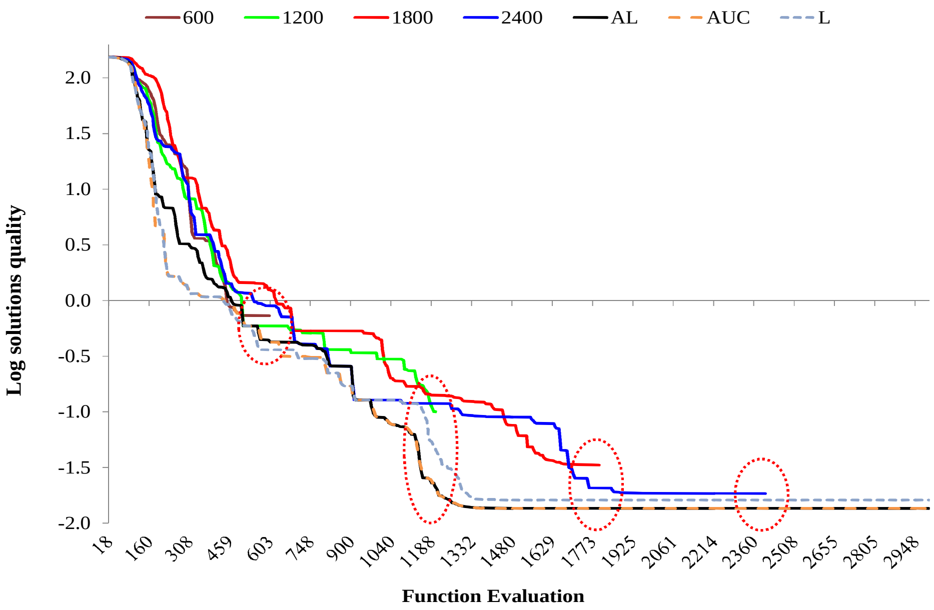


Figure 4.1. Comparison of the Fixed and Flexible Budget methods at different computational budgets. In this example, the Flexible Budget, using L, AUC, or AL, was able to find a better (lower) solution quality at each of the four selected budgets less than 3000 (n_{max}).

The lower-level algorithm is the derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)[1]. This (μ_W, λ)-CMA-ES generates λ offspring for the next generation g+1, from the current population of μ individuals, according to a multi-variate normal distribution with a weighted mean and a covariance matrix.

$$x_k^{g+1} \sim \mathcal{N}\left(x_w^g, {\sigma^g}^2 C^g\right), k = 1, \dots, \lambda, \qquad 4.1$$

where the recombination point $x_w^g = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^g$ is a weighted average of the selected individuals, such that the weights $w_i > 0$ for all $i = 1, \dots, \mu$ and $\sum_{i=1}^{\mu} w_i = 1$. C is the covariance matrix with a step size of σ. The detailed workings of CMA-ES are beyond the scope of this work, the reader is directed to Hansen and Kern (2004) for more details.

Five parameters are to be tuned for CMA-ES, they are: λ, μ, σ_0 (the initial step size), recombination type, and update type. The recombination type parameter specifies the w_i's. Three types are used: *Equal* where $w_i = 1/\mu$, *Linear* where $w_i = \mu - i$, and *Superlinear* (non-linear) where $w_i = log(\mu + 1) - log(i + 1)$. The update type parameter specifies the

[1] CMA-ES is implemented from the Shark library http://image.diku.dk/shark/sphinx_pages/build/html/index.html. See Igel, C., Heidrich-Meisner, V. and Glasmachers, T., 2008. Shark. *Journal of Machine Learning Research*, 9 (1), pp.993-996.)

rank of the matrix used to update *C*. It is either a *Rank-μ* matrix, or a *Rank*-1 matrix. These parameters are to be tuned for different n_{max} and b_i values, see Table 4.1.

Hansen and Ostermeier (2001) showed improved solution quality when using *Superlinear* recombination over *Equal* and *Linear*. They also noted that the best setting of this parameter depends on the function being optimized. *Rank-μ* is expected to reduce the number of generations required to reach the optimal solution, as reported by Hansen *et al.* (2003). Their experimental results showed reduction in the time complexity of CMA-ES from quadratic to linear over various functions.

Table 4.1. n_{max} and b_i for which parameters of CMA-ES are to be tuned.

Set[2]	Dimension	n_{max}	b_i
1	5	3000	$\{600, 1200, 1800, 2400\}$
2	5	3000	$\{600, 1200, 1800, 2400, 3000\}$
	10	3000	$\{600, 1200, 1800, 2400, 3000\}$
	15	4500	$\{3000, 3500, 4000, 4500\}$
3	5	1000	$\{400, 600, 800, 1000\}$
	10		
	15		

The meta-level is a basic EA with crossover and mutation operators (detailed shortly). The population size is 15 and the number of generations is 40. Parents are selected according to a tournament selection of size 2, while survival selection is the elitist selection with a constant population size. The initial values for the recombination and update types are randomly selected from their respective domains, while λ, μ, and σ_0 are randomly initialized from [10, 20], [2, λ], and [0.01, 2], respectively. Note that $2 \leq \mu \leq \lambda$ (Hansen, 2006). Each individual is represented as $\langle \lambda, \mu, \sigma_0, recombination\, type, update\, type \rangle$ and is evaluated over k = 6 replications of CMA-ES. Finally, the meta-EA is replicated 30 times.

The crossover and mutation operators work as follows: following parent selection, a random cut-off point c is chosen from the set $\{2,3,4\}$, where c = 2 means that a cut-off occurs after the second "gene", and so on. Each individual is then cut-off at c and the two

[2] *Set*-1 is a mixture of function, *Set*-2 is the hump family of functions, and *Set*-3 is the quadratic family of functions. All will be detailed shortly.

parts are swapped. This ensures that $\mu \leq \lambda$ is maintained. Mutating λ, μ, and σ_0 of the new individuals is done by drawing a number at random from a normal distribution with its mean centered about the current value, and its standard deviation equal to 1. For λ and μ, the new values are rounded to integers. This indicates that, on average, 38% of the time λ and μ remain the same. If σ_0 becomes negative, it is reset to a value of 1.

4.2.2 Training and testing sets

Three sets of optimization functions are used, each using different random seeds for training and testing. *Set*-1 contains eight functions from which different instances cannot be generated; thus, each function is used for training and testing. These functions are: Ackley, Griewangk, Rastrigin, Schwefel, Schwefel ellipsoid, Schwefel ellipsoid rotated, Rosenbrock, and Rosenbrock rotated, all in 5 dimensions and are implemented as in Igel *et al.* (2008). *Set*-2 and *Set*-3 are the hump and quadratic family of functions, from which various instances can be created. These functions are taken from the generator by Rönkkönen *et al.* (2011), and are implemented in 5, 10, and 15 dimensions. See Figures 4.2-4.3.

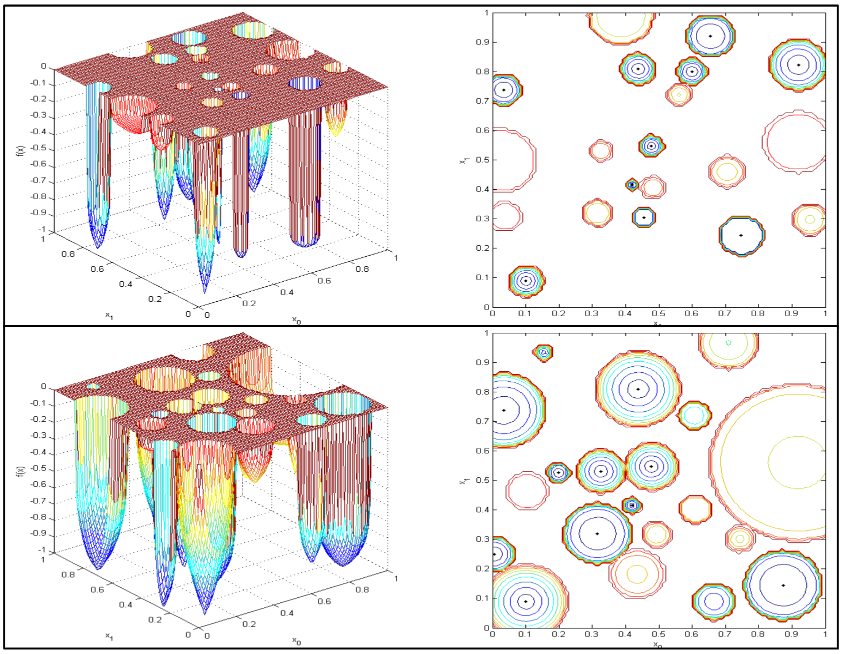


Figure 4.2. Example instances of the hump function in 2D. *Set*-2.

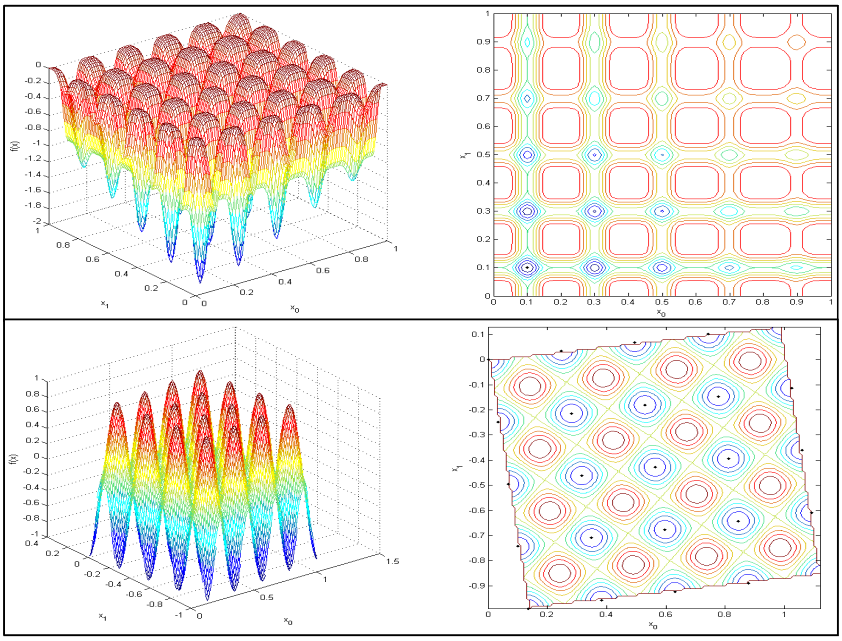


Figure 4.3. Example instances of the quadratic function in 2D. *Set*-3.

4.3 Results

Testing H_0 required using the non-parametric 1-sample sign test, as the data did not follow a normal distribution and was not symmetric. The procedure is designed to test if the population median η is equal to a hypothesized value η_0, in which case one would expect that half the data set will be greater than η_0, against a one-sided or a two-sided alternative. In specific, the hypothesis tested is $H_0: \eta = 0\ vs. H_1: \eta \neq 0$. The sign test reports the number of times the difference in solution quality is below (B), equal (E), or above (A) η_0, along with the p-values. Results are presented separately for each set, followed by a discussion about the best parameter settings each method converged to. Finally, computational savings gained by using the Flexible Budget over the repeated application of the Fixed Budget are shown.

4.3.1 Results for *Set*-1

Only λ and μ were tuned here. Initial results were first presented in Branke and Elomari (2012), and are reproduced in Table 4.2.a. Table 4.2.b. shows the best solution qualities achieved, averaged over all replications. To make the table easier to interpret, column S is added to summarize the results of the sign test. It is read as follows: for the chosen α level of 0.05, a (+) sign indicates that the Flexible Budget method is significantly better, a (≈) sign indicates that there is no significant difference in solution quality between both methods, and a (-) sign indicates that the Fixed Budget method is significantly better.

Overall, these results indicate the superiority of the Flexible Budget method over the repeated application of the Fixed Budget method, especially when the AUC criterion is used. Note, however, that the Flexible Budget method was not designed to improve solution quality, but rather to save computational effort.

Table 4.2.a[*]. Comparison of the solution quality obtained by the Fixed and the Flexible Budget methods at various computational budgets, with α = 0.05. Results are for *Set*-1 functions.

Budget	Function (All in 5D)	Flexible L					Flexible AUC					Flexible AL				
		B	E	A	p-value	S	B	E	A	p-value	S	B	E	A	p-value	S
2400	Ackley	8	0	22	**0.016**	+	1	0	29	**0.000**	+	5	0	25	**0.000**	+
	Griewangk	5	17	8	0.581	≈	0	20	10	**0.002**	+	3	16	11	0.057	≈
	Rastrigin	6	12	12	0.238	≈	3	1	26	**0.000**	+	2	3	25	**0.000**	+
	Schwefel	6	8	16	0.053	≈	5	9	16	**0.027**	+	6	7	17	**0.035**	+
	SchwefelElip	13	11	6	0.167	≈	3	14	13	**0.021**	+	7	12	11	0.481	≈
	SchwefelElipR	14	5	11	0.690	≈	2	15	13	**0.007**	+	13	6	11	0.839	≈
	Rosenbrock	9	2	19	0.087	≈	0	0	30	**0.000**	+	0	0	30	**0.000**	+
	RosenbrockR	8	0	22	**0.016**	+	3	0	27	**0.000**	+	5	0	25	**0.000**	+
1800	Ackley	8	0	22	**0.016**	+	3	0	27	**0.000**	+	6	0	24	**0.001**	+
	Griewangk	7	9	14	0.189	≈	4	10	16	**0.012**	+	9	6	15	0.308	≈
	Rastrigin	7	4	19	**0.029**	+	0	3	27	**0.000**	+	2	3	25	**0.000**	+
	Schwefel	3	6	21	**0.000**	+	3	5	22	**0.000**	+	1	10	19	**0.000**	+
	SchwefelElip	11	9	10	1.000	≈	1	11	18	**0.000**	+	6	8	16	0.053	≈
	SchwefelElipR	10	5	15	0.424	≈	2	10	18	**0.000**	+	9	7	14	0.405	≈
	Rosenbrock	10	11	9	1.000	≈	0	0	30	**0.000**	+	0	0	30	**0.000**	+
	RosenbrockR	10	3	17	0.248	≈	8	0	22	**0.016**	+	7	1	22	**0.008**	+
1200	Ackley	18	0	12	0.362	≈	16	1	13	0.711	≈	19	0	11	0.201	≈
	Griewangk	5	3	22	**0.002**	+	3	1	26	**0.000**	+	4	3	26	**0.000**	+
	Rastrigin	5	8	17	**0.017**	+	4	0	26	**0.000**	+	4	0	26	**0.000**	+
	Schwefel	5	10	15	**0.041**	+	4	11	15	**0.019**	+	6	9	15	0.078	≈
	SchwefelElip	6	9	15	0.078	≈	2	7	21	**0.000**	+	3	7	20	**0.000**	+
	SchwefelElipR	9	6	15	0.307	≈	3	7	20	**0.001**	+	8	6	16	0.152	≈
	Rosenbrock	30	0	0	**0.000**	-	1	2	27	**0.000**	+	5	3	22	**0.002**	+
	RosenbrockR	1	7	22	**0.001**	+	0	5	25	**0.000**	+	1	7	22	**0.000**	+
600	Ackley	13	7	10	0.678	≈	2	17	11	**0.023**	+	5	11	14	0.064	≈
	Griewangk	6	7	17	**0.035**	+	2	10	18	**0.000**	+	3	12	15	**0.008**	+
	Rastrigin	14	2	14	1.000	≈	11	0	19	0.200	≈	12	0	18	0.362	≈
	Schwefel	3	9	18	**0.002**	+	3	8	19	**0.001**	+	5	7	18	**0.012**	+
	SchwefelElip	5	10	15	**0.041**	+	0	11	19	**0.000**	+	5	10	15	**0.041**	+
	SchwefelElipR	8	5	17	0.108	≈	7	4	19	**0.029**	+	9	6	15	0.308	≈
	Rosenbrock	30	0	0	**0.000**	-	4	5	21	**0.001**	+	7	4	19	**0.029**	+
	RosenbrockR	6	2	22	**0.004**	+	2	2	26	**0.000**	+	2	3	25	**0.000**	+
Overall		41% (+), 53% (≈), 6% (-)					94% (+), 6% (≈), 0% (-)					62% (+), 38% (≈), 0% (-)				

[*] Reproduced from Branke and Elomari (2012).

Table 4.2.b. Best solution quality achieved by the Flexible and Fixed Budget methods for *Set*-1. The numbers are averaged over all replications. The optimal value for all functions is 0.0.

Budget	Function (all in 5D)	Flexible-L	Flexible-AUC	Flexible-AL	Fixed
2400	Ackley	0.000	0.000	0.000	0.000
	Griewangk	0.015	0.015	0.025	0.015
	Rastrigin	2.975	1.990	1.990	1.993
	Schwefel	0.001	0.000	0.000	0.000
	SchwefelElip	0.000	0.000	0.000	0.001
	SchwefelElipR	0.000	0.000	0.000	0.001
	Rosenbrock	0.000	0.000	0.000	0.000
	RosenbrockR	0.000	0.000	0.000	0.000
1800	Ackley	0.000	0.000	0.000	0.001
	Griewangk	0.015	0.015	0.025	0.031
	Rastrigin	2.975	1.990	1.990	2.011
	Schwefel	0.001	0.000	0.000	0.001
	SchwefelElip	0.000	0.000	0.000	0.000
	SchwefelElipR	0.000	0.000	0.000	0.000
	Rosenbrock	0.000	0.000	0.000	0.001
	RosenbrockR	0.000	0.000	0.000	0.000
1200	Ackley	0.000	0.000	0.000	0.001
	Griewangk	0.015	0.015	0.025	0.013
	Rastrigin	2.975	2.013	2.013	2.105
	Schwefel	0.001	0.000	0.000	0.000
	SchwefelElip	0.000	0.000	0.000	0.001
	SchwefelElipR	0.000	0.000	0.519	0.014
	Rosenbrock	0.123	0.178	0.123	0.103
	RosenbrockR	0.049	0.049	0.172	0.130
600	Ackley	0.200	0.200	0.200	0.190
	Griewangk	0.337	0.337	0.430	0.389
	Rastrigin	6.580	4.239	4.239	4.583
	Schwefel	0.002	0.000	0.001	0.001
	SchwefelElip	1.017	1.017	1.017	1.017
	SchwefelElipR	0.740	0.897	0.680	0.702
	Rosenbrock	0.981	1.672	0.981	0.791
	RosenbrockR	0.381	0.381	0.410	0.339

Table 4.3 shows the parameter settings suggested by each method, averaged over all 30 replications of the meta-EA. The Flexible Budget proposes smaller μ/λ ratios 50% of the time when using L, 72% of the time when using AL, and 78% of the time when using AUC. See Table 4.3.d. This links to the significantly better results seen in Table 4.2. Clearly, high μ/λ ratios degrade the algorithm's performance under a relatively small budget for these functions.

Table 4.3. The best parameter settings proposed by the Fixed and Flexible Budget methods for *Set*-1.
The numbers are averaged over all 30 replications of the meta-EA.
Table 4.3.a. λ and μ proposed by the Flexible Budget method using **L** as a secondary criterion. Highlighted cells indicate lower μ/λ ratios proposed by the Flexible Budget.

	Flexible L												Fixed											
Function	600			1200			1800			2400			600			1200			1800			2400		
	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ
Ackely	6.10	4.10	0.70	6.10	4.10	0.70	6.10	4.20	0.70	6.10	4.20	0.70	6.20	4.00	0.65	6.70	4.70	0.70	7.30	5.10	0.70	6.80	5.00	0.74
Griewank	5.00	4.50	0.90	5.50	4.50	0.80	7.70	3.90	0.50	8.10	3.10	0.40	7.70	4.40	0.57	8.30	5.60	0.67	8.50	5.70	0.67	8.50	6.10	0.72
Rastrigin	9.30	5.30	0.60	10.10	6.20	0.60	10.20	6.20	0.60	10.30	6.50	0.60	7.90	5.40	0.68	7.60	4.90	0.64	9.10	6.90	0.76	9.50	7.40	0.78
Rosenbrock	9.00	5.00	0.60	9.10	5.00	0.50	7.10	4.30	0.60	7.10	4.30	0.60	7.50	5.40	0.72	7.70	4.00	0.52	7.40	4.10	0.55	7.30	4.30	0.59
RosenbrockR	5.80	2.10	0.40	5.80	2.10	0.40	5.50	2.50	0.50	5.60	2.50	0.40	7.20	4.30	0.60	7.60	4.10	0.54	8.60	4.40	0.51	7.80	4.70	0.60
Schwefel	6.40	4.80	0.80	6.30	4.80	0.80	6.40	5.10	0.80	6.50	5.00	0.80	5.80	2.40	0.41	5.10	2.30	0.45	7.70	3.70	0.48	6.00	4.30	0.72
SchwefelE	6.80	4.20	0.60	6.20	3.80	0.60	7.20	3.60	0.50	6.30	3.40	0.50	5.90	3.20	0.54	6.70	3.10	0.46	6.80	3.30	0.49	5.60	2.60	0.46
SchwefelER	6.50	3.50	0.50	6.40	3.20	0.50	6.00	3.10	0.50	5.80	3.00	0.50	7.50	4.00	0.53	7.00	4.00	0.57	7.20	4.10	0.57	5.60	2.80	0.50

Table 4.3.b. λ and μ proposed by the Flexible Budget method using **AL** as a secondary criterion. Highlighted cells indicate lower μ/λ ratios proposed by the Flexible Budget.

	Flexible AL												Fixed											
Function	600			1200			1800			2400			600			1200			1800			2400		
	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ
Ackely	6.30	4.10	0.65	6.50	4.10	0.63	6.50	4.20	0.65	6.10	4.00	0.66	6.20	4.00	0.65	6.70	4.70	0.70	7.30	5.10	0.70	6.80	5.00	0.74
Griewank	8.30	4.50	0.54	7.30	4.60	0.63	6.70	5.10	0.76	6.50	5.00	0.77	7.70	4.40	0.57	8.30	5.60	0.67	8.50	5.70	0.67	8.50	6.10	0.72
Rastrigin	5.30	2.70	0.51	9.20	6.20	0.67	9.50	6.70	0.71	9.50	6.50	0.68	7.90	5.40	0.68	7.60	4.90	0.64	9.10	6.90	0.76	9.50	7.40	0.78
Rosenbrock	8.00	5.00	0.63	7.40	3.70	0.50	7.50	3.80	0.51	7.30	3.80	0.52	7.50	5.40	0.72	7.70	4.00	0.52	7.40	4.10	0.55	7.30	4.30	0.59
RosenbrockR	7.20	3.40	0.47	7.80	3.40	0.44	7.40	3.00	0.41	6.90	2.70	0.39	7.20	4.30	0.60	7.60	4.10	0.54	8.60	4.40	0.51	7.80	4.70	0.60
Schwefel	6.20	4.90	0.79	6.20	4.90	0.79	6.60	5.60	0.85	6.40	5.30	0.83	5.80	2.40	0.41	5.10	2.30	0.45	7.70	3.70	0.48	6.00	4.30	0.72
SchwefelE	5.80	3.60	0.62	4.60	2.00	0.43	6.00	2.40	0.40	4.60	2.00	0.43	5.90	3.20	0.54	6.70	3.10	0.46	6.80	3.30	0.49	5.60	2.60	0.46
SchwefelER	6.70	3.50	0.52	6.20	3.10	0.50	5.80	3.20	0.55	5.70	2.80	0.49	7.50	4.00	0.53	7.00	4.00	0.57	7.20	4.10	0.57	5.60	2.80	0.50

Table 4.3.c. λ and μ proposed by the Flexible Budget method using **AUC** as a secondary criterion.
Highlighted cells indicate lower μ/λ ratios proposed by the Flexible Budget.

	Flexible AUC												Fixed											
Function	600			1200			1800			2400			600			1200			1800			2400		
	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ	λ	μ	μ/λ
Ackely	6.00	4.10	0.68	6.10	4.10	0.67	6.30	4.10	0.65	6.00	4.10	0.68	6.20	4.00	0.65	6.70	4.70	0.70	7.30	5.10	0.70	6.80	5.00	0.74
Griewank	8.20	3.80	0.46	8.60	3.30	0.38	7.70	4.20	0.55	7.60	4.20	0.55	7.70	4.40	0.57	8.30	5.60	0.67	8.50	5.70	0.67	8.50	6.10	0.72
Rastrigin	5.70	2.70	0.47	8.90	6.40	0.72	8.90	6.60	0.74	8.90	6.40	0.72	7.90	5.40	0.68	7.60	4.90	0.64	9.10	6.90	0.76	9.50	7.40	0.78
Rosenbrock	7.70	4.40	0.57	7.10	3.20	0.45	7.10	3.20	0.45	7.10	3.20	0.45	7.50	5.40	0.72	7.70	4.00	0.52	7.40	4.10	0.55	7.30	4.30	0.59
RosenbrockR	6.20	2.90	0.47	6.10	3.00	0.49	6.10	2.60	0.43	5.70	2.60	0.46	7.20	4.30	0.60	7.60	4.10	0.54	8.60	4.40	0.51	7.80	4.70	0.60
Schwefel	5.80	4.50	0.78	5.90	4.30	0.73	5.90	4.60	0.78	5.80	4.30	0.74	5.80	2.40	0.41	5.10	2.30	0.45	7.70	3.70	0.48	6.00	4.30	0.72
SchwefelE	5.80	3.50	0.60	4.50	1.50	0.33	5.90	2.20	0.37	4.50	2.00	0.44	5.90	3.20	0.54	6.70	3.10	0.46	6.80	3.30	0.49	5.60	2.60	0.46
SchwefelER	6.10	3.10	0.51	5.50	2.50	0.45	5.40	2.40	0.44	5.20	2.20	0.42	7.50	4.00	0.53	7.00	4.00	0.57	7.20	4.10	0.57	5.60	2.80	0.50

Table 4.3.d. Percentage of time the Flexible Budget proposed higher, lower, or equal μ/λ ratios compared to the Fixed Budget method.
The numbers are based on Tables 4.3.a – 4.3.c for all budgets.

	Fixed vs. L	Fixed vs. AL	Fixed vs. AUC
Equal	0.09	0.03	0.00
High	0.41	0.25	0.22
Low	0.50	0.72	0.78

4.3.2 Results for *Set*-2

Different instances can be generated from the hump family of functions. Both methods are first trained on five instances, and then the best parameter settings suggested by each method, at various computational budgets, are tested on four other instances with different random seeds. Training was done by simply averaging the convergence curve of a parameter setting over all five instances, then using the averaged curved in assessing an individual's performance at the meta-level. As with *Set*-1, comparison is based on solution quality using a 1-sample sign test with $\alpha = 0.05$. Results for 5, 10, and 15 dimensions are in Tables 4.4-4.6. The best average solution quality achieved is also reported.

Table 4.4.a. Comparison of the solution quality obtained by the Fixed and Flexible Budget methods at various computational budgets, with α = 0.05.
Results are for the **hump** function in **5D** where training and testing instances are different.

Budget	Instance	Flexible L					Flexible AUC					Flexible AL				
		B	E	A	p-value	S	B	E	A	p-value	S	B	E	A	p-value	S
3000	Hump-1	8	0	22	**0.000**	+	20	0	10	**0.000**	-	5	0	25	**0.000**	+
	Hump-2	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-3	1	0	29	**0.000**	+	6	0	24	**0.000**	+	3	0	27	**0.000**	+
	Hump-4	28	0	2	**0.000**	-	9	0	21	**0.000**	+	17	0	13	**0.000**	-
2400	Hump-1	23	0	7	**0.005**	-	23	0	7	**0.005**	-	25	0	5	**0.000**	-
	Hump-2	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-3	4	0	26	**0.000**	+	24	0	6	**0.001**	-	6	0	24	**0.001**	+
	Hump-4	11	0	19	0.201	≈	9	0	21	**0.043**	+	9	0	21	**0.043**	+
1800	Hump-1	27	0	3	**0.000**	-	26	0	4	**0.000**	-	25	0	5	**0.000**	-
	Hump-2	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-3	18	0	12	0.362	≈	14	0	16	0.856	≈	20	0	10	0.099	≈
	Hump-4	28	0	2	**0.000**	-	1	0	29	**0.000**	+	27	0	3	**0.000**	-
1200	Hump-1	1	0	29	**0.000**	+	1	0	29	**0.000**	+	1	0	29	**0.000**	+
	Hump-2	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-3	1	0	29	**0.000**	+	1	0	29	**0.000**	+	1	0	29	**0.000**	+
	Hump-4	4	0	26	**0.000**	+	4	0	26	**0.000**	+	2	0	28	**0.000**	+
600	Hump-1	2	0	28	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Hump-2	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-3	19	0	11	0.201	≈	17	0	13	0.585	≈	14	0	16	0.856	≈
	Hump-4	10	0	20	0.099	≈	12	0	18	0.362	≈	9	0	21	**0.043**	+
Overall		35% (+), 45% (≈), 20% (-)					40% (+), 40% (≈), 20% (-)					45% (+), 35% (≈), 20% (-)				

Table 4.4.b. Best solution quality achieved by the Flexible and Fixed Budget methods for *Set*-2. The numbers are averaged over all replications. The optimal value for all functions is -1.0. Results are for the **hump** function in **5D** where training and testing instances are different.

Budget	Instance	Flexible-L	Flexible-AUC	Flexible-AL	Fixed
3000	Hump-1	-0.421	-0.356	-0.331	-0.618
	Hump-2	0.000	0.000	0.000	0.000
	Hump-3	-0.578	-0.486	-0.382	-0.837
	Hump-4	-0.499	-0.416	-0.379	-0.787
2400	Hump-1	-0.373	-0.294	-0.225	-0.606
	Hump-2	0.000	0.000	0.000	0.000
	Hump-3	-0.607	-0.450	-0.235	-0.758
	Hump-4	-0.373	-0.234	-0.225	-0.606
1800	Hump-1	-0.201	-0.158	-0.129	-0.420
	Hump-2	0.000	0.000	0.000	0.000
	Hump-3	-0.364	-0.281	-0.181	-0.634
	Hump-4	-0.301	-0.206	-0.181	-0.586
1200	Hump-1	-0.183	-0.156	-0.121	-0.456
	Hump-2	0.000	0.000	0.000	0.000
	Hump-3	-0.277	-0.252	-0.195	-0.750
	Hump-4	-0.272	-0.246	-0.194	-0.750
600	Hump-1	-0.118	-0.079	-0.046	-0.191
	Hump-2	0.000	0.000	0.000	0.000
	Hump-3	-0.251	-0.155	-0.116	-0.404
	Hump-4	-0.251	-0.167	-0.117	-0.405

Table 4.5.a. Comparison of the solution quality obtained by the Fixed the Flexible Budget methods at various computational budgets, with α = 0.05. Results are for the **hump** function in **10D** where training and testing instances are different.

Budget	Instance	Flexible L					Flexible AUC					Flexible AL				
		B	E	A	p-value	S	B	E	A	p-value	S	B	E	A	p-value	S
3000	Hump-1	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-2	7	0	23	**0.005**	+	11	0	19	0.201	≈	10	0	20	0.099	≈
	Hump-3	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-4	7	0	23	**0.005**	+	11	0	19	0.201	≈	10	0	20	0.099	≈
2400	Hump-1	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-2	11	0	19	0.201	≈	9	0	21	**0.043**	+	9	0	21	**0.043**	+
	Hump-3	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-4	11	0	19	0.201	≈	9	0	21	**0.043**	+	9	0	21	**0.043**	+
1800	Hump-1	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-2	23	0	7	**0.005**	-	23	0	7	**0.005**	-	23	0	7	**0.005**	-
	Hump-3	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-4	26	0	4	**0.000**	-	3	0	27	**0.000**	+	3	0	27	**0.000**	+
1200	Hump-1	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-2	4	0	26	**0.000**	+	4	0	26	**0.000**	+	9	0	21	**0.043**	+
	Hump-3	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-4	28	0	2	**0.000**	-	29	0	1	**0.000**	-	23	0	7	**0.005**	-
600	Hump-1	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-2	0	7	23	**0.000**	+	0	4	26	**0.000**	+	0	4	26	**0.000**	+
	Hump-3	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-4	12	0	18	0.362	≈	11	0	19	0.201	≈	9	0	21	**0.043**	+
Overall		20% (+), 65% (≈), 15% (-)					25% (+), 65% (≈), 10% (-)					30% (+), 60% (≈), 10% (-)				

Table 4.5.b. Best solution quality achieved by the Flexible and Fixed Budget methods for *Set*-2.
The numbers are averaged over all replications. The optimal value for all functions is -1.0.
Results are for the **hump** function in **10D** where training and testing instances are different.

Budget	Instance	Flexible-L	Flexible-AUC	Flexible-AL	Fixed
3000	Hump-1	-0.005	-0.005	-0.005	-0.005
	Hump-2	-0.228	-0.212	-0.200	-0.211
	Hump-3	-0.107	-0.107	-0.107	-0.107
	Hump-4	-0.230	-0.213	-0.202	-0.213
2400	Hump-1	0.000	0.000	0.000	0.000
	Hump-2	-0.020	-0.020	-0.018	-0.020
	Hump-3	-0.013	-0.013	-0.013	-0.013
	Hump-4	-0.251	-0.218	-0.198	-0.217
1800	Hump-1	0.000	0.000	0.000	0.000
	Hump-2	-0.017	-0.017	-0.014	-0.016
	Hump-3	-0.009	-0.009	-0.009	-0.009
	Hump-4	-0.247	-0.215	-0.194	-0.214
1200	Hump-1	0.000	0.000	0.000	0.000
	Hump-2	-0.013	-0.012	-0.005	-0.012
	Hump-3	0.000	0.000	0.000	0.000
	Hump-4	-0.295	-0.262	-0.166	-0.304
600	Hump-1	0.000	0.000	0.000	0.000
	Hump-2	-0.002	-0.003	-0.005	-0.002
	Hump-3	0.000	0.000	0.000	0.000
	Hump-4	-0.181	-0.149	-0.204	-0.105

Table 4.6.a. Comparison of the solution quality obtained by the Fixed the Flexible Budget methods at various computational budgets, with α = 0.05.
Results are for the **hump** function in **15D** where training and testing instances are different.

Budget	Instance	Flexible L					Flexible AUC					Flexible AL				
		B	E	A	p-value	S	B	E	A	p-value	S	B	E	A	p-value	S
3000	Hump-1	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-2	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-3	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-4	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
3500	Hump-1	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-2	0	30	0	1.000	≈	0	30	0	1.000	≈	0	23	7	**0.016**	+
	Hump-3	0	30	0	1.000	≈	0	30	0	1.000	≈	0	23	7	**0.016**	+
	Hump-4	0	30	0	1.000	≈	0	30	0	1.000	≈	0	23	7	**0.016**	+
4000	Hump-1	0	30	0	1.000	≈	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-2	0	30	0	1.000	≈	0	30	0	1.000	≈	0	19	11	**0.001**	+
	Hump-3	0	30	0	1.000	≈	0	30	0	1.000	≈	0	19	11	**0.001**	+
	Hump-4	0	30	0	1.000	≈	0	30	0	1.000	≈	0	19	11	**0.001**	+
4500	Hump-1	0	26	4	**0.032**	+	0	30	0	1.000	≈	0	30	0	1.000	≈
	Hump-2	0	30	0	1.000	≈	0	30	0	1.000	≈	3	22	5	**0.041**	+
	Hump-3	0	30	0	1.000	≈	5	15	10	**0.011**	+	0	30	0	1.000	≈
	Hump-4	0	30	0	1.000	≈	0	30	0	1.000	≈	0	29	1	0.703	≈
Overall		6% (+), 94% (≈), 0% (-)					6% (+), 94% (≈), 0% (-)					44% (+), 56% (≈), 0% (-)				

Table 4.6.b. Best solution quality achieved by the Flexible and Fixed Budget methods for *Set*-2. The numbers are averaged over all replications. The optimal value for all functions is -1.0. Results are for the **hump** function in **15D** where training and testing instances are different.

Budget	Instance	Flexible-L	Flexible-AUC	Flexible-AL	Fixed
3000	Hump-1	0.000	0.000	0.000	0.000
	Hump-2	0.000	0.000	0.000	0.000
	Hump-3	0.000	0.000	0.000	0.000
	Hump-4	0.000	0.000	0.000	0.000
3500	Hump-1	0.000	0.000	0.000	0.000
	Hump-2	0.000	0.000	0.000	0.000
	Hump-3	0.000	0.000	0.000	0.000
	Hump-4	0.000	0.000	0.000	0.000
4000	Hump-1	0.000	0.000	-0.001	0.000
	Hump-2	0.000	0.000	-0.001	0.000
	Hump-3	0.000	0.000	-0.002	0.000
	Hump-4	0.000	0.000	-0.002	0.000
4500	Hump-1	0.000	0.000	0.000	0.000
	Hump-2	0.000	0.000	-0.002	0.000
	Hump-3	0.000	-0.001	-0.001	0.000
	Hump-4	0.000	0.000	0.000	0.000

When the test instances are different from the training ones, the Flexible Budget performs just as well as the Fixed Budget most of the time. For the cases where there is a significant difference in performance, the Flexible Budget is still superior more often. The inferior performance observed over 12 instances in Table 4.4 and 7 instances in Table 4.5, reflects degradation in solution quality that is in the order of 10^{-3}, and whether this loss is acceptable, in support of the time savings, depends on the application.

With such performance, one expects both methods to converge to similar parameter settings. This is supported by the results in Tables 4.7-4.9, where the numerical parameters are averaged over all replications, while each categorical parameter is presented as a percentage of the number of times it was selected.

The chosen recombination and update types are very similar for both methods, and follow the same pattern. That is, the recombination type *Equal* is chosen the least, and *Superlinear* is chosen the most, which is consistent with the findings of Hansen and Ostermeier (2001). For 10 and 15 dimensions, update types are selected almost equally as parameter settings are evaluated based on solution quality, not running time, so there should be no advantage for *Rank-μ* over *Rank*-1. Still, for functions in 5 dimensions, *Rank-μ* is obviously preferred all the time. It is unclear why this is so. Numerical parameters are quite similar too, particularly in 10 and 15 dimensions. While the Flexible Budget is proposing lower μ/λ ratios, this does not translate into significantly better results as seen in *Set*-1, probably because the hump function has flat areas where the search stagnates.

Table 4.7. Best parameter settings suggested by the Fixed and Flexible Budget methods. Highlighted cells indicate lower μ/λ ratios proposed by the Flexible Budget. **Hump** function in **5D**.

Budget	Method	λ	μ	σ_0	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	μ/λ
600	Flexible L	18.4	7.1	4.1	13%	27%	60%	30%	70%	0.39
	Flexible AL	17.7	7.5	4.5	0%	47%	53%	23%	77%	0.42
	Flexible AUC	17.3	6.8	3.7	13%	37%	50%	23%	77%	0.39
	Fixed	14.8	7.4	3.8	23%	30%	47%	10%	90%	0.50
1200	Flexible L	18.5	7.5	4.3	7%	20%	73%	23%	77%	0.41
	Flexible AL	17.2	6.7	4.4	7%	47%	47%	30%	70%	0.39
	Flexible AUC	18.0	7.0	4.1	7%	17%	77%	27%	73%	0.39
	Fixed	14.6	6.8	2.6	7%	37%	57%	27%	73%	0.47
1800	Flexible L	17.4	7.2	3.4	3%	23%	73%	33%	67%	0.41
	Flexible AL	17.0	7.1	3.1	0%	27%	73%	33%	67%	0.42
	Flexible AUC	17.5	6.8	2.9	3%	17%	80%	30%	70%	0.39
	Fixed	15.6	6.2	2.0	3%	17%	80%	27%	73%	0.40
2400	Flexible L	17.3	7.0	3.2	0%	13%	87%	20%	80%	0.40
	Flexible AL	17.5	7.0	3.0	0%	23%	77%	30%	70%	0.40
	Flexible AUC	17.6	7.3	3.3	7%	13%	80%	23%	77%	0.41
	Fixed	15.4	5.9	2.3	3%	23%	73%	30%	70%	0.38
3000	Flexible L	17.4	6.9	3.6	0%	13%	87%	27%	73%	0.40
	Flexible AL	17.4	6.7	3.0	0%	23%	77%	33%	67%	0.39
	Flexible AUC	17.9	6.8	3.3	3%	17%	80%	30%	70%	0.37
	Fixed	14.3	5.5	2.3	0%	43%	57%	43%	57%	0.38

Table 4.8. Best parameter settings suggested by the Fixed and Flexible Budget methods. Highlighted cells indicate lower μ/λ ratios proposed by the Flexible Budget. **Hump** function in **10D**.

Budget	Method	λ	μ	σ_0	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	μ/λ
600	Flexible L	16.7	7.5	0.7	17%	37%	47%	37%	63%	0.45
	Flexible AL	17.4	7.1	0.5	10%	40%	50%	47%	53%	0.41
	Flexible AUC	17.4	6.8	0.5	17%	23%	60%	40%	60%	0.39
	Fixed	17.7	9.6	0.7	20%	37%	43%	37%	63%	0.54
1200	Flexible L	16.7	7.2	1.1	10%	27%	63%	53%	47%	0.43
	Flexible AL	17.1	7.2	0.9	17%	37%	47%	50%	50%	0.42
	Flexible AUC	17.0	6.3	0.8	7%	17%	77%	33%	67%	0.37
	Fixed	16.7	8.5	1.3	10%	33%	57%	57%	43%	0.51
1800	Flexible L	16.9	7.0	1.3	13%	33%	53%	50%	50%	0.41
	Flexible AL	17.2	6.6	1.1	13%	47%	40%	47%	53%	0.38
	Flexible AUC	17.1	6.5	1.0	10%	20%	70%	43%	57%	0.38
	Fixed	15.9	7.3	1.3	17%	23%	60%	53%	47%	0.46
2400	Flexible L	16.3	6.8	1.4	10%	33%	57%	47%	53%	0.42
	Flexible AL	17.5	6.1	1.2	17%	37%	47%	43%	57%	0.35
	Flexible AUC	17.0	7.0	0.9	3%	20%	77%	40%	60%	0.41
	Fixed	16.4	7.2	2.2	0%	10%	90%	30%	70%	0.44
3000	Flexible L	16.5	7.5	1.5	7%	33%	60%	50%	50%	0.45
	Flexible AL	17.7	6.7	1.4	13%	37%	50%	43%	57%	0.38
	Flexible AUC	16.3	6.5	1.2	7%	27%	67%	37%	63%	0.40
	Fixed	12.8	5.5	1.8	17%	20%	63%	50%	50%	0.43

Table 4.9. Best parameter settings suggested by the Fixed and Flexible Budget methods. Highlighted cells indicate lower μ/λ ratios proposed by the Flexible Budget. **Hump** function in **15D**.

Budget	Method	λ	μ	σ_0	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	μ/λ
3000	Flexible L	15.8	7.3	2.2	33%	40%	27%	63%	37%	0.46
	Flexible AL	17.5	9.3	1.5	27%	27%	47%	50%	50%	0.53
	Flexible AUC	15.6	8.9	1.7	10%	47%	43%	53%	47%	0.56
	Fixed	14.8	8.5	1.9	27%	27%	47%	47%	53%	0.57
3500	Flexible L	16.3	7.4	2.1	27%	47%	27%	67%	33%	0.45
	Flexible AL	17.0	9.1	2.0	27%	27%	47%	53%	47%	0.54
	Flexible AUC	15.5	8.9	1.6	17%	40%	43%	53%	47%	0.57
	Fixed	15.6	7.3	2.0	20%	33%	47%	73%	27%	0.47
4000	Flexible L	16.5	7.4	2.0	30%	43%	27%	67%	33%	0.45
	Flexible AL	17.1	9.0	1.8	27%	33%	40%	53%	47%	0.53
	Flexible AUC	15.3	9.1	1.5	20%	43%	37%	57%	43%	0.59
	Fixed	16.0	7.3	1.8	23%	30%	47%	47%	53%	0.46
4500	Flexible L	16.6	7.1	1.9	33%	43%	23%	67%	33%	0.43
	Flexible AL	17.2	9.3	1.8	27%	37%	37%	47%	53%	0.54
	Flexible AUC	15.2	8.7	1.5	20%	47%	33%	60%	40%	0.57
	Fixed	16.5	7.3	1.7	33%	37%	30%	67%	33%	0.44

Examining the parameters across dimensions, λ, μ, σ_0 seem to vary less compared to the categorical parameters. See Table 4.10, where the results are limited to the 3000 budget case as it is the only one common between all three dimensions. It is also observed that the μ/λ ratios increase with dimensionality for both methods, their performance becomes more equivalent as dimensionality increases as well. The connection is unclear though, as the change happens for both algorithms.

Table 4.10. Comparison of the best parameters proposed by each method across dimensions for *Set*-2. Results are limited to the 3000 budget case as it is the only common case between all dimensions.

Method	Dimension	λ	μ	σ_0	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	μ/λ
Fixed	5	14.3	5.5	2.3	0.0	0.4	0.6	0.4	0.6	0.38
	10	12.8	5.5	1.8	0.2	0.2	0.6	0.5	0.5	0.43
	15	14.8	8.5	1.9	0.3	0.3	0.5	0.5	0.5	0.57
Flexible L	5	17.4	6.9	3.6	0.0	0.1	0.9	0.3	0.7	0.40
	10	16.5	7.5	1.5	0.1	0.3	0.6	0.5	0.5	0.45
	15	15.8	7.3	2.2	0.3	0.4	0.3	0.6	0.4	0.46
Flexible AL	5	17.4	6.7	3.0	0.0	0.2	0.8	0.3	0.7	0.39
	10	17.7	6.7	1.4	0.1	0.4	0.5	0.4	0.6	0.38
	15	17.5	9.3	1.5	0.3	0.3	0.5	0.5	0.5	0.53
Flexible AUC	5	17.9	6.8	3.3	0.0	0.2	0.8	0.3	0.7	0.38
	10	16.3	6.5	1.2	0.1	0.3	0.7	0.4	0.6	0.40
	15	15.6	8.9	1.7	0.1	0.5	0.4	0.5	0.5	0.57

4.3.3 Results for *Set*-3

The same set of experiments is carried out on the quadratic family of functions, with five training instances and four test instances. Results are summarized in Tables 4.11-4.13.

Table 4.11.a. Comparison of the solution quality obtained by the Fixed Budget method and the Flexible Budget method at various computational budgets, with α = 0.05.
Results are for the **quadratic** function in **5D**.

Budget	Instance	Flexible L					Flexible AUC					Flexible AL				
		B	E	A	*p*-value	S	B	E	A	*p*-value	S	B	E	A	*p*-value	S
400	Quad-1	6	0	24	**0.001**	+	4	0	26	**0.000**	+	10	0	20	0.099	≈
	Quad-2	3	0	27	**0.000**	+	2	0	28	**0.000**	+	2	0	28	**0.000**	+
	Quad-3	6	0	24	**0.001**	+	3	0	27	**0.000**	+	3	0	27	**0.000**	+
	Quad-4	11	0	19	0.201	≈	12	0	18	0.362	≈	10	0	20	0.099	≈
600	Quad-1	18	0	12	0.362	≈	14	0	16	0.856	≈	12	0	18	0.362	≈
	Quad-2	1	0	29	**0.000**	+	1	0	29	**0.000**	+	3	0	27	**0.000**	+
	Quad-3	22	0	8	**0.016**	-	23	0	7	**0.005**	-	19	0	11	0.201	≈
	Quad-4	11	0	19	0.201	≈	5	0	25	**0.000**	+	7	0	23	**0.005**	+
800	Quad-1	0	0	30	**0.000**	+	3	0	27	**0.000**	+	1	0	29	**0.000**	+
	Quad-2	12	0	18	0.362	≈	13	0	17	0.585	≈	15	0	15	1.000	≈
	Quad-3	19	0	11	0.201	≈	17	0	13	0.585	≈	13	0	17	0.585	≈
	Quad-4	19	0	11	0.201	≈	15	0	15	1.000	≈	15	0	15	1.000	≈
1000	Quad-1	26	0	4	**0.000**	-	28	0	2	**0.000**	-	27	0	3	**0.000**	-
	Quad-2	15	0	15	1.000	≈	20	0	10	0.099	≈	20	0	10	0.099	≈
	Quad-3	24	0	6	**0.001**	-	22	0	8	**0.016**	-	26	0	4	**0.000**	-
	Quad-4	9	0	21	**0.043**	+	14	0	16	0.856	≈	7	0	23	**0.005**	+
Overall		38% (+), 43% (≈), 19% (-)					38% (+), 43% (≈), 19% (-)					38% (+), 49% (≈), 13% (-)				

Table 4.11.b. Best solution quality achieved by the Flexible and Fixed Budget methods for *Set*-3.
The numbers are averaged over all replications. The optimal value for all functions is -2.0.
Results are for the **quad** function in **5D** where training and testing instances are different.

Budget	Instance	Flexible-L	Flexible-AUC	Flexible-AL	Fixed
400	Quad-1	-0.111	-0.096	-0.077	-0.039
	Quad-2	-0.113	-0.094	-0.077	-0.038
	Quad-3	-1.344	-1.954	-1.551	-0.775
	Quad-4	-0.112	-0.089	-0.072	-0.036
600	Quad-1	-0.815	-0.808	-0.821	-0.815
	Quad-2	-0.809	-0.814	-0.812	-0.800
	Quad-3	-1.531	-1.457	-1.615	-1.481
	Quad-4	-0.887	-0.898	-0.901	-0.889
800	Quad-1	-0.384	-0.166	-0.406	-0.294
	Quad-2	-0.379	-0.176	-0.404	-0.298
	Quad-3	-0.980	-1.811	-1.474	-1.614
	Quad-4	-0.404	-0.188	-0.447	-0.317
1000	Quad-1	-0.294	-0.300	-0.262	-0.120
	Quad-2	-0.294	-0.299	-0.266	-0.121
	Quad-3	-0.677	-0.802	-1.229	-1.878
	Quad-4	-0.322	-0.321	-0.292	-0.128

Table 4.12.a. Comparison of the solution quality obtained by the Fixed Budget method and the Flexible Budget method at various computational budgets, with α = 0.05.
Results are for the **quadratic** function in **10D**.

Budget	Instance	Flexible L					Flexible AUC					Flexible AL				
		B	E	A	p-value	S	B	E	A	p-value	S	B	E	A	p-value	S
400	Quad-1	3	0	27	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-2	3	0	27	**0.000**	+	1	0	29	**0.000**	+	0	0	30	**0.000**	+
	Quad-3	2	0	28	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-4	3	0	27	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
600	Quad-1	3	0	27	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-2	3	0	27	**0.000**	+	1	0	29	**0.000**	+	0	0	30	**0.000**	+
	Quad-3	2	0	28	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-4	3	0	27	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
800	Quad-1	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-2	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-3	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-4	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
1000	Quad-1	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-2	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-3	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-4	0	0	30	**0.000**	+	0	0	30	**0.000**	+	0	0	30	**0.000**	+
Overall		100% (+), 0% (≈), 0% (-)					100% (+), 0% (≈), 0% (-)					100% (+), 0% (≈), 0% (-)				

Table 4.12.b. Best solution quality achieved by the Flexible and Fixed Budget methods for *Set*-3. The numbers are averaged over all replications. The optimal value for all functions is -2.0.
Results are for the **quad** function in **10D** where training and testing instances are different.

Budget	Instance	Flexible-L	Flexible-AUC	Flexible-AL	Fixed
400	Quad-1	0.437	0.035	0.600	1.460
	Quad-2	0.423	0.071	0.568	1.426
	Quad-3	-1.000	-1.000	-1.000	1.486
	Quad-4	0.480	0.068	0.603	1.437
600	Quad-1	-0.186	-0.334	-0.334	1.300
	Quad-2	-0.128	-0.312	-0.336	1.313
	Quad-3	-1.000	-1.000	-1.000	1.299
	Quad-4	-0.160	-0.306	-0.330	1.280
800	Quad-1	-0.231	-0.347	-0.269	0.662
	Quad-2	-0.235	-0.343	-0.262	0.652
	Quad-3	-1.000	-1.000	-1.000	0.666
	Quad-4	-0.261	-0.355	-0.270	0.656
1000	Quad-1	-0.251	-0.327	-0.209	0.075
	Quad-2	-0.232	-0.337	-0.230	0.083
	Quad-3	-1.000	-1.000	-1.000	0.079
	Quad-4	-0.253	-0.336	-0.232	0.093

Table 4.13.a. Comparison of the solution quality obtained by the Fixed Budget method and the Flexible Budget method at various computational budgets, with α = 0.05.
Results are for the **quadratic** function in **15D**.

Budget	Instance	Flexible L					Flexible AUC					Flexible AL				
		B	E	A	p-value	S	B	E	A	p-value	S	B	E	A	p-value	S
400	Quad-1	3	0	27	**0.000**	+	0	0	30	**0.000**	+	2	0	28	**0.000**	+
	Quad-2	3	0	27	**0.000**	+	0	0	30	**0.000**	+	0	0	30	**0.000**	+
	Quad-3	1	0	29	**0.000**	+	3	0	27	**0.000**	+	4	0	26	**0.000**	+
	Quad-4	1	0	29	**0.000**	+	0	0	30	**0.000**	+	0	0	30	**0.000**	+
600	Quad-1	2	0	28	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-2	3	0	27	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-3	1	0	29	**0.000**	+	0	0	30	**0.000**	+	0	0	30	**0.000**	+
	Quad-4	3	0	27	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
800	Quad-1	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-2	0	0	30	**0.000**	+	0	0	30	**0.000**	+	0	0	30	**0.000**	+
	Quad-3	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-4	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
1000	Quad-1	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-2	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-3	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
	Quad-4	0	0	30	**0.000**	+	0	0	30	**0.000**	+	1	0	29	**0.000**	+
Overall		100% (+), 0% (≈), 0% (-)					100% (+), 0% (≈), 0% (-)					100% (+), 0% (≈), 0% (-)				

Table 4.13.b. Best solution quality achieved by the Flexible and Fixed Budget methods for *Set*-3.
The numbers are averaged over all replications. The optimal value for all functions is -2.0.
Results are for the **quad** function in **15D** where training and testing instances are different.

Budget	Instance	Flexible-L	Flexible-AUC	Flexible-AL	Fixed
400	Quad-1	2.572	1.457	2.983	5.224
	Quad-2	2.598	1.444	2.821	5.182
	Quad-3	2.555	1.445	2.988	5.211
	Quad-4	2.593	1.515	2.871	5.157
600	Quad-1	0.449	0.115	0.001	4.903
	Quad-2	0.493	0.093	-0.019	4.877
	Quad-3	0.473	0.011	-0.056	4.379
	Quad-4	0.437	0.106	-0.046	4.919
800	Quad-1	-0.165	-0.324	-0.079	2.849
	Quad-2	-0.167	-0.286	-0.144	2.896
	Quad-3	-0.166	-0.234	-0.089	2.667
	Quad-4	-0.166	-0.319	-0.093	2.828
1000	Quad-1	-0.197	-0.300	-0.084	0.855
	Quad-2	-0.225	-0.331	-0.095	0.832
	Quad-3	-0.201	-0.336	-0.079	0.819
	Quad-4	-0.221	-0.297	-0.111	0.837

As with the *Set*-2, the Flexible Budget method performs just as well, or better than, the Fixed Budget method in most cases. Some exceptions are observed in Table 4.11 though, with the actual degrade being in the order of 10^{-2}. Smaller μ/λ ratios, Tables 4.14-4.16, did help CMA-ES on the quadratic function given the relatively small budget as seen in *Set*-1. The recombination and update types follow the same pattern observed earlier; that is, both methods select *Superlinear* the most and *Equal* the least, although *Linear* seems to be selected more often here. Also, *Rank-μ* update is preferred in all cases, which is understandable as the budget is relatively small and CMA-ES converges faster with *Rank-μ*.

Table 4.14. Best parameter settings suggested by the Fixed and Flexible Budget methods for various computational budgets. Results are for the **quadratic** family of functions in **5D.**

Budget	Method	*λ*	*μ*	*σ*	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	*μ/λ*
400	Flexible L	18.37	7.07	4.13	13%	27%	60%	30%	70%	0.38
	Flexible AL	17.67	7.53	4.49	0%	47%	53%	23%	77%	0.43
	Flexible AUC	17.33	6.80	3.71	13%	37%	50%	23%	77%	0.39
	Fixed	14.80	7.37	3.85	23%	30%	47%	10%	90%	0.50
600	Flexible L	10.87	5.17	0.25	0%	27%	73%	7%	93%	0.48
	Flexible AL	8.80	4.03	0.36	0%	40%	60%	13%	87%	0.46
	Flexible AUC	10.10	5.17	0.27	3%	30%	67%	7%	93%	0.51
	Fixed	11.93	5.60	0.24	3%	27%	70%	7%	93%	0.47
800	Flexible L	10.37	6.87	0.54	0%	27%	73%	7%	93%	0.66
	Flexible AL	9.23	5.83	0.66	0%	40%	60%	7%	93%	0.63
	Flexible AUC	10.97	7.53	0.54	0%	20%	80%	3%	97%	0.69
	Fixed	11.30	8.27	0.63	0%	30%	70%	0%	100%	0.73
1000	Flexible L	10.70	7.10	0.77	0%	30%	70%	0%	100%	0.66
	Flexible AL	10.13	7.00	0.92	0%	47%	53%	7%	93%	0.69
	Flexible AUC	11.27	7.90	0.74	7%	20%	73%	3%	97%	0.70
	Fixed	12.40	10.10	1.06	0%	27%	73%	0%	100%	0.81

Table 4.15. Best parameter settings suggested by the Fixed and Flexible Budget methods for various computational budgets. Results are for the **quadratic** family of functions in **10D.**

Budget	Method	*λ*	*μ*	*σ*	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	*μ/λ*
400	Flexible L	9.77	5.50	0.55	20%	27%	53%	40%	60%	0.56
	Flexible AL	9.83	4.43	0.48	23%	53%	23%	47%	53%	0.45
	Flexible AUC	9.80	4.40	0.28	17%	47%	37%	23%	77%	0.45
	Fixed	18.93	8.20	0.94	23%	50%	27%	43%	57%	0.43
600	Flexible L	9.43	4.40	0.21	7%	23%	70%	37%	63%	0.47
	Flexible AL	9.40	3.53	0.10	17%	37%	47%	47%	53%	0.38
	Flexible AUC	9.77	3.73	0.08	17%	33%	50%	27%	73%	0.38
	Fixed	18.93	8.20	1.08	30%	53%	17%	30%	70%	0.43
800	Flexible L	9.87	3.97	0.05	13%	10%	77%	37%	63%	0.40
	Flexible AL	9.53	3.37	0.08	17%	37%	47%	47%	53%	0.35
	Flexible AUC	9.73	3.67	0.05	17%	33%	50%	27%	73%	0.38
	Fixed	16.67	6.80	0.55	17%	47%	37%	40%	60%	0.41
1000	Flexible L	9.80	3.97	0.05	13%	10%	77%	40%	60%	0.41
	Flexible AL	9.70	3.43	0.08	17%	37%	47%	43%	57%	0.35
	Flexible AUC	10.00	3.73	0.04	17%	33%	50%	23%	77%	0.37
	Fixed	15.00	5.00	0.29	33%	33%	33%	30%	70%	0.33

Table 4.16. Best parameter settings suggested by the Fixed and Flexible Budget methods for various computational budgets.
Results are for the **quadratic** family of functions in **15D.**

Budget	Method	λ	μ	σ	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	μ/λ
400	Flexible L	9.77	5.60	0.55	20%	27%	53%	40%	60%	0.57
	Flexible AL	9.83	4.53	0.48	33%	34%	33%	43%	57%	0.46
	Flexible AUC	9.80	4.50	0.38	27%	36%	37%	33%	67%	0.46
	Fixed	19.03	8.20	1.04	23%	40%	37%	47%	53%	0.43
600	Flexible L	9.43	4.40	0.21	17%	13%	70%	37%	63%	0.47
	Flexible AL	9.50	3.63	0.20	17%	26%	57%	43%	57%	0.38
	Flexible AUC	9.77	3.83	0.08	27%	13%	60%	27%	73%	0.39
	Fixed	18.93	8.20	1.18	40%	43%	17%	30%	70%	0.43
800	Flexible L	9.97	4.07	0.15	13%	10%	77%	37%	63%	0.41
	Flexible AL	9.63	3.47	0.18	27%	26%	47%	47%	53%	0.36
	Flexible AUC	9.83	3.77	0.05	27%	13%	60%	27%	73%	0.38
	Fixed	16.67	6.90	0.65	17%	36%	47%	50%	50%	0.42
1000	Flexible L	9.80	4.07	0.15	23%	0%	77%	50%	50%	0.42
	Flexible AL	9.70	3.53	0.18	17%	26%	57%	48%	52%	0.36
	Flexible AUC	10.00	3.83	0.04	17%	33%	50%	23%	77%	0.38
	Fixed	15.00	5.00	0.29	43%	14%	43%	30%	70%	0.33

Examining the best parameter settings across dimensions reveals the μ/λ ratios decrease with increasing dimensionality. The only exception is the low budget case where the ratios increase for the Flexible Budget and decrease for the Fixed Budget. These observations do not link directly to performance as the Flexible Budget becomes better with increasing dimensionality, regardless of the available budget. No obvious pattern is observed for categorical parameters. See Tables 4.17-4.20.

Table 4.17. Comparison of the best parameters proposed by each method across dimensions for *Set*-3. Results are for the 400 function evaluations budget.

Method	Dimension	λ	μ	σ_0	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	μ/λ
Fixed	5	14.8	7.4	3.9	0.2	0.3	0.5	0.1	0.9	0.50
	10	18.9	8.2	0.9	0.2	0.5	0.3	0.4	0.6	0.43
	15	19.0	8.2	1.0	0.2	0.4	0.4	0.5	0.5	0.43
Flexible L	5	18.4	7.1	4.1	0.1	0.3	0.6	0.3	0.7	0.39
	10	9.8	5.5	0.6	0.2	0.3	0.5	0.4	0.6	0.56
	15	9.8	5.6	0.6	0.2	0.3	0.5	0.4	0.6	0.57
Flexible AL	5	17.7	7.5	4.5	0.0	0.5	0.5	0.2	0.8	0.42
	10	9.8	4.4	0.5	0.2	0.5	0.2	0.5	0.5	0.45
	15	9.8	4.5	0.5	0.3	0.3	0.3	0.4	0.6	0.46
Flexible AUC	5	17.3	6.8	3.7	0.1	0.4	0.5	0.2	0.8	0.39
	10	9.8	4.4	0.3	0.2	0.5	0.4	0.2	0.8	0.45
	15	9.8	4.5	0.4	0.3	0.4	0.4	0.3	0.7	0.46

Table 4.18. Comparison of the best parameters proposed by each method across dimensions for *Set*-3. Results are for the 600 function evaluations budget.

Method	Dimension	λ	μ	σ_0	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	μ/λ
Fixed	5	11.9	5.6	0.2	0.0	0.3	0.7	0.1	0.9	0.47
	10	18.9	8.2	1.1	0.3	0.5	0.2	0.3	0.7	0.43
	15	18.9	8.2	1.2	0.4	0.4	0.2	0.3	0.7	0.43
Flexible L	5	10.9	5.2	0.3	0.0	0.3	0.7	0.1	0.9	0.48
	10	9.4	4.4	0.2	0.1	0.2	0.7	0.4	0.6	0.47
	15	9.4	4.4	0.2	0.2	0.1	0.7	0.4	0.6	0.47
Flexible AL	5	8.8	4.0	0.4	0.0	0.4	0.6	0.1	0.9	0.45
	10	9.4	3.5	0.1	0.2	0.4	0.5	0.5	0.5	0.37
	15	9.5	3.6	0.2	0.2	0.3	0.6	0.4	0.6	0.38
Flexible AUC	5	10.1	5.2	0.3	0.0	0.3	0.7	0.1	0.9	0.51
	10	9.8	3.7	0.1	0.2	0.3	0.5	0.3	0.7	0.38
	15	9.8	3.8	0.1	0.3	0.1	0.6	0.3	0.7	0.39

Table 4.19. Comparison of the best parameters proposed by each method across dimensions for *Set*-3. Results are for the 800 function evaluations budget.

Method	Dimension	λ	μ	σ_0	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	μ/λ
Fixed	5	11.3	8.3	0.6	0.0	0.3	0.7	0.0	1.0	0.73
	10	16.7	6.8	0.6	0.2	0.5	0.4	0.4	0.6	0.41
	15	16.7	6.9	0.7	0.2	0.4	0.5	0.5	0.5	0.41
Flexible L	5	10.4	6.9	0.5	0.0	0.3	0.7	0.1	0.9	0.66
	10	9.9	4.0	0.1	0.1	0.1	0.8	0.4	0.6	0.40
	15	10.0	4.1	0.2	0.1	0.1	0.8	0.4	0.6	0.41
Flexible AL	5	9.2	5.8	0.7	0.0	0.4	0.6	0.1	0.9	0.63
	10	9.5	3.4	0.1	0.2	0.4	0.5	0.5	0.5	0.36
	15	9.6	3.5	0.2	0.3	0.3	0.5	0.5	0.5	0.36
Flexible AUC	5	11.0	7.5	0.5	0.0	0.2	0.8	0.0	1.0	0.68
	10	9.7	3.7	0.1	0.2	0.3	0.5	0.3	0.7	0.38
	15	9.8	3.8	0.1	0.3	0.1	0.6	0.3	0.7	0.39

Table 4.20. Comparison of the best parameters proposed by each method across dimensions for *Set*-3. Results are for the 1000 function evaluations budget.

Method	Dimension	λ	μ	σ_0	*Equal*	*Linear*	*Superlinear*	*Rank-1*	*Rank-μ*	μ/λ
Fixed	5	12.4	10.1	1.1	0.0	0.3	0.7	0.0	1.0	0.81
	10	15.0	5.0	0.3	0.3	0.3	0.3	0.3	0.7	0.33
	15	15.0	5.0	0.3	0.4	0.1	0.4	0.3	0.7	0.33
Flexible L	5	10.7	7.1	0.8	0.0	0.3	0.7	0.0	1.0	0.66
	10	9.8	4.0	0.1	0.1	0.1	0.8	0.4	0.6	0.41
	15	9.8	4.1	0.2	0.2	0.0	0.8	0.5	0.5	0.42
Flexible AL	5	10.1	7.0	0.9	0.0	0.5	0.5	0.1	0.9	0.69
	10	9.7	3.4	0.1	0.2	0.4	0.5	0.4	0.6	0.35
	15	9.7	3.5	0.2	0.2	0.3	0.6	0.5	0.5	0.36
Flexible AUC	5	11.3	7.9	0.7	0.1	0.2	0.7	0.0	1.0	0.70
	10	10.0	3.7	0.0	0.2	0.3	0.5	0.2	0.8	0.37
	15	10.0	3.8	0.0	0.2	0.3	0.5	0.2	0.8	0.38

4.3.4 Time savings

Computational effort savings are measured by comparing the number of Function Evaluations (FEs) required by the Flexible Budget to those required by the Fixed Budget for all discretization levels b_i's. For instance, in *Set*-1, the Flexible Budget ran for only 2400 FEs, while the Fixed Budget ran for 6000 FEs ($\sum_{i=1}^{4} b_i$). Clearly, savings increase with increasing discretization (the number of times the problem has to be solved with different budgets). For instance, if *Set*-1 ran for another 400 FEs, savings would increase to 67%. See Table 4.21.

Table 4.21. Computational effort savings gained by the Flexible Budget method over the Fixed Budget method.

Problem set	Fixed Budget	Flexible Budget	% Savings
Set-1	6000	2400	60%
Set-2 (5D and 10D)	9000	3000	66%
Set-2 (15D)	15000	4500	70%
Set-3	2800	1000	64%
Suitable for	A fixed number of budgets (4 or 5 here)	Any budget less than n_{max}	

The space complexity of the Flexible Budget method is represented by the additional memory required to store the entire convergence curves. Based on the experiments conducted here, this additional space was minor in today's computing standards. Moreover, the additional storage hardly caused any noticeable slowdowns as it was done on the RAM.

4.3.5 Summary

In summary, the Flexible Budget method was able to find parameter settings for any computational budget, less than n_{max}, without compromising the solution quality of CMA-ES. This saved a lot of computational effort as it was done in a single run, compared to the repeated applications of the Fixed Budget method. Moreover, on some instances, the Flexible Budget method found parameter settings that enabled CMA-ES to find significantly better solutions, compared to those suggested by the Fixed Budget method. On other instances, however, the Flexible Budget method was significantly worse. See Table 4.22.

The computational budget savings increase with the number of times similar instances have to be solved with different budgets. For the experiments conducted here, the savings ranged from 60%-70% of that required by the repeated applications of the Fixed Budget method. See Table 4.23.

Table 4.22. Percentage of times the Flexible Budget method performed significantly better than (+), significantly worse than (-), or no significant difference than (≈) the Fixed Budget method at α = 0.05.

Set	Dimension	L			AUC			AL		
		+	≈	-	+	≈	-	+	≈	-
Set-1	5	41%	**53%**	6%	**94%**	6%	0%	**62%**	38%	0%
Set-2	5	35%	**45%**	20%	40%	**40%**	20%	**45%**	35%	20%
Set-2	10	20%	**65%**	15%	25%	**65%**	10%	30%	**60%**	10%
Set-2	15	6%	**94%**	0%	6%	**94%**	0%	44%	**56%**	0%
Set-3	5	38%	**43%**	19%	38%	**43%**	19%	38%	**49%**	13%
Set-3	10	**100%**	0%	0%	**100%**	0%	0%	**100%**	0%	0%
Set-3	15	**100%**	0%	0%	**100%**	0%	0%	**100%**	0%	0%

Table 4.23. Computational effort savings gained by the Flexible Budget method over the Fixed Budget method.

Problem set	Fixed Budget	Flexible Budget	% Savings
Set-1	6000	2400	60%
Set-2 (5D and 10D)	9000	3000	66%
Set-2 (15D)	15000	4500	70%
Set-3	2800	1000	64%
Suitable for	A fixed number of budgets (4 or 5 here)	Any budget less than n_{max}	

4.4 Conclusion

Two meta-optimizers were experimentally evaluated in this chapter. The Fixed Budget method which tunes parameters for only a predetermined fixed computational budget, and the Flexible Budget method which tunes parameters for *any* computational budget less than a specified maximum in a *single* run. Both meta-optimizers tuned five parameters of the same target algorithm, CMA-ES, three that were numeric: parent population size, offspring population size, and the initial step-size, and the other two are categorical: recombination type and update type. The meta-level algorithm, on the other hand, was an EA with crossover and mutation operators, tournament selection of parents, and an elitist survival selection.

CMA-ES solved three sets of multi-modal optimization functions; *Set*-1 had eight functions, all in five dimensions, which were used for both training (finding the best parameter settings) and testing (running CMA-ES with the proposed parameter settings). *Set*-2 and *Set*-3, on the other hand, had five functions for training and four other functions for testing, all in 5, 10, and 15 dimensions. Following training, CMA-ES was run with the parameter settings proposed by each method, for a pre-determined set of computational budgets. The difference in solution qualities were then compared to see if they significantly differ from zero. The expectation was that both methods can reach the same solution quality, but the Flexible Budget can do so in a single run, hence saving computational effort.

From the experiments conducted in this thesis, the Flexible Budget method found, in most cases, parameter settings which performed just as well, or better than, those suggested by the Fixed Budget method over all three data sets at the various computational budgets selected. Nonetheless, in few cases the Fixed Budget method performed significantly better even though the differences in solution quality did not exceed 10^{-3}-10^{-2}. This exceeded the expectations set for the Flexible Budget as it is designed to save time, not improve solution quality. Time savings ranged from 60%-70% for the experiments conducted here. Refer to Tables 4.22-4.23 for a summary.

As to which secondary criterion to use, results suggest that AL produced significantly better results on most cases, and L produced results with no significant difference on most cases. However, this is limited to the functions, and experimental setup, used here.

Finally, the actual parameter settings each method converged to were analyzed to help explain the observed performance. Overall, it was clear that both meta-optimizers selected roughly the same values for the categorical parameters, while the Fixed Budget method reported higher μ/λ ratios. This forced CMA-ES to explore more solutions under the relatively limited budget used here, and eventually degraded its performance.

CHAPTER 5 Experiments, Results, and Analysis

Computational Budget Allocators

5.1 Introduction

This chapter describes the experiments conducted to validate the second and third contributions: Racing with reset and one-way Racing with intelligent budget allocation. It also assesses their performances against other algorithms currently used in the literature: A-Race, F-Race, and OCBA\CBA. The chapter is organized as follows: Section 5.2 presents the experimental setup (competing algorithms, performance measures, and training and testing sets), followed by the results and analysis in Section 5.3. Finally, Section 5.4 concludes this chapter.

5.2 Experimental setup

5.2.1 Competing algorithms and performance measures

OCBA/CBA is compared to parametric and non-parametric Racing algorithms, with and without the reset idea. Since standard Racing algorithms (without reset) may terminate before the entire budget is consumed, comparison is possible only if they set the budget for the other algorithms. For instance, if, on a particular instance a winner is identified using $n \leq N_{max}$ samples (N_{max} is the maximum allowed budget), then OCBA/CBA is permitted just the n samples for that instance. Racing with reset, on the other hand, can run for any fixed budget. Thus, two sets of experiments are carried out: *Set*-1 runs with a user-specified budget, and *Set*-2 runs with a variable budget specified by Racing. Table 5.1 and Table 5.2 list the competing algorithms for *Set*-1 and *Set*-2 respectively.

Table 5.1. Competing algorithms for *Set*-1 experiments.

Algorithm	Description
A-RaceRR_1WUB	Normal model-based 1-way ANOVA Race with Reset and Resample (unbalanced)
KW-RaceRR	Non-parametric 1-way Kruskal-Wallis ANOVA Race with Reset and Resample (unbalanced)
A-RaceR_2Way	Normal model-based 2-way ANOVA Race with Reset (balanced)
F-RaceR	Non-parametric 2-way Freidman ANOVA Race with Reset (balanced)
OCBA	Optimal Computing Budget Allocation from Simulation Optimization literature
CBA	Correlated Budget Allocation from Simulation Optimization literature
KW-RaceRR_OCBA	Non-parametric 1-way Kruskal-Wallis ANOVA Race with Reset and Resample (unbalanced), with intelligent budget allocation through OCBA
EBA	Equal Budget Allocation

Table 5.2. Competing algorithms for *Set*-2 experiments.
Either *KW*-Race or *A*-Race_2Way set the budget for OCBA, CBA, and EBA.

Algorithm	Description
KW-Race	Non-parametric 1-way Kruskal-Wallis ANOVA Race (balanced)
A-Race_2Way	Normal model-based 2-way ANOVA Race (balanced)
OCBA	Optimal Computing Budget Allocation from Simulation Optimization literature
CBA	Correlated Budget Allocation from Simulation Optimization literature
EBA	Equal Budget Allocation

The performance measures chosen for *Set*-1 are the Probability of Incorrect Selection ($PICS$) and the Expected Opportunity Cost ($\mathbb{E}[OC]$), estimated experimentally over a large number of independent replications. Both measures should decrease as more samples are consumed. The $PICS$ represents the average failure rate of an algorithm, while the $\mathbb{E}[OC]$ represents the average loss endured when selecting an inferior system/parameter setting relative to the true best. The latter measure penalizes bad choices more than slightly bad choices. For instance, one may prefer to be wrong 99% of the time if the penalty for being wrong is only $1 ($\mathbb{E}[OC] = 0.99 * 1 = \$0.99$), compared to being wrong only 1% of the time if the penalty is $1,000 ($\mathbb{E}[OC] = 0.01 * 1000 = \$10$) (Chick and Inoue, 2001).

The same performance measures are used for *Set*-2; however, they are not tracked against the consumed budget, as it varies from one replication to the next. Instead, the minimum average failure rate f_r and $\mathbb{E}[OC]$ achieved at termination are tracked against the significance level α set for Racing. These will be referred to as Efficiency Curves (EC). Both measures are expected to drop as α decrease, up to a certain level. If α becomes very small, the statistical tests of Racing will reject the null hypotheses less often. Hence, systems are rarely discarded, and Racing's performance degrades as it becomes similar to EBA. Since OCBA and CBA do not require a significance level, their f_r and $\mathbb{E}[OC]$ "$at\ a\ certain\ \alpha$" should be interpreted as the performance achieved using the budget set by Racing at that α.

Estimating these measures requires prior knowledge of the true best. This might not be possible when tuning parameters, as it requires the exhaustive search of the $\langle parameter, instance \rangle$ space. Therefore, the performance of each parameter setting, on all possible instances, is simulated with a probability distribution with pre-determined characteristics. Moreover, the distributions are made correlated, with varying degrees, to better capture the performance of real parameters. With this setup, the true best is defined as the distribution with the lowest mean, and the opportunity cost is the difference between the mean of the *selected* best and the mean of the true best.

It is difficult to predict at this stage which algorithm, or set of algorithms, will outperform the rest, especially for the experiments in *Set*-1. This is because Racing with reset overcomes the limitation of terminally discarding the best system. *Set*-2, on the other hand, does seem to give a slight advantage to Racing, by allowing it to set the budget; still, this does not indicate that it should be superior to OCBA/CBA. The results of this chapter will hopefully clarify under which conditions each algorithm performs best, and why.

5.2.2 Training and testing sets

As explained in Chapter 3, using Racing, or OCBA/CBA, as offline tuners entails identifying the best parameter setting from a training set, then applying it to another test set. The working assumption is that the training set is a representative sample, and both sets do not differ much; therefore, the parameter setting which performed best during training should also perform best during testing. As the experiments conducted here rely on probability distributions, one only needs to identify the true best within the available training, or sampling, budget. There is no need for a testing set.

In determining which algorithm performs best, the following factors are varied: first, the number of systems: $k = 10$ and $k = 50$. Second, the probability distribution type: Normal, Gamma, and Weibull. Third, the mean/location and variance/scale parameters: monotonically increasing means with equal variance $\sim\mathcal{D}(i, 6^2)\forall i = 0, \dots, k-1$, monotonically increasing means with exponentially increasing variance $\sim\mathcal{D}(i, (i+1)^4)\forall i = 0, \dots, k-1$, monotonically increasing means with exponentially decreasing variance $\sim\mathcal{D}(i, (k-i)^3)\forall i = 0, \dots, k-1$, and finally the means and variances are drawn at random from a uniform distribution $\sim\mathcal{D}\big(U(0,k), U(10{,}24)\big)$, where $\mathcal{D}$ represents the probability distribution. These means and variances were chosen in specific to be able to benchmark OCBA/CBA's results with the ones published in Chen and Lee (2010) and Fu *et al.* (2007). And fourth, the correlation level ρ between the k systems: 0.0, 0.3, 0.6, 0.9, and mixed where the correlation between systems *i* and *j* is randomly chosen from [0.2,0.6].

For *Set*-1 experiments, the total fixed budget is $N = 2000$ samples, from which $n_0 = 10 * k$ samples are equally distributed among all k systems to initially estimate their performance (i.e. the parameters of the distributions). OCBA and CBA run sequentially, allocating $\Delta = k$ samples in each iteration; thus, they run for a constant number of iterations, 200. Racing with reset algorithms, except *KW*-RaceRR_OCBA, equally allocate $\Delta = number\ of\ survivors$ in each iteration; hence, they may run for any number of iterations, but always consuming the same budget of 2000 samples. For *KW*-RaceRR_OCBA, $\Delta = k$. Moreover, Racing with reset requires an initial significance level, this value is set to $\alpha_0 = 0.1$, and a reduction factor $\gamma = 0.5$. Branke and Elomari (2013) showed that Racing with reset is fairly insensitive to γ. Finally, the $PICS$ and $\mathbb{E}[OC]$ measures are estimated over $r = 100{,}000$ replications.

For *Set*-2, the maximum budget allowed for a standard Racing algorithm is $N_{max} = 1000$ samples, with $n_0 = 10 * k$ samples. Again, Racing may terminate at any budget $n \leq N_{max}$, in which case OCBA and CBA are permitted just the n samples for that specific replication. The performance measures f_r and $\mathbb{E}[OC]$ are tracked on EC plots using a range of fixed α values: $\{0.3, 0.2, 0.15, 0.1, 0.08, 0.06, 0.04, 0.02, 0.01, 0.001, 0.0001, 0.00001\}$. Finally, f_r and $\mathbb{E}[OC]$ are estimated over $r = 10{,}000$ independent replications for each α. Both N_{max} and r were reduced here, compared to *Set*-1, in order to cover such a wide range of α within the available timeframe.

Given the large number of factor-level combinations to vary, *Set*-1 and *Set*-2 experiments are limited to $k = 10$ systems and a single distribution, the Normal. The best algorithm(s) of *Set*-1 is then tested on the remaining distributions, with only 0.0 and 0.9 correlation levels and $k = 50$ systems. All other settings are identical to *Set*-1. These experiments will be designated as *Set*-3.

Finally, the following method was used to create data with pre-determined means, variances, and correlation levels:

- Create a $k \times k$ positive definite covariance matrix based on the correlation.
- Find the Cholesky decomposition of the covariance matrix.
- Draw a $k \times 1$ vector of random numbers from the basic distribution. Normal(0, 1), Weibull(1, 8), or Gamma(5, 6) in these experiments.
- Pre-multiply the decomposed covariance matrix by the $k \times 1$ vector. This will create the required correlation.
- Rescale the $k \times 1$ vector with the required means and variances.

5.3 Results

All competing algorithms will eventually reach a zero $PICS$ and $\mathbb{E}[OC]$ if N is large enough. Nonetheless, they differ in their convergence rate to a target value. Given a fixed N_{max}, not all algorithms can achieve the same target value; hence, each case will be analyzed separately. The analysis will be restricted to the best algorithm, the worst algorithm (except EBA), and two other algorithms performing at intermediate levels.

The results for each set are organized in this fashion: first, the performance outcome is presented, highlighting the major findings at 0.0 and 0.9 correlations. Second, further examination is conducted to try and explain the observed performance. This is also done for 0.0 and 0.9 correlations only. Third, a summary is presented at the end. The reader can skip to the summary section to learn about the key outcomes.

5.3.1 Results for *Set*-1

5.3.1.1 *Case* 1: *Monotonically increasing means with equal variances*

Samples are drawn from $\sim\mathcal{N}(i, 6^2)\forall i = 0, \dots, k-1$. Since the variance is relatively low, and constant, most algorithms are expected to perform well within the available budget of 2000 samples. As seen in Table 5.3, all measures are below 0.05, with the lowest highlighted in bold. The mixed correlation case does not seem to degrade the performance of any algorithm, as the results are close to those obtained for 0.3 and 0.6.

Table 5.3. Lowest *PICS* and *E[OC]* achieved in Case 1 at 2000 samples for various correlation levels.

	Correlation									
Algorithm	0.0		0.3		0.6		0.9		Mixed	
	PICS	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*
EBA	0.040	0.002	0.018	0.000	0.003	0.000	0.000	0.000	0.025	0.001
*A*_RaceR_2Way	0.034	0.001	0.021	0.001	0.012	0.000	0.003	0.000	0.021	0.001
*F*_RaceR	0.031	0.001	0.019	0.000	0.010	0.000	0.001	0.000	0.025	0.001
CBA	0.005	0.000	0.013	0.000	0.003	0.000	0.000	0.000	0.006	0.000
OCBA	0.001	0.000	0.001	0.000	0.001	0.000	0.000	0.000	0.001	0.000
*A*_RaceRR_1WUB	0.001	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000	0.000
*KW*_RaceRR	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
KW-RaceRR_OCBA	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000

From Table 5.3, it is clear that, by the end of the run, OCBA outperforms CBA even under correlation. To clarify why this is so, a different set of experiments was carried out. See Appendix A1 for details. Briefly, the conclusion was that CBA poorly estimates the covariance matrix, leading to degradation in its performance. This issue was never addressed in the original paper of Fu *et al.* (2007). Another finding is that if CBA is provided with the correct covariance matrix, instead of estimating it through sampling, it will outperform OCBA only under high correlation (≥ 0.6). Moreover, its allocation becomes close to EBA's, and at 0.9 both are almost identical. If correlation is low, on the other hand, CBA does not seem to have any performance advantage over OCBA, even when the covariance matrix is provided.

The convergence curves for 0.0 and 0.9 correlation levels are in Figure 5.1, with $PICS$ displayed on a $\log_{10}$ scale. The $\mathbb{E}[OC]$ curves are moved to Appendix A3 as they follow the same pattern as $PICS$. The same hold for the other correlation levels. The curves reveal the ordering of the algorithms during the course of the run.

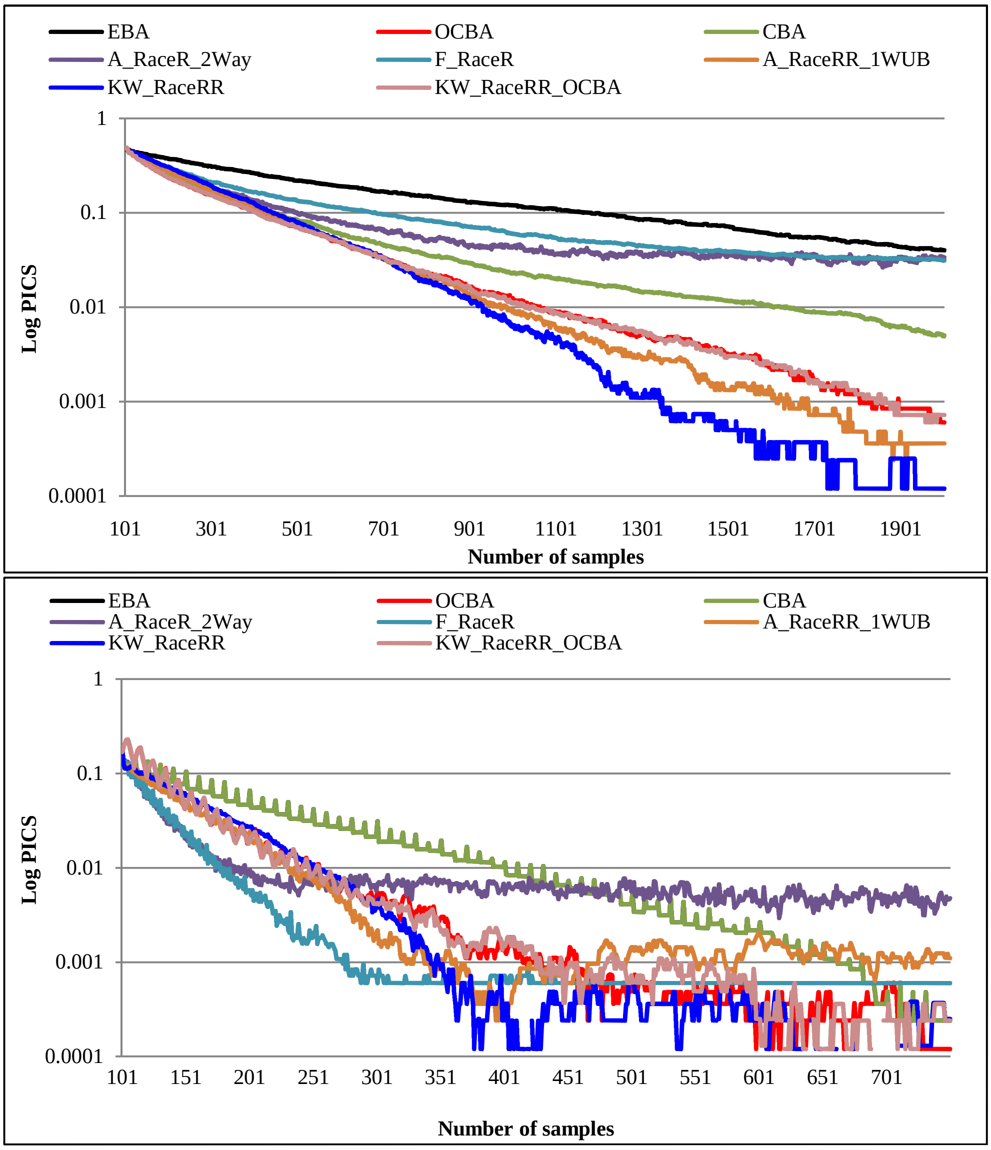


Figure 5.1. Comparison of different algorithms based on *PICS* and *E[OC]* under 0.0 (above) and 0.9 (below) correlation. $\sim\mathcal{N}(i, 6^2)\forall i = 0, \dots, k-1$.

At 0.0 correlation, OCBA is slightly better than its immediate competitor, KW-RaceRR, during the first 500 samples. Afterwards, KW-RaceRR converges at a higher rate and eventually outperforms all other algorithms. KW-RaceRR_OCBA performs close to OCBA, indicating that equally allocating $\Delta = number\ of\ survivors$ works better here, compared to trying to intelligently distribute $\Delta = k$ samples among the surviving systems. Moreover, preliminary experiments showed that setting $\Delta = number\ of\ survivors$ with KW-RaceRR_OCBA caused the algorithm to perform like KW-RaceRR, pointing out that OCBA could not find a better distribution of Δ than the default equal allocation.

At 0.9 correlation, only the first 750 samples are displayed for better viewing. Most algorithms reach nearly a zero $PICS$, and beyond 350 samples, their performance is indistinguishable with only 100,000 replications. As expected, CBA behaves like EBA as indicated by their overlapping curves. Plus, F-RaceR and A-RaceR_2Way are the fastest algorithms to converge, though both slow down fairly quickly and hold at a steady rate.

A final note, if the correlation is high, an oscillating pattern is seen for CBA EBA and OCBA. This is because the performance measures are calculated after each sample; meaning, the $PICS$ will be at minimum at the end of each *iteration*, when all the observations are collected. This is why the oscillations repeat every 10 samples (a single iteration). See Table 5.4. This pattern is clearest when correlation is high, as the observations taken from each distribution are consistent, and is almost none existent if correlation is zero. It also fades away as the budget gets focused on a few systems as in Racing and OCBA.

Table 5.4. An example of why the *PICS* are at minimum near the end of an iteration.
The best mean after each sample is shown in **bold**. The data are for Case 1 at 0.9 correlation.

	System	0	1	2	3	4	Correct Selection?
	Observations of iteration *x*	8.83	8.16	8.55	9.53	11.58	
	Current means	**0.45**	1.87	3.13	3.95	5.12	
Iteration x	Updated mean after 1st sample	4.67	**1.87**	3.13	3.95	5.12	No
	Updated mean after 2nd sample	4.67	5.01	**3.13**	3.95	5.12	No
	Updated mean after 3rd sample	4.67	5.01	5.84	**3.95**	5.12	No
	Updated mean after 4th sample	**4.67**	5.01	5.84	6.74	5.12	Yes
	Updated mean after 5th sample	**4.67**	5.01	5.84	6.74	8.35	Yes

Moving on to the analysis, since this is the equal variances case, an ideal algorithm would shift the budget to the crucial systems, and barely sample the inferior ones beyond n_0. The difference in performance depends on how fast an algorithm makes this shift. The crucial systems are defined as those systems whose parameters require a more accurate estimation, compared to the remaining systems, such that a correct selection is made.

Figure 5.2 displays the overall allocations made per system at 0.0 and 0.9 correlations, averaged over all replications. As anticipated, all algorithms spend most of the budget on the best two systems (0 and 1), and discard the rest rapidly, but at different rates. For the independent case, OCBA and KW-RaceRR shift more of the budget to the best two systems, compared to F-RaceR, and A-RaceR_2Way, leading to superior performance. Under high correlation, however, F-RaceR and A-RaceR_2Way drop off inferior systems almost immediately after n_0 and focus the majority of the budget on the top two, this links to the fast convergence.

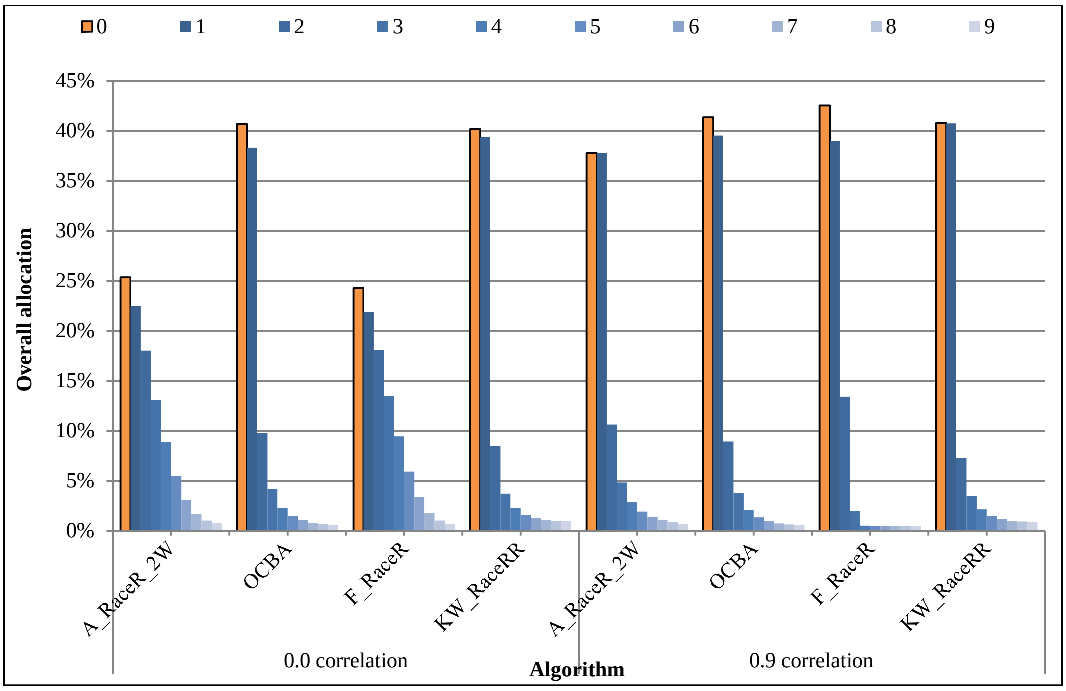


Figure 5.2. Overall allocation of different algorithms at 0.0 and 0.9 correlation. $\sim\mathcal{N}(i, 6^2)\forall i = 0, \dots, k-1$. The numbers are averaged over all replications.

The reason behind the behavior of *A*-RaceR_2Way can be understood by recalling the sum of squares equations (3.17-3.20). With a 2-way normal-model based ANOVA, SS_T was divided into SS_A (parameter setting effect), SS_B (instance effect), SS_{AB} (interaction effect), and SS_E (random error). If no correlation is present, SS_E is likely to be high relative to SS_B and SS_{AB} as there is or no instance effect or interaction effect. Hence, the *F*-ratio for factor *A* will drop and the null hypothesis is rejected less often. This degrades the performance of *A*-RaceR_2Way as it behaves more like EBA. The situation is reversed under high correlation, SS_E gets smaller as SS_B and SS_{AB} become larger, increasing the *F*-ratio for factor *A* and leading to a faster drop out of inferior systems.

A-RaceRR_1WUB, on the other hand, sums SS_B, SS_{AB}, and SS_E in one term: SS_E. This does not have much effect when correlation is zero, because SS_B and SS_{AB} are expected to be low. However, under high correlation, this will degrade performance as the *F*-ratio for factor *A* will drop and the null hypothesis is rejected less often. The reason why *A*-RaceRR_1WUB performs better for the independent case, compared to *A*-RaceR_2Way, is the use of a different reset scheme which utilizes the entire history. Recall Chapter 3.

While *F*-RaceR also relies on a 2-way ANOVA model, it is rank based and does not use sum of squares. Under zero correlation, the samples drawn from each system, *in each iteration*, will have higher variability compared to those drawn under high correlation. Put differently, the ranks of each system will be less consistent across iterations if correlation is zero; hence, it requires *F*-RaceR more samples to discard inferior systems, compared to the high correlation case. Table 5.5 shows the variability of the rank averages assigned to each system over all replications. The *NA* indicates that no ranks were assigned to that system beyond n_0, as it was discarded in all replications.

The same argument holds for *KW*-RaceRR; however, since the observations are first pooled then ranked together, the variability in the ranks should be much higher compared to *F*-RaceR. Such a behavior serves *KW*-RaceRR by forcing it to delay the dropouts, and obtain better results by the end of the run, at the cost of a slower convergence at the beginning. This holds for high and low correlation levels.

Table 5.5. Variability in the average ranks assigned to each system by *F*-RaceR and *KW*-RaceRR, averaged over all replications. The *NA* indicates that no ranks were assigned to that system beyond n_0.

System	*F*-RaceR		*KW*-RaceRR	
	0.0 correlation	0.9 correlation	0.0 correlation	0.9 correlation
0	0.30	0.04	4.89	5.31
1	0.28	0.08	5.95	6.41
2	0.26	0.09	5.89	8.23
3	0.28	0.08	6.74	9.76
4	0.25	0.01	7.59	11.17
5	0.17	*NA*	8.36	12.40
6	0.10	*NA*	9.04	13.58
7	0.09	*NA*	9.82	14.94
8	0.08	*NA*	10.62	16.42
9	0.05	*NA*	11.49	17.84

Going back to OCBA and *KW*-RaceRR, Figure 5.2 shows that *KW*-RaceRR does not allocate a lot more samples to the best two systems compared to OCBA, although it runs for more iterations. This makes it harder to understand why *KW*-RaceRR performs better; therefore, further examination of how the budget is allocated over time was carried out. Figures 5.4 and 5.5 (pages 112-113) show the accumulated percentage of the budget allocated per system as more samples are taken, averaged over all replications, for 0.0 and 0.9 correlation levels respectively. The graph also represents the *survival* length of each system (i.e. how long a system is sampled on average).

With no correlation, Figure 5.4, OCBA and *KW*-RaceRR separate between the systems faster than *F*-RaceR and *A*-RaceR_2Way, which relates to their higher convergence rate. This means that the ranking scheme of *KW*-RaceRR (pooling all observations then ranking them), and the calculated OCBA ratios, better differentiate between the systems compared to *F*-RaceR and *A*-RaceR_2Way. The figure also shows that *KW*-RaceRR, relative to OCBA, allocates more samples to systems 2 and 3, at the expense of systems 0 and 1 (this occurs up to sample 1300). Thus, better estimates of their means make it less likely to mistakenly select any of them as the best. See Figure 5.3. Although this strategy caused *KW*-RaceRR to be slower at the beginning, it achieved higher gains by the end of the run.

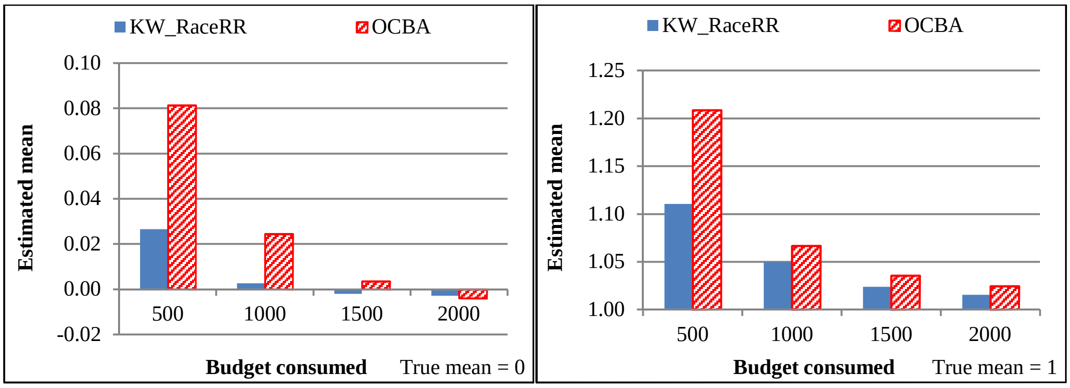


Figure 5.3. Estimated mean of the best two systems under 0.0 correlation at selected points throughout the run. $\sim\mathcal{N}(i, 6^2) \forall i = 0, \dots, k-1$.

The same remarks can be made under high correlation, in Figure 5.4, although one should note that since the samples drawn are more consistent now, the systems closest to the best should be sampled more often, compared to the independent case, so as not to incorrectly select them as the true best. *F*-RaceR and *A*-RaceR_2Way are the fastest algorithms to isolate systems 0 and 1, which relates to their higher convergence rate at the beginning. Also, *A*-RaceR_2Way samples systems 0 and 1 almost equally, as can be seen from their overlapping curves, but it is slower than *F*-RaceR in discarding inferior systems (3-9), which may explain why its convergence rate drops earlier. *KW*-RaceRR is again investing more samples at the beginning in systems 2 and 3, before discarding them. A behavior connected with slower convergence at the start, and superior performance later on.

In conclusion, when the variances are equal, if, at the beginning, the budget allocated to the crucial systems is more evenly spread among them (i.e. more samples are invested in better estimating the parameters of the second and/or third best distributions), lower $PICS$ can be achieved later on. This comes at the expense of a slightly slower convergence rate at the start. For the experiments conducted here, when correlation is 0.0 OCBA is best if the budget is roughly ≤ 500, and *KW*-RaceRR is best if the budget is roughly ≥ 700. With 0.9 correlation, *F*-RaceR is best if the budget is approximately ≤ 350. Beyond that budget, all algorithms reach very low measures and overlap in terms of performance. Still, *KW*-RaceRR seems to have performed better.

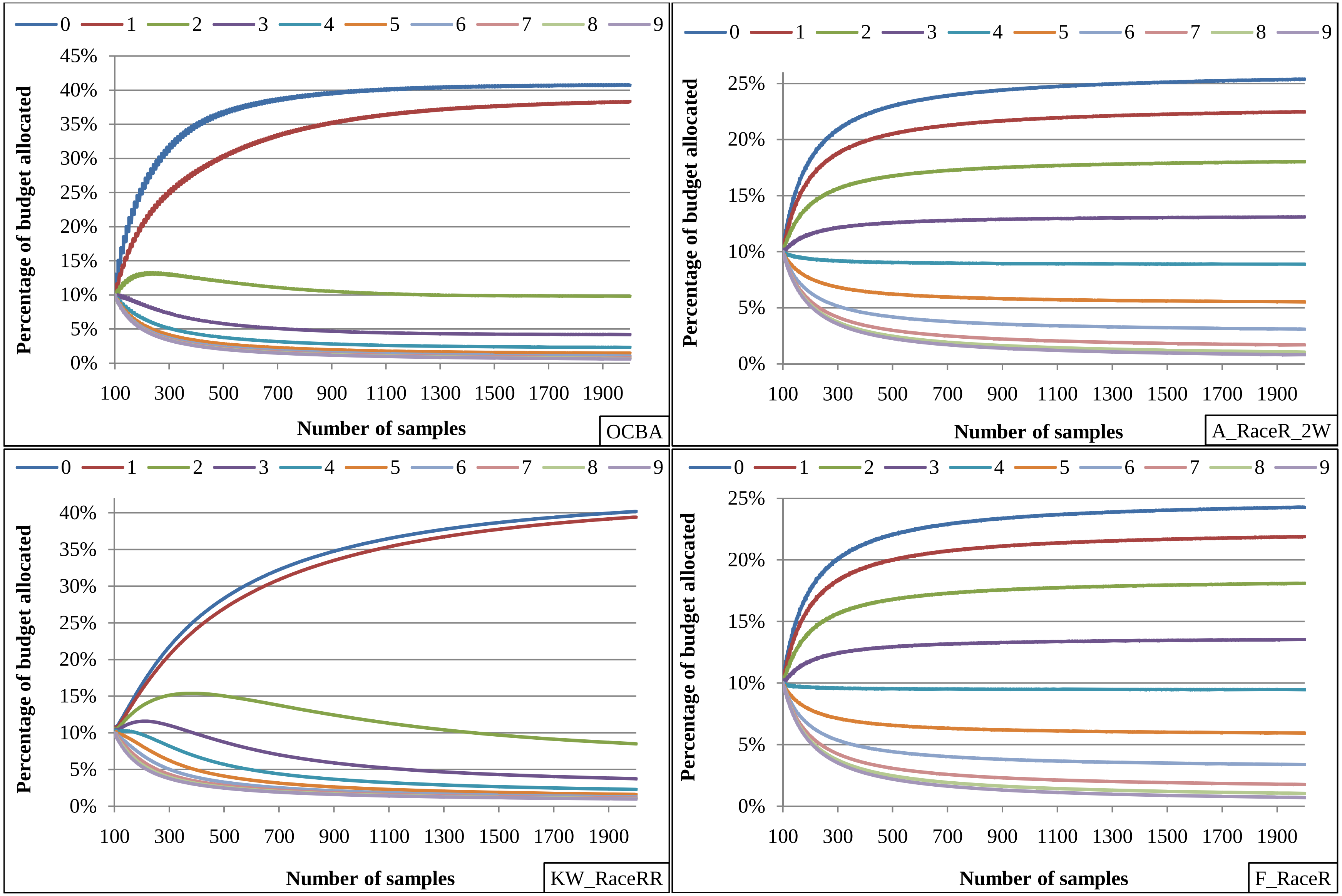


Figure 5.4. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}(i, 6^2)\forall i = 0, \dots, k-1$. Correlation = 0.0.

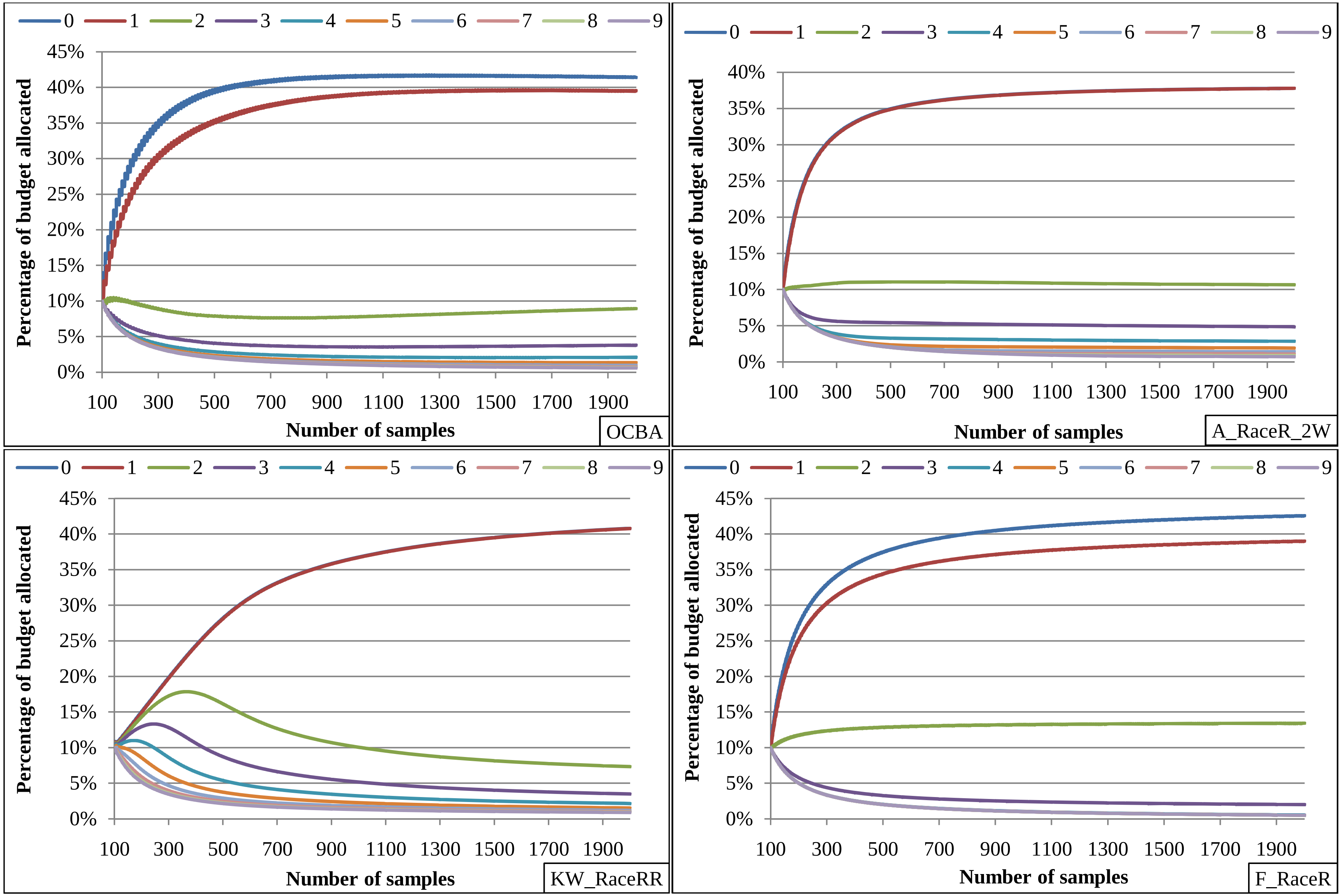


Figure 5.5. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}(i, 6^2)\forall i = 0, \dots, k-1$. Correlation = 0.9.

5.3.1.2 *Case 2: Monotonically increasing means with exponentially decreasing variances*

Data are drawn from $\sim\mathcal{N}(i,(k-i)^3)\forall i=0,\ldots,k-1$. Results in Table 5.6 and Figure 5.6 show that, by the end of the run, OCBA is superior, except at high correlation. This is expected since OCBA\CBA accounts for the ratio of σ_i/σ_j when allocating the budget, Racing algorithms do not. Properly estimating the covariance matrix degrades CBA.

Table 5.6. Lowest *PICS* and *E[OC]* achieved in Case 2 at 2000 samples for various correlation levels.

Correlation	0.0		0.3		0.6		0.9		Mixed	
Algorithm	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*
EBA	0.436	0.257	0.395	0.204	0.323	0.126	0.152	0.025	0.396	0.195
*A*_RaceR_2Way	0.377	0.188	0.312	0.123	0.217	0.057	**0.065**	**0.006**	0.323	0.128
*F*_RaceR	0.469	0.316	0.419	0.243	0.358	0.168	0.181	0.039	0.438	0.258
CBA	0.335	0.143	0.385	0.197	0.325	0.126	0.163	0.029	0.345	0.153
OCBA	**0.288**	**0.101**	**0.251**	**0.075**	**0.203**	**0.050**	0.120	0.019	**0.264**	**0.083**
*A*_RaceRR_1WUB	0.370	0.177	0.314	0.121	0.252	0.063	0.082	0.008	0.370	0.177
*KW*_RaceRR	0.360	0.164	0.314	0.121	0.250	0.062	0.100	0.012	0.327	0.126
*KW*_RaceRR_OCBA	0.316	0.125	0.276	0.093	0.220	0.061	0.134	0.026	0.274	0.100

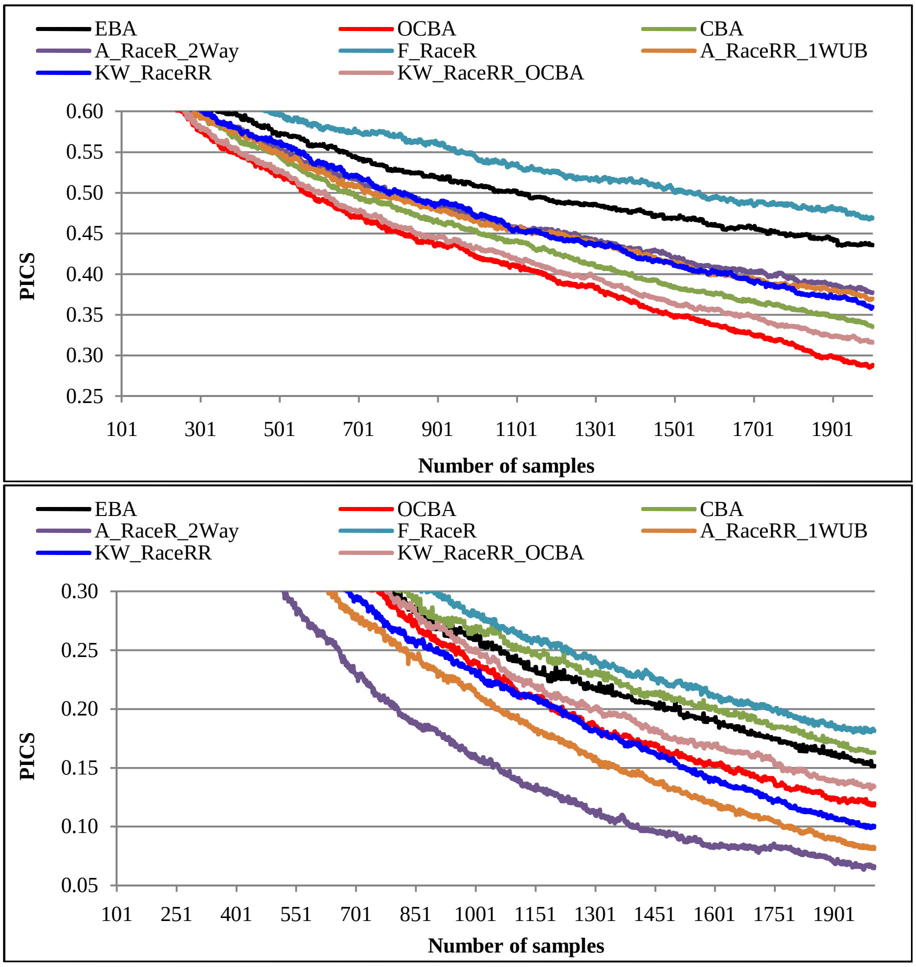


Figure 5.6. Comparison of different algorithms based on *PICS* and *E[OC]* under 0.0 (above) and 0.9 (below) correlation. $\sim\mathcal{N}(i,(k-i)^3)\forall i=0,\ldots,k-1$.

For better viewing, the curves at the beginning of the run are not displayed, especially that their performances are indistinguishable there. Throughout the run, OCBA outperforms all other algorithms for the independent case, and *A*-RaceR_2Way is best if correlation is high. *KW*-RaceRR_OCBA again follows OCBA in terms of performance. As OCBA performs better than *KW*-RaceRR, intelligently allocating Δ is beneficial here.

Moving on to the analysis, since the variances are decreasing exponentially, the means of all the other distributions fall within 1σ. An ideal algorithm is expected to focus most of the budget on the best systems, and almost never sample the inferior ones past n_0.

Racing algorithms will require more samples to start discarding systems, compared to Case 1, since many systems overlap with the best. OCBA/CBA, on the other hand, overcomes this issue by utilizing σ_i/σ_j. Recall Chapter 3. For OCBA, the better systems have higher variances; hence, their n_i/n_j ratios will be higher than that for the inferior systems. This will quickly focus the budget on the best systems at the expense of the inferior ones. The effect of $\delta_{b,j}/\delta_{b,i}$ is minor, because the means are not that far apart, compared to how the variances differ by orders of magnitude. See Figure 5.7.

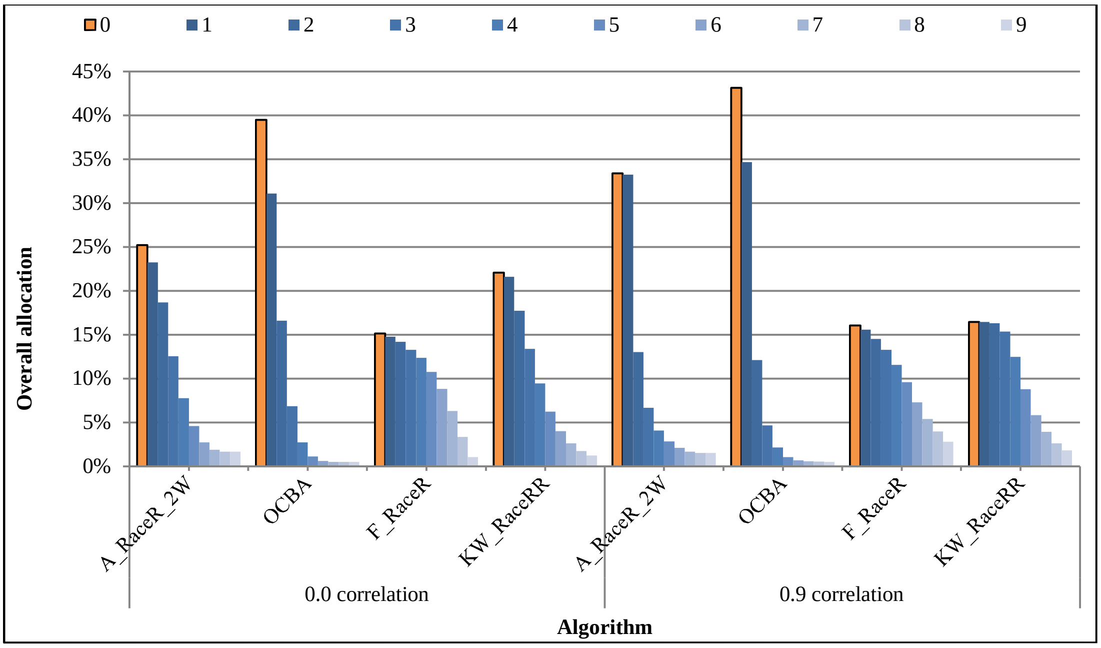


Figure 5.7. Overall allocation of different algorithms at 0.0 and 0.9 correlation. $\sim\mathcal{N}(i,(k-i)^3)\forall i=0,\dots,k-1$. The numbers are averaged over all replications

Moving on to the analysis, at 0.0 correlation, the survival plots (Figure 5.8) show that OCBA is the earliest algorithm to stop sampling system 3 (roughly after 500 samples) and focus the budget on systems 0-2, which can be considered crucial. The remaining systems are nearly never sampled past n_0. Racing algorithms are following the same patter, but are slower in making the shift. This allowed OCBA to obtain more accurate estimates of the distribution means (of systems 0-2), compared to the others. See Table 5.7.

Table 5.7. Deviation of the estimated means from their nominal values.
The numbers reported for $\sim\mathcal{N}(i,(k-i)^3)\forall i=0,\ldots,k-1$ with 0.0 correlation.

System	Nominal mean	*A*_RaceR_2Way	*F*_RaceR	*KW*_RaceRR	OCBA
0	0	1.79	2.52	1.45	1.25
1	1	1.64	2.20	1.18	1.08
2	2	1.61	1.81	1.22	1.21
3	3	1.41	1.58	1.19	2.40
4	4	1.02	1.19	0.92	1.80
5	5	0.64	0.94	0.71	1.07
6	6	0.28	0.64	0.43	0.35
7	7	0.07	0.32	0.21	0.03
8	8	0.02	0.09	0.08	-0.01
9	9	0.00	0.00	0.00	0.00

It is worth noting how the ranking scheme of *KW*-RaceRR and *F*-RaceR is forcing these algorithms to retain some non-crucial systems, by assigning almost equal ranks to most systems. This, in turn, requires allocating more samples to them at the expense of system 0. Table 5.8 shows how *KW*-RaceRR is better in distinguishing between the systems, compared to *F*-RaceR, as observed by the location of the largest gap in rank averages.

Table 5.8. Averages and variances of the ranks assigned to all systems by *F*-RaceR and *KW*-RaceRR.
The numbers are based on all replications for $\sim\mathcal{N}(i,(k-i)^3)\forall i=0,\ldots,k-1$ with 0.0 correlation.
The numbers in **bold** represent the location of the largest gap in rank averages.

System	*F*-RaceR				*KW*-RaceRR			
	0.0 correlation		0.9 correlation		0.0 correlation		0.9 correlation	
	Av.	Var.	Av.	Var.	Av.	Var.	Av.	Var.
0	5.50	0.02	6.10	0.04	351.99	334.24	402.99	407.92
1	5.53	0.04	6.20	0.06	361.75	363.48	409.07	423.21
2	5.56	0.06	6.30	0.08	349.64	318.54	416.40	440.99
3	5.59	0.07	6.35	0.11	329.68	260.12	421.23	445.48
4	5.63	0.09	6.27	0.13	310.04	205.49	406.78	379.56
5	5.61	0.10	**6.07**	0.15	**280.38**	143.65	**362.06**	243.19
6	5.57	0.12	**5.69**	0.15	**250.95**	95.86	**306.17**	132.67
7	**5.41**	0.12	5.38	0.15	221.94	64.89	257.06	70.54
8	**5.08**	0.12	5.27	0.16	194.71	47.48	207.83	33.88
9	4.85	0.11	5.20	0.15	170.81	39.24	169.40	15.81

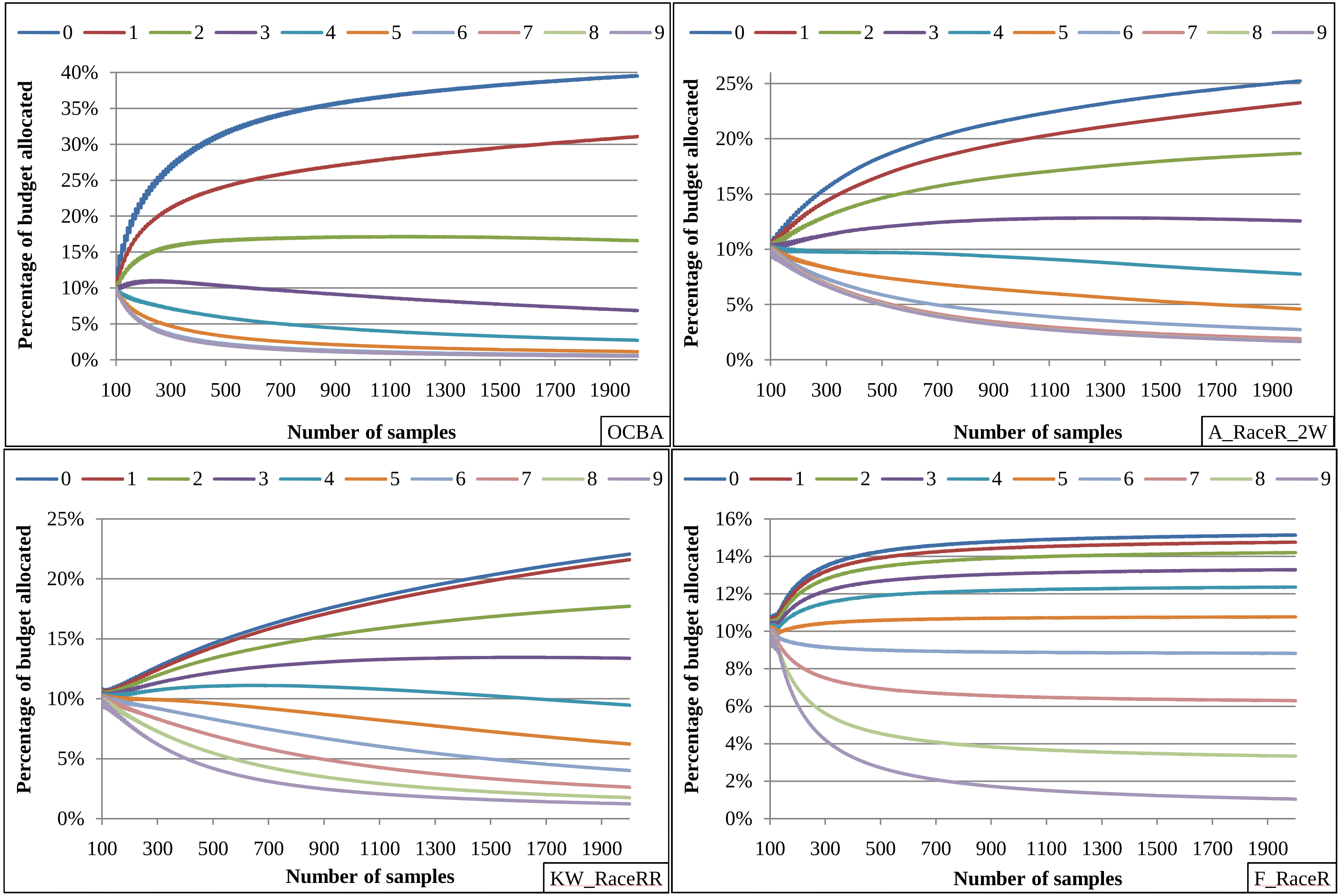


Figure 5.8. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}(i,(k-i)^3)\forall i = 0,\dots,k-1$. Correlation = 0.0.

At 0.9 correlation, the samples taken from each system, in every iteration, are more consistent. This means that the systems closest to the best, probably the second and third best, should be allocated more of the budget, compared to the 0.0 correlation case, so as not to be mistakenly selected as the best. This was observed earlier in Case 1, and again in Figure 5.9 for all Racing algorithms and, to a less extent, OCBA. The survival plots make it clear why F-RaceR performs the worst. It, again, holds on to many non-crucial systems due to the high variance, and its ranking scheme discussed before. KW-RaceRR is faster in discarding systems 6-9 and in shifting their budget to systems 0-3, which allows it to perform better than F-RaceR. A-RaceR_2Way follows the same pattern as KW-RaceRR but to a higher extent, enabling it to allocate more of the budget to the best two systems and discard the inferior ones almost immediately after n_0. Also, it isolates the correlation effect.

The survival plot of OCBA is a bit peculiar, as it is very comparable to that of A-RaceR_2Way, but it translates to a worse performance. A closer look will show that while OCBA is allocating more of the budget to system 0, compared to A-RaceR_2Way, it is not doing so at the expense of system 1, but rather at the expense of the inferior systems. See Figure 5.10. This indicates that for Case 2 with high correlation, the worst systems should not be neglected quickly, as OCBA does, nor should they be allocated nearly as many samples as the best systems have, as F-RaceR does, some middle ground is required and A-RaceR_2Way makes the best balance.

Finally, the performance of F-RaceR is worse than EBA, even though the top 6 systems get more than N_{max}/k samples on average. To understand why, the estimated mean of each system is calculated under two conditions: if it is sampled more than N_{max}/k, on average, and if it is sampled less. Table 5.9 indicates that all systems are sampled more than N_{max}/k, as expected. But, when they are not, their means are poorly estimated, which affects the overall estimates. This explains why F-RaceR converges slower than EBA.

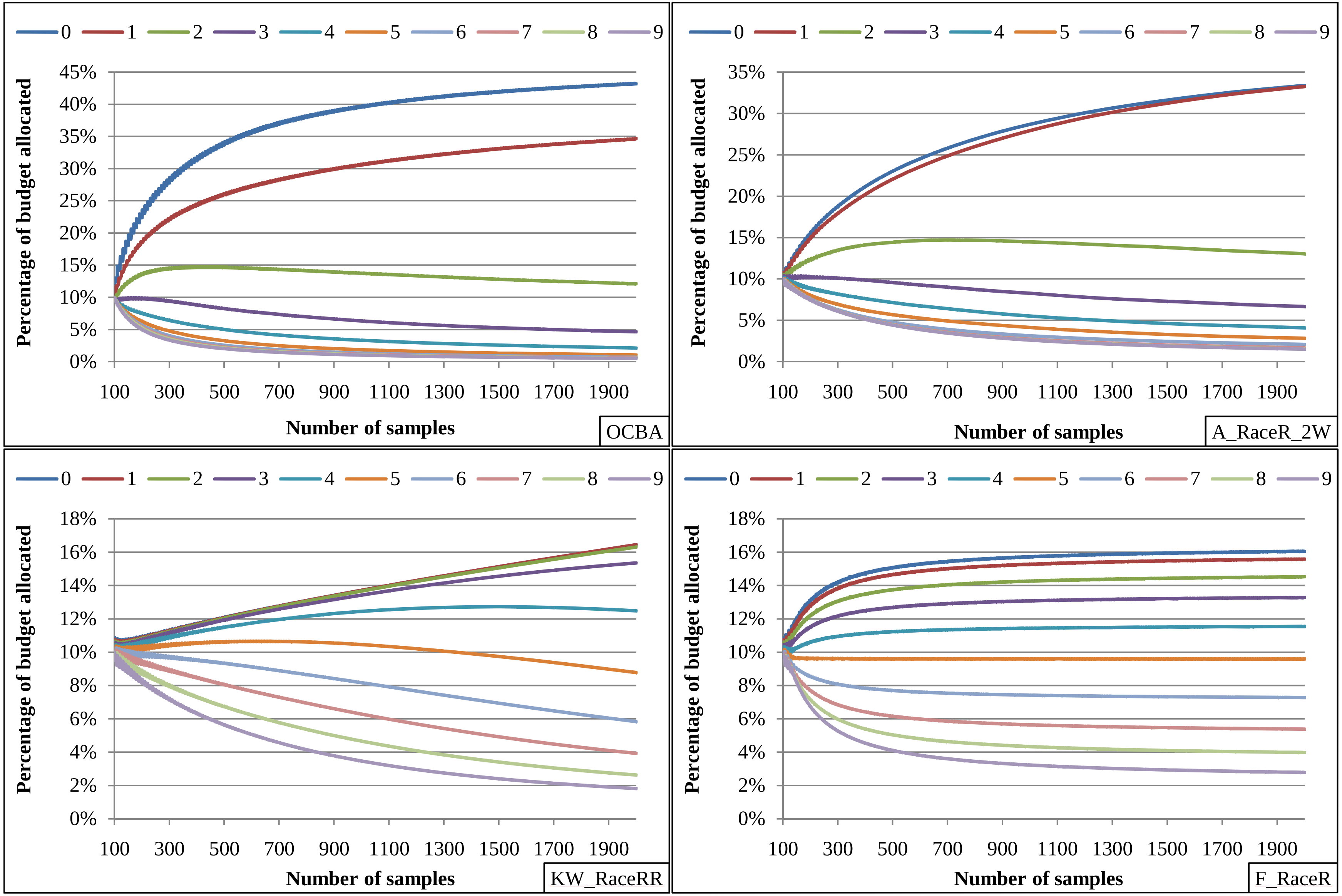


Figure 5.9. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}(i,(k-i)^3)\forall i = 0,\dots,k-1$. Correlation = 0.9.

Table 5.9. Estimated mean of each system if it is sampled less, or more, than N_{max}/k, when F-RaceR allocates the budget under 0.0 correlation.

	Sampled less than N_{max}/k			Sampled more than N_{max}/k			Means estimated with EBA
Nominal value	Estimated mean	Frequency	Av. # of samples	Estimated mean	Frequency	Av. # of samples	
0	10.27	16.16%	12.44	0.00	83.84%	970.61	0.01
1	8.06	17.56%	12.25	1.01	82.44%	949.58	1.01
2	5.92	19.45%	11.95	2.01	80.55%	912.39	2.00
3	4.16	23.78%	11.63	3.00	76.22%	866.29	3.00
4	3.37	29.39%	11.32	4.01	70.61%	813.24	4.00
5	3.76	38.82%	11.18	5.02	61.18%	760.91	5.00
6	4.82	49.91%	11.11	6.02	50.09%	719.86	6.00
7	6.24	61.92%	11.23	7.02	38.08%	680.61	7.00
8	7.63	71.59%	11.41	8.01	28.41%	657.45	8.00
9	8.90	81.39%	11.66	9.01	18.61%	658.60	9.00

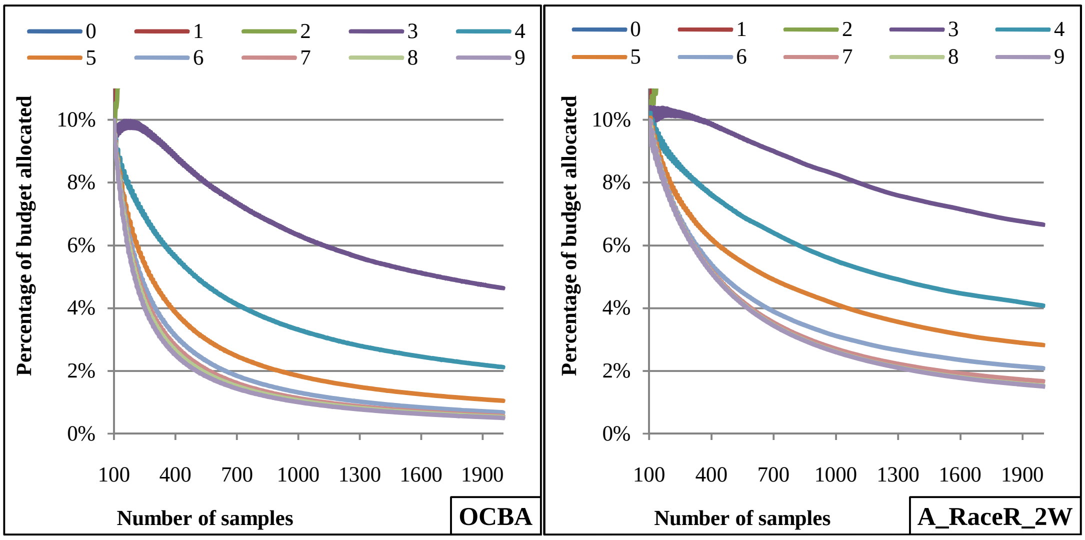


Figure 5.10. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}(i,(k-i)^3)\forall i = 0, \dots, k-1$. Correlation = 0.9.
Only the systems getting less than n_0 are displayed.

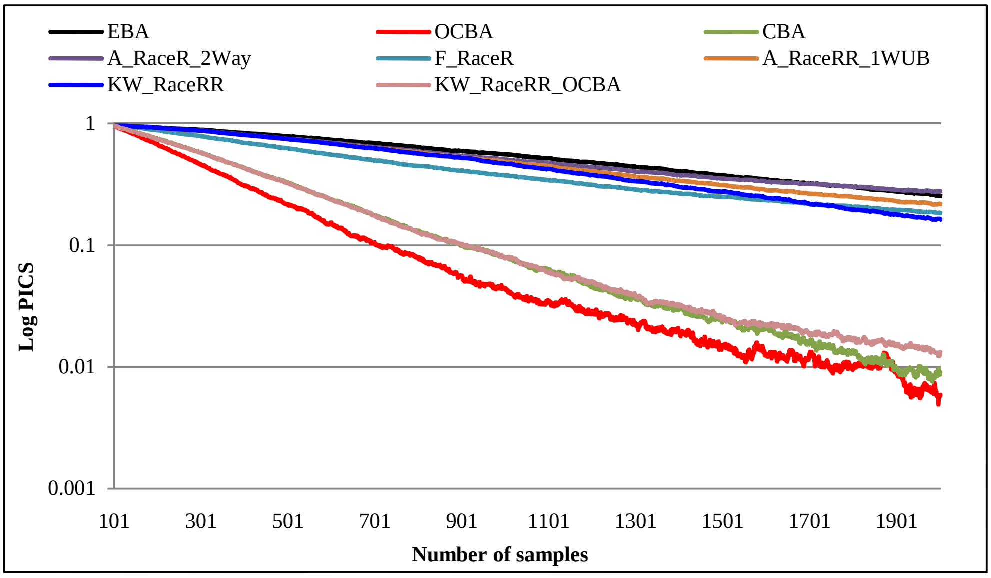
In conclusion, with no correlation, the fastest algorithm to neglect the inferior systems, and focus the budget on the best ones, achieves lower PICS. OCBA is able to make this separation due to accounting for the ratio of the variances. Racing algorithms follow this same pattern, but with varying degrees; hence, they cannot perform just as well. At high correlation, the systems closest to the best should be sampled more often, compared to the independent case, while inferior systems are not to be neglected completely. A-RaceR_2Way achieves such a balance and isolates the correlation effect, as it did in Case 1, enabling it to outperform all other algorithms.

5.3.1.3 *Case* 3: *Monotonically increasing means with exponentially increasing variances*

Data are drawn from $\sim\mathcal{N}(i,(i+1)^4)\forall i=0,\dots,k-1$. Results in Table 5.10 and Figure 5.11 agree with what was observed in Case 2; OCBA outperforms all other algorithms due to utilizing the ratio of the variances. CBA is second in line, but never reaching better results even under correlation. *KW*-RaceRR_OCBA here too benefits from intelligently allocating Δ in every iteration, realizing performance close to that of OCBA's. All Racing algorithms perform very closely to each other with *KW*-RaceRR being slightly better at 0.0 correlation by the end of the run, and *F*-RaceR being better up to approximately sample 1500. At 0.9 correlation, EBA outperforms all Racing algorithms.

Table 5.10. Lowest *PICS* and *E[OC]* achieved in Case 3 at 2000 samples for various correlation levels.

Algorithm	Correlation									
	0.0		0.3		0.6		0.9		Mixed	
	PICS	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*
EBA	0.255	0.489	0.216	0.359	0.168	0.224	0.116	0.114	0.196	0.301
*A*_RaceR_2Way	0.278	0.535	0.257	0.477	0.238	0.424	0.226	0.384	0.021	0.001
*F*_RaceR	0.184	0.201	0.223	0.278	0.244	0.331	0.255	0.357	0.231	0.294
CBA	0.009	0.000	0.012	0.001	0.023	0.004	0.028	0.006	0.015	0.002
OCBA	**0.006**	**0.000**	**0.011**	**0.001**	**0.014**	**0.001**	**0.019**	**0.002**	**0.012**	**0.001**
*A*_RaceRR_1WUB	0.218	0.332	0.221	0.361	0.191	0.282	0.159	0.199	0.218	0.332
*KW*_RaceRR	0.163	0.193	0.183	0.244	0.177	0.224	0.172	0.193	0.179	0.231
*KW*_RaceRR_OCBA	0.013	0.001	0.021	0.002	0.029	0.003	0.049	0.009	0.097	0.120



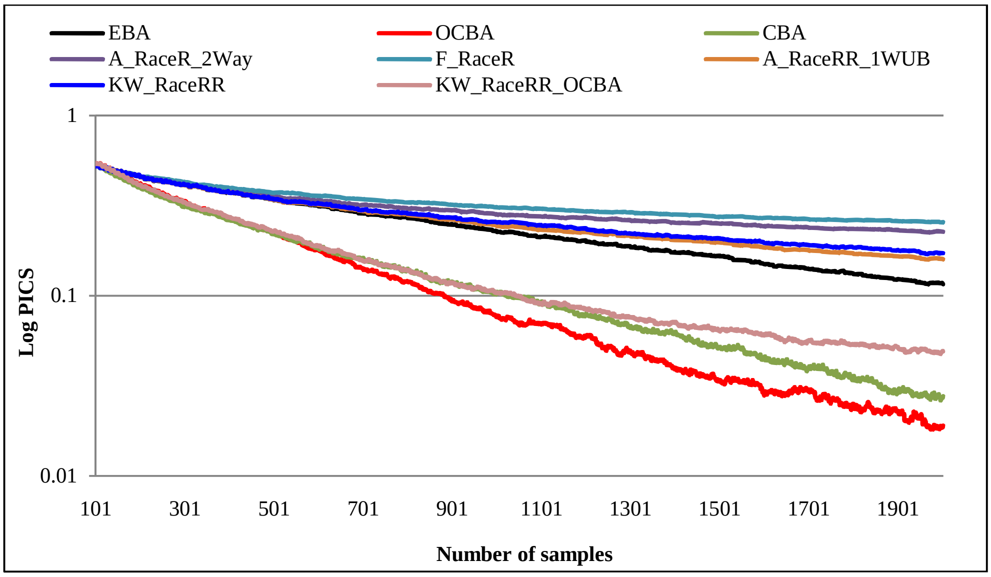


Figure 5.11. Comparison of different algorithms based on *PICS* and *E[OC]* under 0.0 (above) and 0.9 (below) correlation. $\sim\mathcal{N}(i,(i+1)^4)\forall i = 0,\ldots,k-1$.

Moving on to the analysis, as the variances are exponentially increasing, the best system has the lowest variance. It should, therefore, be easy to isolate it from the rest. Additional budget is required for the inferior systems so as not to incorrectly select them as the best. OCBA/CBA follows this pattern due to exploiting σ_i/σ_j, Racing algorithms, however, do not. Once Racing identifies the best, inferior systems will be discarded, shifting the remainder of the budget to surviving systems. It is therefore expected to find Racing distribute more samples to the best systems. See Figures 5.12-5.14.

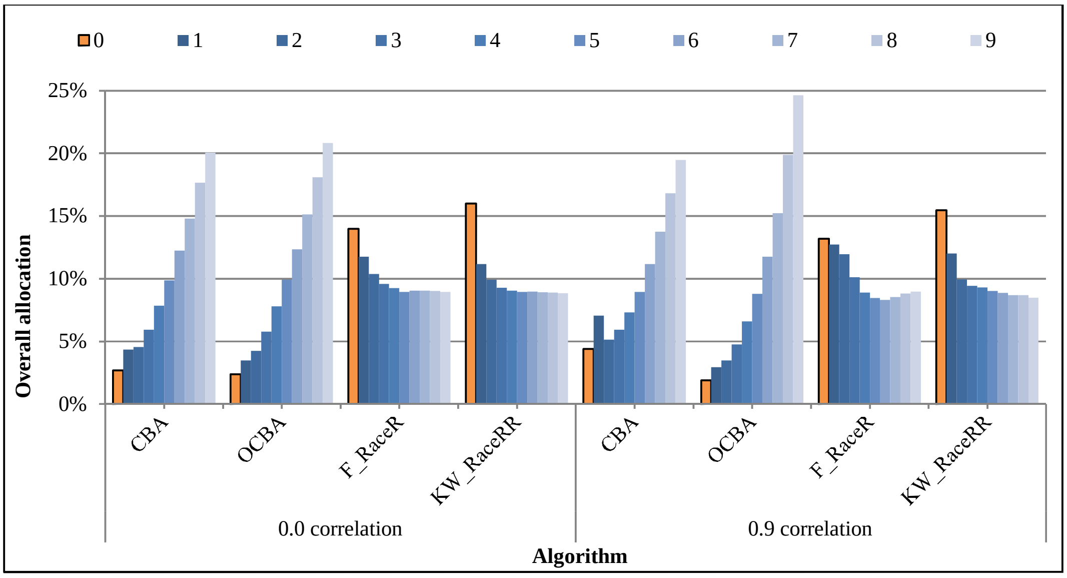


Figure 5.12. Overall allocation of different algorithms at 0.0 and 0.9 correlation.

$\sim\mathcal{N}(i,(i+1)^4)\forall i = 0,\ldots,k-1$. The numbers are averaged over all replications

The survival plots reflect the expected allocations. They also explain the difference in performance between F-RaceR and KW-RaceRR. For 0.0 correlation, F-RaceR is faster in separating between the systems leading to improved performance approximately up to sample 1700. From that point on, its performance overlaps with KW-RaceRR, which performs somewhat better at the end, but it is unclear why. At 0.9 correlation, as seen in Case 2 previously, the high variance requires the inferior systems to be sampled more often, compared to low correlation. It is clear that F-RaceR discards systems 5-9 sooner than KW-RaceRR, which degrades its performance during the entire run.

Finally, the performance of all Racing algorithms is worse than EBA at high correlation. Yet, it cannot be concluded that EBA is better in the limit, as Racing algorithms were able to separate between the systems by the end of the run. And, if given a higher budget, should outperform EBA. To test this, EBA and KW-RaceRR are run with a budget of 10,000 samples instead of 2000. See Figure 5.15 (page 126). Observe how, by the end of the run, the allocations of KW-RaceRR resemble those of OCBA\CBA. However, the convergence curves indicate that by the time KW-RaceRR reaches this “optimal” allocation, EBA will have reached almost a zero $PICS$, and at that point it will not make any difference if KW-RaceRR is allocating the budget optimally or not.

In conclusion, OCBA\CBA benefits from utilizing σ_i/σ_j and outperforms all other algorithms, all throughout the run. KW-RaceRR_OCBA also makes the same benefit and performs better than KW-RaceRR. All other Racing algorithms focus the budget on the best system, as it has the lowest variance, but poorly distribute the budget among the others (especially the inferior ones), due to their high variance. This leads to sampling them almost equally within the available budget. Further investigation showed that if KW-RaceRR is allowed a higher budget, its allocation will resemble more that of OCBA\CBA, but it requires many samples to do so, at which point EBA is a better option.

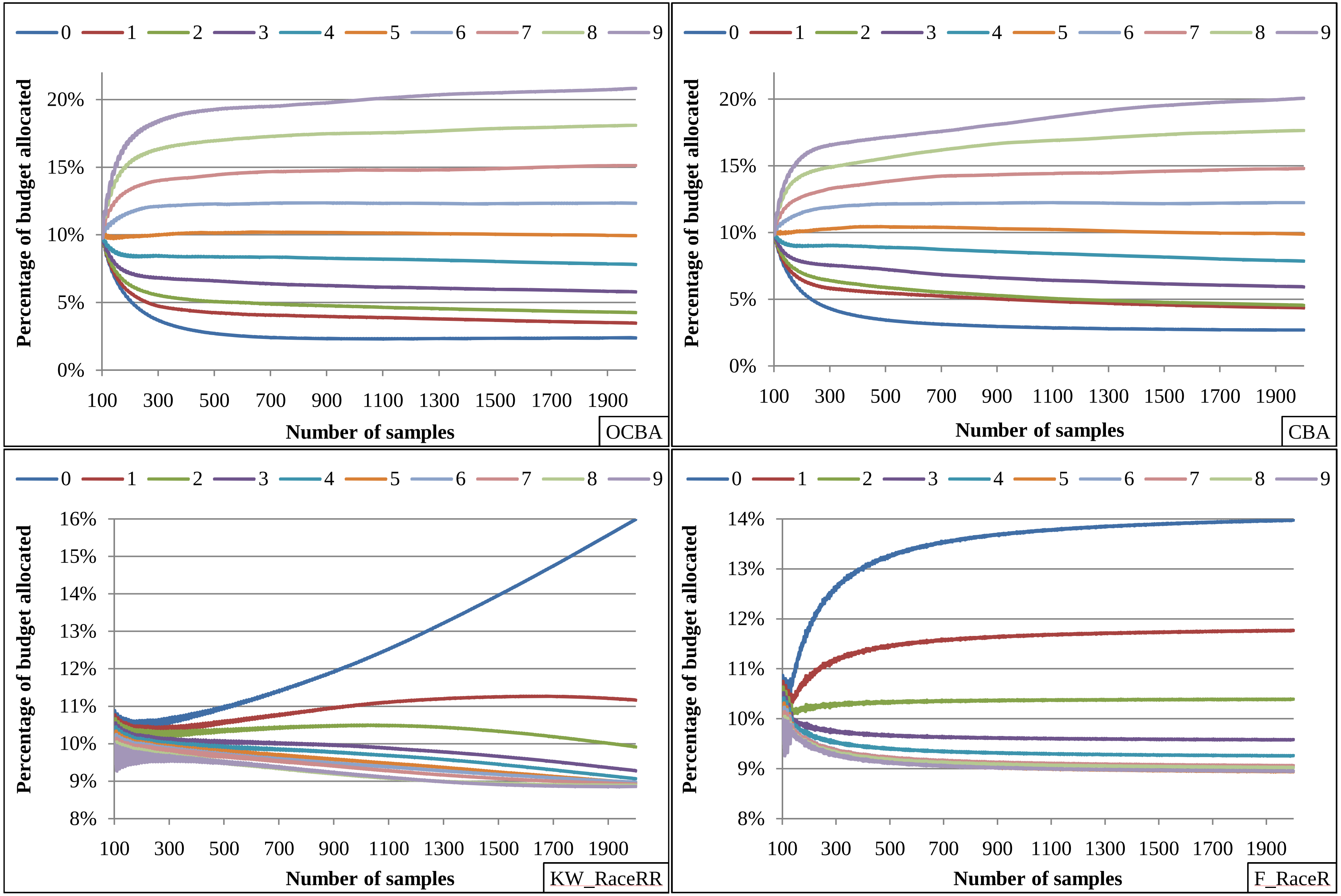


Figure 5.13. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}(i,(i+1)^4)\forall i = 0, \dots, k-1$. Correlation = 0.0.

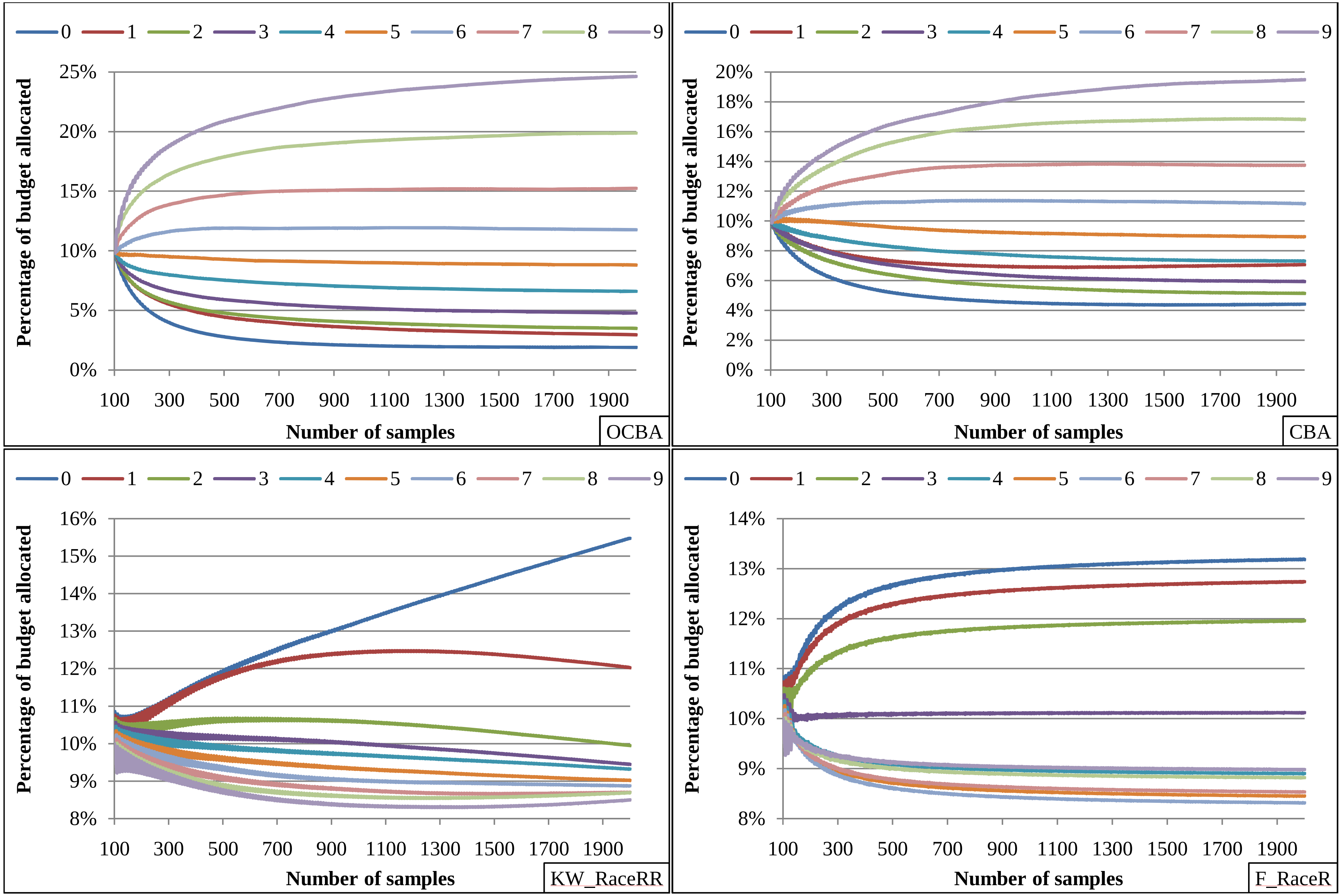


Figure 5.14. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}(i,(i+1)^4)\forall i = 0, \dots, k-1$. Correlation = 0.9.

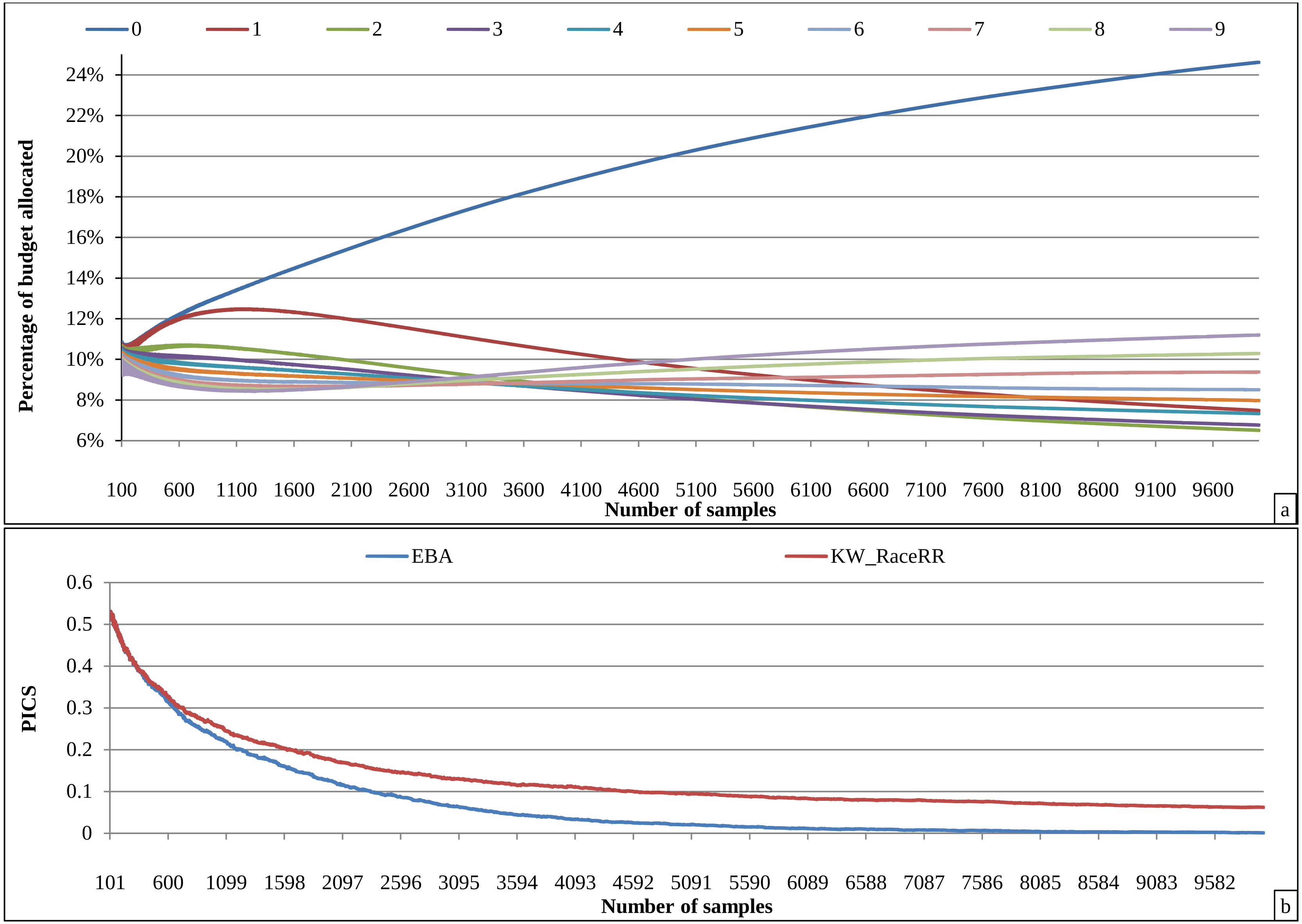


Figure 5.15. Cumulative allocations (a) by KW-RaceRR and convergence curves (b) based on $PICS$ for $\sim\mathcal{N}(i,(i+1)^4)\forall i=0,\dots,k-1$. Correlation = 0.9.

5.3.1.4 *Case* 4*: Means and variances are randomly selected*

The performance of an algorithm under various parameter settings is not likely to be as structured, in terms of the means and variances, as was presented up till now. In Case 4, no assumptions are made on the means and variance, instead they are drawn at random. Two sets of means and variances are drawn from $\sim\mathcal{N}\big(U(0,k),U(10{,}24)\big)$, with the exact values listed in Table 5.11. Generally, the variances in both sets are close to each other, and judging from the means, systems 0-2, in *Set*-A, and 0-3, in *Set*-B, may be the most crucial in identifying the best. Based on that, *Set*-B poses a harder problem because the distances between the means of systems 0-3 are smaller. The only expectation for this random case is that the crucial systems should be sampled more often, but it is unclear how the budget should be distributed among the remaining ones.

Table 5.11. The exact means and variances drawn from $\sim\mathcal{N}\big(U(0,k),U(10{,}24)\big)$.

Set	System	0	1	2	3	4	5	6	7	8	9
A[1]	Mean	0.10	0.98	1.32	3.27	6.21	6.49	8.03	8.34	9.10	9.78
	Variance	35.93	44.34	42.10	24.42	43.39	28.12	44.49	34.35	24.31	39.72
B	Mean	0.23	0.50	1.09	1.65	5.51	5.87	7.50	8.31	8.38	9.85
	Variance	39.70	39.40	39.43	29.34	46.50	34.92	35.69	37.39	45.04	40.77

Beginning with *Set*-A, Table 5.12 and Figure 5.16 show that, under 0.0 correlation, *KW*-RaceRR produces the best results from sample 700 (roughly) onwards, while OCBA is slightly better beforehand. This was observed earlier in Case 1 as well. *KW*-RaceRR_OCBA is following the behavior of OCBA and is not offering any noticeable advantage over it. At 0.9 correlation, CBA is doing equal allocation as can be seen from its overlapping curve with EBA. Also, *F*-RaceR and *A*-RaceR_2Way are converging fast at the beginning, then slowing down later on. Again, this was observed earlier in Case 1, and is expected here as the variances are close.

[1] The same means and variances used by Fu, M. C., Hu, J. Q., Chen, C. H. and Xiong, X., 2007. Simulation allocation for determining the best design in the presence of correlated sampling. *INFORMS Journal on Computing,* 19 (1), pp.101-111.

Table 5.12. Lowest *PICS* and *E[OC]* achieved in Case 4, *Set*-A, at 2000 samples for various correlation levels.

Algorithm	Correlation									
	0.0		0.3		0.6		0.9		Mixed	
	PICS	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*
EBA	0.084	0.007	0.044	0.002	0.012	0.000	0.000	0.000	0.055	0.003
*A*_RaceR_2Way	0.048	0.002	0.035	0.001	0.016	0.000	0.006	0.000	0.038	0.001
*F*_RaceR	0.035	0.001	0.023	0.001	0.013	0.000	0.002	0.000	0.030	0.001
CBA	0.063	0.004	0.031	0.001	0.012	0.000	0.000	0.000	0.025	0.001
OCBA	0.043	0.002	0.034	0.001	0.024	0.001	0.003	0.000	0.034	0.001
*A*_RaceRR_1WUB	0.021	0.000	0.011	0.000	0.003	0.000	0.000	0.000	0.021	0.000
*KW*_RaceRR	**0.009**	**0.000**	**0.005**	**0.000**	**0.001**	**0.000**	**0.000**	**0.000**	**0.004**	**0.000**
*KW*_RaceRR_OCBA	0.041	0.328	0.032	0.328	0.025	0.328	0.003	0.328	0.016	0.000

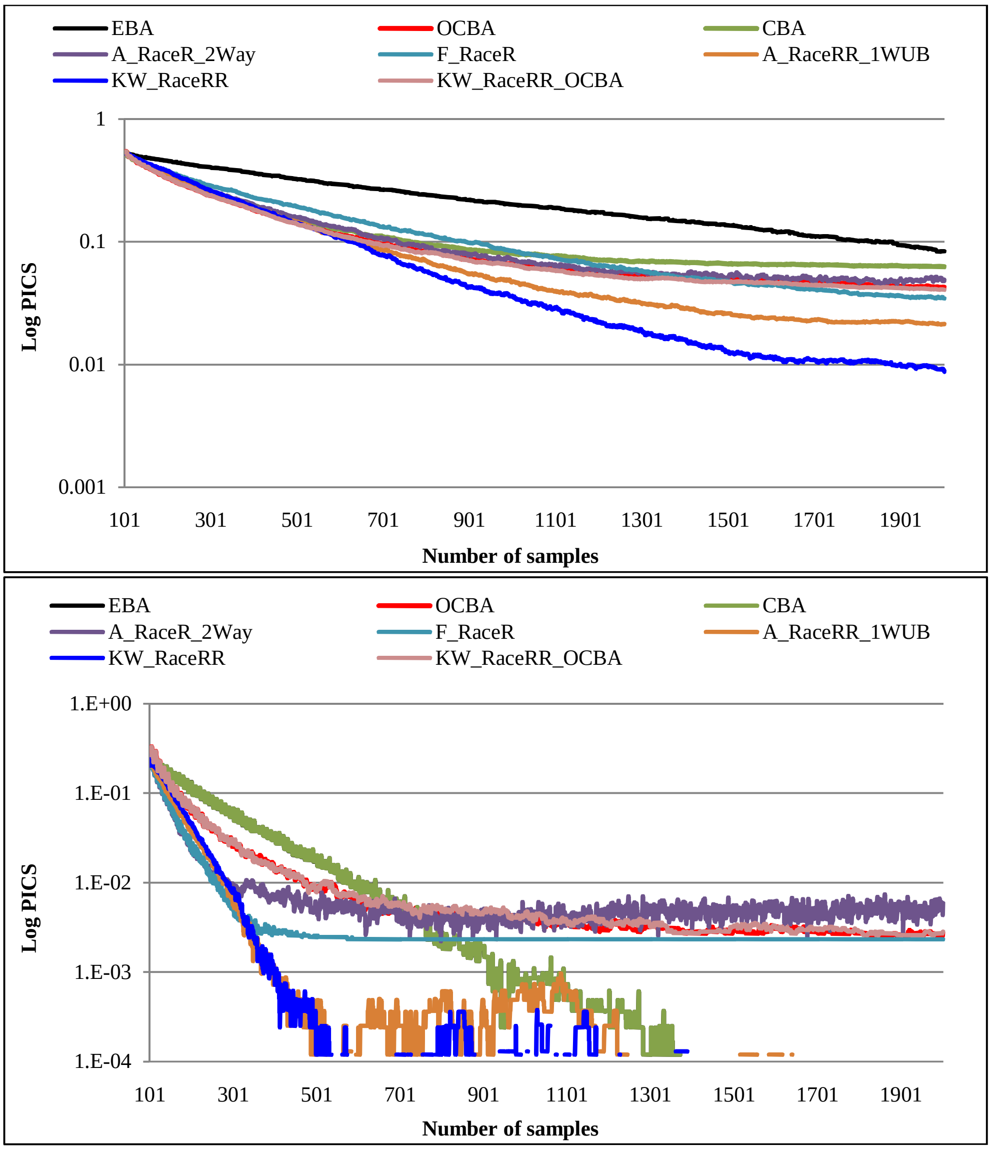


Figure 5.16. Comparison of different algorithms based on *PICS* and *E[OC]* under 0.0 (above) and 0.9 (below) correlation. $\sim\mathcal{N}\left(U(0,k), U(10,24)\right)$ *Set*-A.

Moving on to the analysis, the algorithms presented in Figure 5.17 all focus the majority of the budget to systems 0-2, which supports the conjecture of them being the crucial ones. For the independent case, *F*-RaceR and *A*-RaceR_2Way are allocating more samples to system 3. This has translated into inferior performance, indicating a better use of that system's budget was possible.

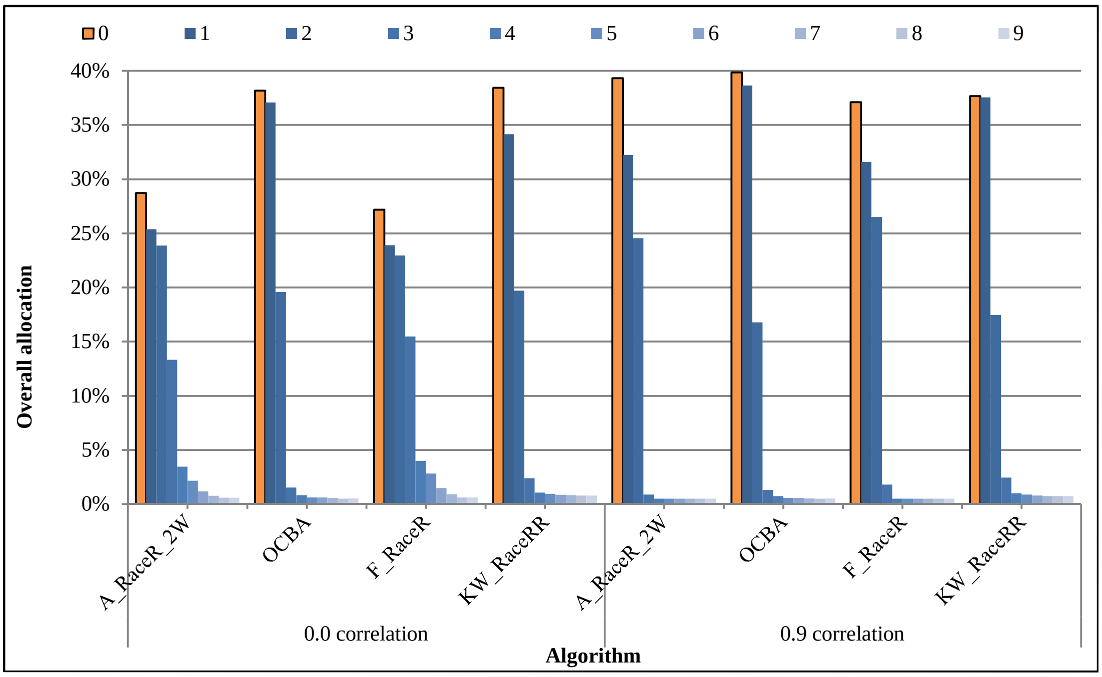


Figure 5.17. Overall allocation of different algorithms at 0.0 and 0.9 correlation. $\sim\mathcal{N}\big(U(0,k),U(10{,}24)\big)$ *Set*-A. The numbers are averaged over all replications.

The survival plots also reflect the observed performance, and the conclusions made from them are in line with the Case 1. In specific, under 0.0 correlation, Figure 5.18, OCBA is the fastest algorithm to separate the crucial systems, particularly system 0, which translates into is slightly faster convergence at the beginning. On the other hand, the budget allocated to the crucial systems is more evenly spread among them by *KW*-RaceRR, which invests more samples in system 2, compared to OCBA, before it discards it. This enabled it to achieve better estimates of the crucial systems' means, and ultimately reach the best results after sample 700, approximately. See Table 5.13.

Table 5.13. Deviation of the estimated means from their nominal values.
The numbers are reported at the end of the run, and are based on all replications.
$\sim\mathcal{N}\big(U(0,k),U(10{,}24)\big)$. *Set*-A. Note how *KW*-RaceRR has the lowest deviation for the crucial systems except system 0.0 under 0.9 correlation.

System	True mean	Deviation from true mean							
		0.0 correlation				0.9 correlation			
		*A*_RaceR_2W	*F*_RaceR	*KW*_RaceRR	OCBA	*A*_RaceR_2W	*F*_RaceR	*KW*_RaceRR	OCBA
0	0.10	0.117	0.087	**0.023**	0.134	0.001	0.000	0.002	0.005
1	0.98	0.341	0.313	**0.054**	0.064	0.196	0.158	**0.005**	0.048
2	1.32	0.382	0.337	**0.263**	0.339	0.219	0.191	**0.131**	0.371
3	3.27	0.447	0.407	0.318	0.477	-0.020	-0.039	0.119	0.511
4	6.21	0.352	0.384	0.286	0.402	-0.022	-0.022	0.213	0.408
5	6.49	0.185	0.240	0.173	0.176	-0.015	-0.015	0.083	0.155
6	8.03	0.146	0.185	0.140	0.257	-0.015	-0.015	0.093	0.209
7	8.34	0.068	0.088	0.090	0.129	-0.008	-0.008	0.036	0.085
8	9.10	0.007	0.019	0.024	0.017	-0.013	-0.013	-0.005	0.004
9	9.78	0.024	0.038	0.046	0.090	-0.012	-0.012	0.011	0.054

The same extends to 0.9 correlation in Figure 5.19, where it is clear that *F*-RaceR and *A*-RaceR_2Way are the fastest algorithms to separate the crucial systems, and, compared to OCBA, are distributing their budget more evenly. This is connected with their quick convergence at the start. *KW*-RaceRR follows the same pattern, but to a greater extent with systems 0-2 being sampled equally till about sample 500 (the investment). This is linked with the somewhat slower convergence at the start, but superior performance later on.

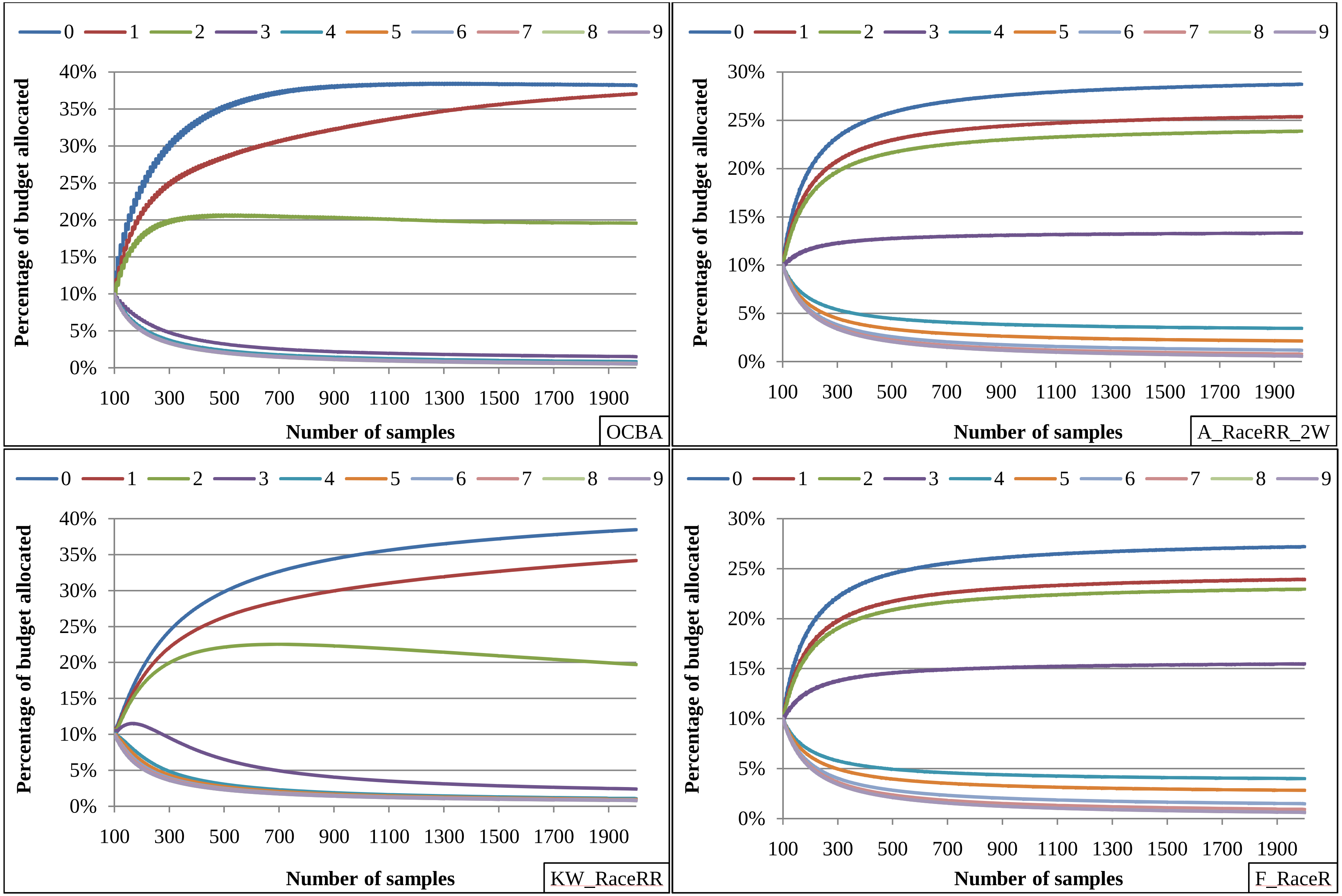


Figure 5.18. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}\left(U(0,k),U(10,24)\right)$ *Set*-A. Correlation = 0.0.

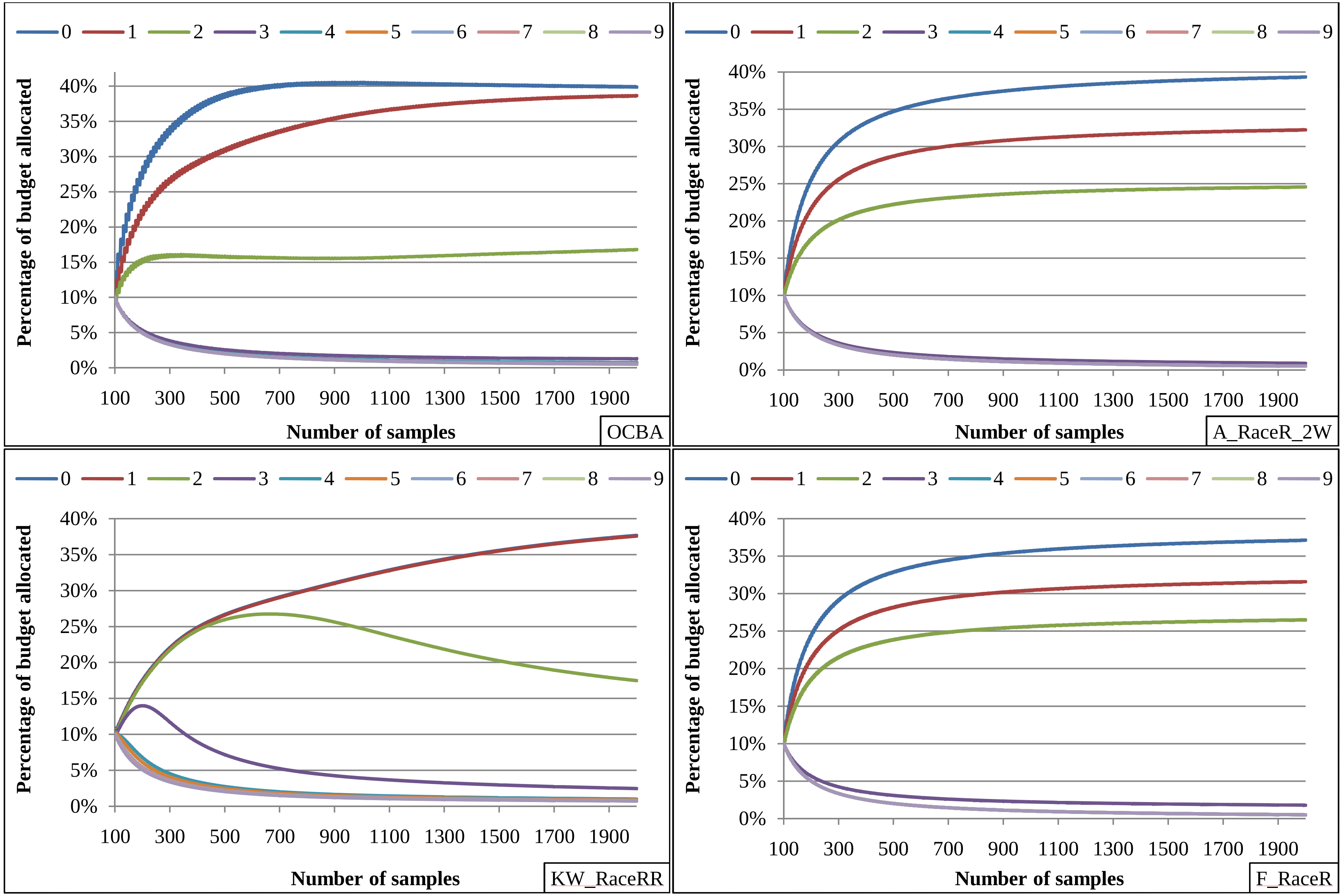


Figure 5.19. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}\left(U(0,k),U(10,24)\right)$ *Set*-A. Correlation = 0.9.

Moving to *Set*-B, Table 5.14 and Figure 5.20 indicate that, with the exception of *F*-RaceR and EBA, all algorithms perform closely to each other, with *KW*-RaceRR achieving better results, to some extent, by the end of the run. All the figures here are moved to Appendix A2 as they are similar to *Set*-A.

Table 5.14. Lowest *PICS* and *E[OC]* achieved in Case 4, *Set*-B, at 2000 samples.

	Correlation									
Algorithm	0.0		0.3		0.6		0.9		Mixed	
	PICS	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*	*PICS*	*E[OC]*
EBA	0.348	0.040	0.306	0.028	0.238	0.016	0.076	0.002	0.316	0.029
*A*_RaceR_2Way	0.172	0.009	0.118	0.004	0.045	0.001	0.014	0.000	0.130	0.005
*F*_RaceR	0.260	0.012	0.2050	0.012	0.126	0.004	0.024	0.000	0.223	0.014
CBA	0.196	0.011	0.250	0.019	0.238	0.016	0.076	0.002	0.181	0.010
OCBA	0.163	0.007	0.118	0.004	0.067	0.001	0.017	0.000	0.130	0.005
*A*_RaceRR_1WUB	0.164	0.008	0.116	0.004	0.061	0.001	0.002	0.000	0.164	0.008
*KW*_RaceRR	**0.159**	**0.007**	**0.113**	**0.003**	**0.060**	**0.001**	**0.001**	**0.000**	**0.126**	**0.004**
*KW*_RaceRR_OCBA	0.166	0.008	0.119	0.004	0.067	0.001	0.018	0.000	0.090	0.002

The overall allocations and survival plots are interpreted in the same fashion as in *Set*-A, and offer the same explanation for the observed performance. It also holds that *KW*-RaceRR better estimates the means of the crucial systems as indicated by Table 5.15. However, it is unclear why *F*-RaceR performs worse than *A*-RaceR_2Way in the independent case, since their survival plots are similar. Their mean estimates for the best four systems are also close, though *F*-RaceR gives a more accurate estimate of system 0.

A more important outcome from Case 4 is that while the variances are unequal, OCBA\CBA is not gaining much from utilizing σ_i/σ_j, as seen before when the variances were exponentially increasing or decreasing.

Table 5.15. Deviation of the estimated means from their nominal values.
The numbers are reported at the end of the run, and are based on all replications.
$\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$. *Set*-B.

System	True mean	Deviation from true mean							
		0.0 correlation				0.9 correlation			
		*A*_RaceR_2W	*F*_RaceR	*KW*_RaceRR	OCBA	*A*_RaceR_2W	*F*_RaceR	*KW*_RaceRR	OCBA
0	0.10	0.184	0.172	**0.049**	0.157	0.023	0.024	**0.006**	0.042
1	0.98	0.248	0.227	**0.059**	0.112	0.059	0.061	**0.004**	0.052
2	1.32	0.363	0.322	**0.317**	0.703	0.153	0.146	**0.022**	0.866
3	3.27	0.375	0.324	**0.338**	0.656	-0.074	-0.061	**-0.011**	0.709
4	6.21	0.489	0.477	0.425	0.331	-0.023	-0.023	0.129	0.155
5	6.49	0.322	0.344	0.308	0.169	-0.017	-0.017	-0.003	0.044
6	8.03	0.132	0.151	0.169	0.075	-0.013	-0.013	-0.001	0.000
7	8.34	0.075	0.089	0.116	0.062	-0.008	-0.008	0.005	-0.002
8	9.10	0.091	0.115	0.143	0.074	-0.017	-0.017	0.025	-0.003
9	9.78	0.024	0.031	0.062	0.028	-0.012	-0.012	-0.002	-0.011

In conclusion, if the variances are not equal, but are relatively close, the tested algorithms behaved similarly to Case 1. *KW*-RaceRR is the best algorithm, on both measures, if the budget is high, irrespective of the correlation. For a low budget, OCBA is better to some extent for the independent case, and *A*-RaceR_2Way is better if correlation is high. The values of “high” and “low” will depend on the sets of means and variances used, but for the experiments of Case 4, a budget of 1500 samples, or more, was sufficient for *KW*-RaceRR to outperform the other algorithms.

5.3.1.5 *Summary*

To summarize the results of *Set*-1 experiments, the following remarks are made: **First**, if the variances are equal, or if they are randomly selected, but are relatively close, *KW*-RaceRR performs best if the budget is high, regardless of the correlation level. **Second**, for Case 1 with a low budget, OCBA performs *slightly* better under no correlation, and 2-way Racing is best if correlation is high. **Third**, if the variances are exponentially increasing or decreasing, OCBA\CBA makes more use of σ_i/σ_j and outperforms the other algorithms throughout the run, although for the decreasing variance case *A*-RaceR_2Way was best under high correlation. **Fourth**, trying to intelligently allocate Δ in *KW*-RaceRR using OCBA is advantageous when the variances are exponentially increasing or decreasing, otherwise it is similar to OCBA. **Fifth**, under high correlation, EBA is competitive, given its simplicity, but it is not the best strategy. See Table 5.16.

Table 5.16. A summary of the best performing algorithms (based on *PICS*) in *Set*-1 experiments.

Case	Distribution	Budget	Correlation	Best algorithm
1	$\sim\mathcal{N}(i,6^2)\forall i=0,\ldots,k-1$	≥ 700	0.0 or 0.9	*KW*-RaceRR
		≤ 500	0.0	OCBA
		≤ 350	0.9	*F*-RaceR
2	$\sim\mathcal{N}(i,(k-i)^3)\forall i=0,\ldots,k-1$	Throughout	0.0	OCBA
		Throughout	0.9	*A*-RaceR_2Way
3	$\sim\mathcal{N}(i,(i+1)^4)\forall i=0,\ldots,k-1$	Throughout	0.0 or 0.9	OCBA
4	$\sim\mathcal{N}\big(U(0,k),U(10{,}24)\big)$ *Set*-A and *Set*-B	High[2]	0.0 or 0.9	*KW*-RaceRR
		Low[2]	0.0	OCBA
		Low[2]	0.9	*A*-RaceR_2Way

[2] The exact values will depend on the means and variances used.

5.3.2 Results for *Set*-2

In these experiments Racing algorithms set the budget for OCBA, CBA, and EBA. The performance measures are f_r and $\mathbb{E}[OC]$ tracked over a wide range of α values and calculated over 10,000 replications for each α. The objectives of *Set*-2 experiments are to see if allowing Racing to set the budget will affect its performance against OCBA\CBA, and to show how Racing with reset improves upon the standard algorithms by adapting α_0. In previous work it was shown that *F*-RaceR improves standard *F*-Race by adapting α_0 (Branke and Elomari, 2013). In addition, *F*-RaceR, with a relatively high α_0, achieved significantly better results compared to *F*-Race using any fixed α set by the user.

Next, *KW*-Race and *A*-Race_2Way will be the algorithms setting the budget, as they have shown to be competitive to OCBA\CBA in *Set*-1. EBA is added to the comparison as a benchmark, along with Racing with reset (*KW*-RaceRR and *A*-RaceR_2Way) to show the performance gains over the standard Racing algorithms. The means, variances, and correlation levels are the same as those used in *Set*-1. Survival plots cannot be generated for *Set*-2 experiments as the consumed budget varies with each replication; hence, ECs are used instead. To make this chapter easier to read, the experiments of *A*-Race_2Way are moved to the appendix as their conclusions do not differ from *KW*-Race's as will be shown.

Before going into the results and analysis, it should be noted that as α decreases, the budget required for termination increases. With that in mind, there is always some α that will force Racing to consume the entire budget, at which point Racing is not "setting" the budget, but is using a fixed budget, as in *Set*-1, without the reset. For such α values, the comparison to OCBA\CBA and EBA does not serve the objectives of *Set*-2 experiments.

Figure 5.20 shows the average consumed budget by *KW*-Race for all α values and correlation levels used, along with box plots. Generally, for $\alpha > 0.1$ and $\rho \leq 0.3$ the average budget consumed is less than the allowed limit, and in some cases, even the maximum is less than the limit. In such situations, *KW*-Race is indeed setting the budget for the other algorithms. At the other extreme, $\alpha < 0.001$, the average consumed budget is almost equal to the allowed maximum, which means that, most of the time, *KW*-Race was not able to find a winner. In these situations, *KW*-Race is not specifying the budget and is not expected to have performance advantage.

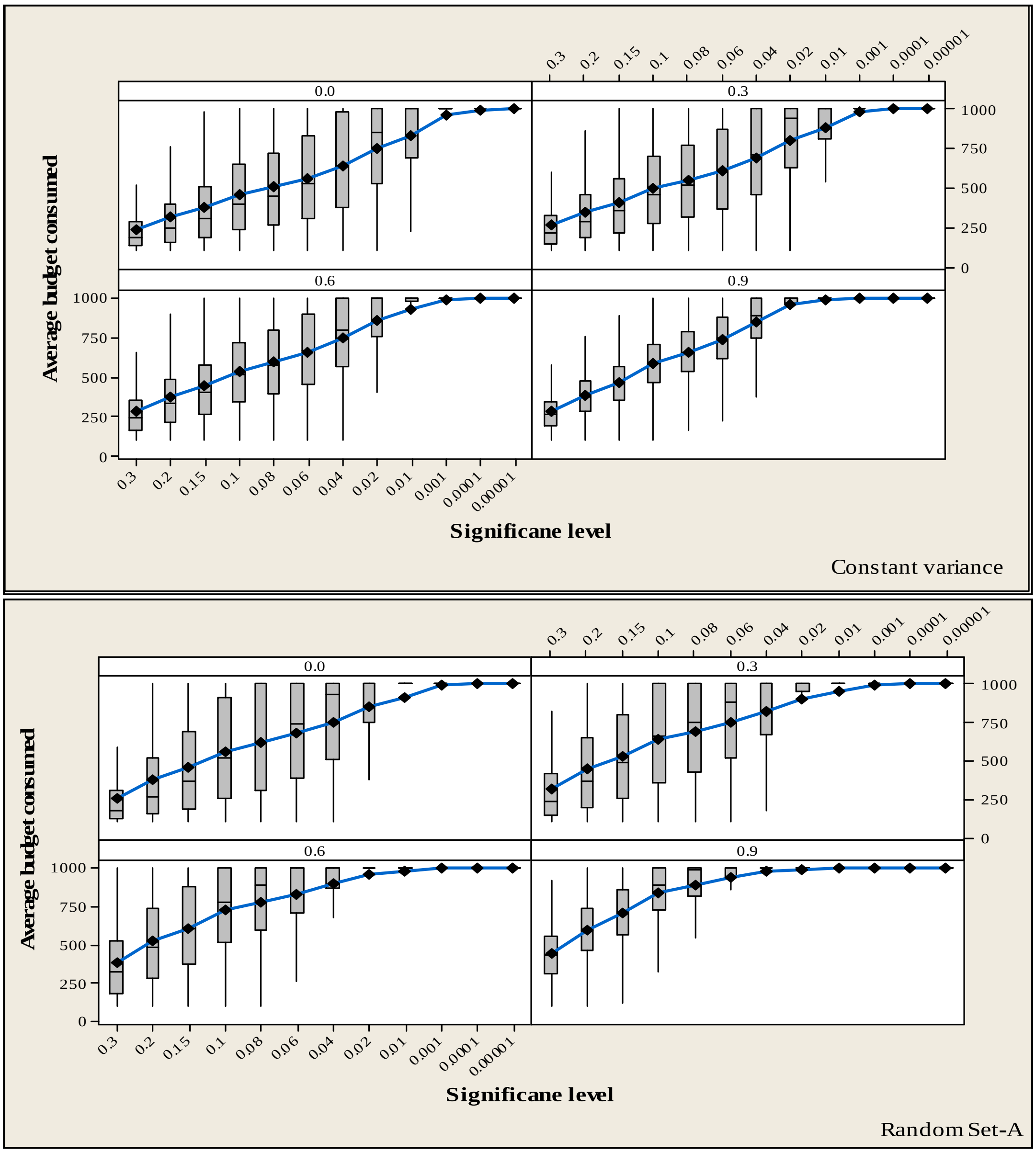


Figure 5.20. Box plots of the budget consumed by *KW*-Race at various α values, and different correlation levels. The connecting line is that of the means.

5.3.2.1 *Case* 1 *and Case* 4*: Monotonically increasing means with equal variances, and means and variances are randomly selected*

These two cases are combined together because their conclusions are similar. In Case 1, the box plots indicate that *KW*-Race sets the budget, in most replications, when $0.3 \geq \alpha \geq 0.06$. Within that range, *KW*-Race outperforms all other algorithms as seen in Figure 5.21 (the *E[OC]* results are in Appendix A3) and Table 5.17, though the difference is minor at $\alpha = 0.3$ and $\rho = 0.0$ or 0.3. Note that there results for $\rho = 0.0$ and 0.3 are summarized in the tables, but are not plotted as they follow the same pattern. For $\alpha \leq 0.01$ *KW*-Race starts to consume the entire budget in most replications, and behave similar to EBA, eventually degrading its performance. Note that under high correlation and low α, all algorithms reach zero f_r and $\mathbb{E}[OC]$. This is in line with the results of *Set*-1.

The same observations apply to Case 4 *Set*-A and *Set*-B, but for different ranges of α. See Tables 5.18-5.19, the remaining results are in Appendix A3. Note how, for the independent case of *Set*-A, OCBA is better for high α values (lower budget), and *KW*-Race is better for low α values (high budget). In *Set*-B, under no correlation, OCBA, CBA, and *KW*-Race perform closely to each other. Both of these behaviors were seen in *Set*-1 experiments earlier.

Finally, in Case 1 and Case 4, *KW*-RaceRR's performance is fairly stable over a wide range of α values, and is significantly better ($p - value \ll 0.05$), at a high α_0 (e.g. 0.3), than *KW*-Race at any fixed α. This does not apply for small α_0 values as it starts to act like EBA and degrades its performance.

Table 5.17. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}(i, 6^2)\forall i = 0, \dots, k-1$.

	0.3 correlation					0.6 correlation				
Alpha	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR
0.00001	0.030	0.006	0.041	0.079	0.030	0.006	0.003	0.029	0.029	0.006
0.001	0.017	0.006	0.041	0.078	0.017	0.003	0.003	0.029	0.030	0.003
0.01	0.010	0.006	0.042	0.079	0.010	0.001	0.003	0.030	0.030	0.001
0.1	0.004	0.007	0.045	0.088	0.003	0.000	0.003	0.033	0.034	0.000
0.3	0.083	0.093	0.184	0.279	0.000	0.023	0.044	0.173	0.179	0.000

Table 5.18. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$. *Set*-A.

	0.3 correlation					0.6 correlation				
Alpha	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR
0.00001	0.304	0.261	0.338	0.390	0.304	0.236	0.186	0.324	0.324	0.235
0.001	0.291	0.261	0.336	0.391	0.291	0.226	0.187	0.322	0.323	0.226
0.01	0.282	0.261	0.338	0.390	0.282	0.219	0.185	0.323	0.324	0.218
0.1	0.265	0.260	0.338	0.390	0.264	0.203	0.186	0.324	0.326	0.203
0.3	0.310	0.329	0.423	0.494	0.219	0.180	0.217	0.384	0.388	0.145

Table 5.19. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$. *Set*-B.

	0.3 correlation					0.6 correlation				
Alpha	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR
0.00001	0.032	0.045	0.006	0.141	0.031	0.008	0.028	0.000	0.068	0.007
0.001	0.021	0.045	0.011	0.140	0.021	0.005	0.028	0.000	0.067	0.005
0.01	0.015	0.045	0.026	0.142	0.015	0.003	0.027	0.000	0.068	0.003
0.1	0.012	0.046	0.066	0.145	0.012	0.002	0.028	0.001	0.069	0.002
0.3	0.127	0.158	0.115	0.360	0.013	0.035	0.081	0.003	0.236	0.003

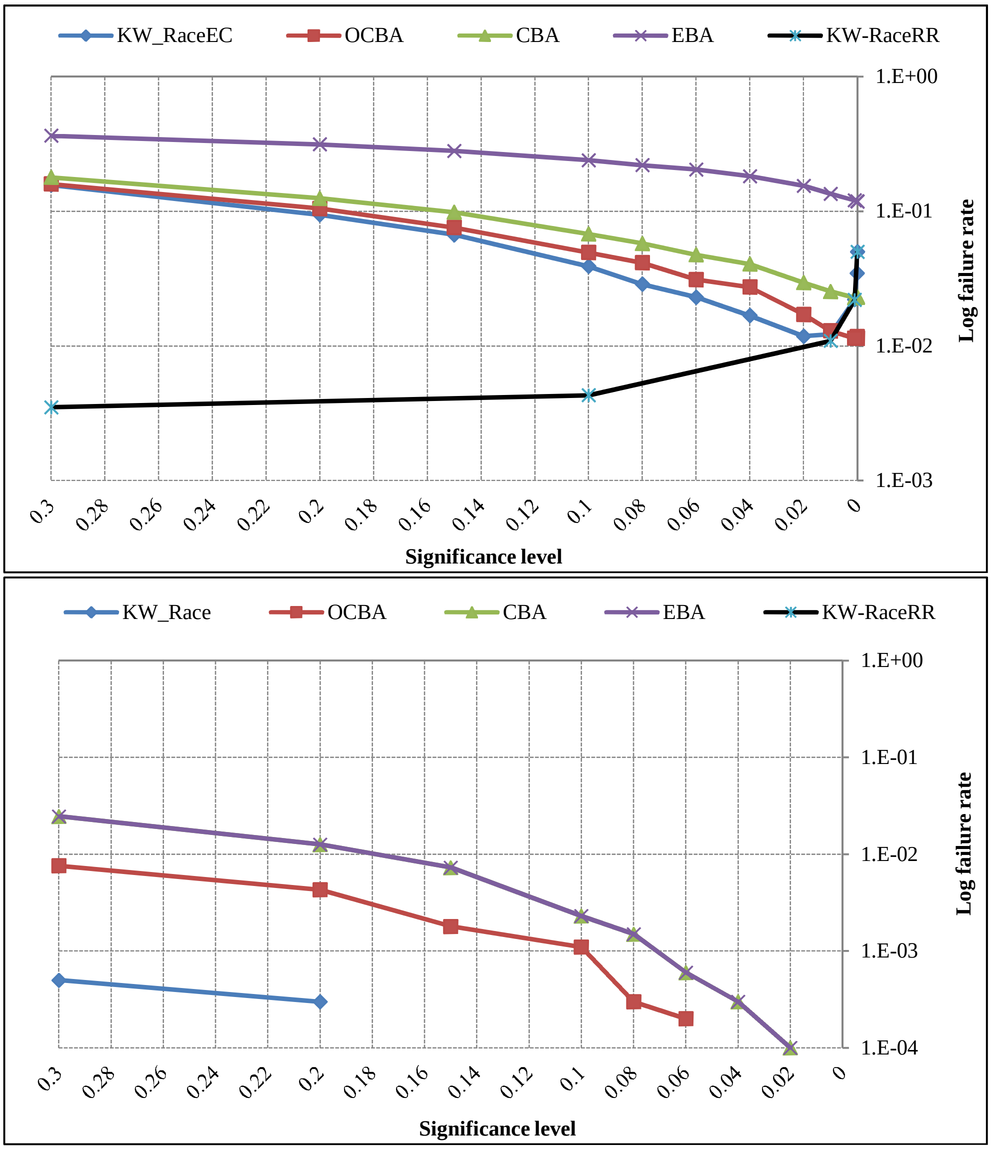


Figure 5.21. ECs based on f_r at 0.0 (above) and 0.9 (below) correlation levels. Data are drawn from $\sim\mathcal{N}(i,6^2)\forall i = 0,\dots,k-1$.

5.3.2.2 *Case* 2 *and Case* 3*: Monotonically increasing means with increasing or decreasing variance*

In *Set*-1 experiments, both these cases had OCBA outperforming KW-RaceRR due to utilizing σ_i/σ_j. The only exception was for the decreasing variance case under high correlation. If allowing KW-Race to set the budget gives it an advantage, it should become apparent here. The results of Figures 5.22-5.23 and Tables 5.20-5.21suggest otherwise. OCBA is still better than KW-Race in both cases, with the same exception.

Moreover, notice how for the decreasing variance case KW-Race consumes nearly all the budget over all α values if correlation is 0.9. This limits the number of resets of KW-RaceRR, which explains why its performance is close to KW-Race's, and it explains why OCBA's performance is rather stable. Finally, KW-RaceRR, at a high α_0 (say 0.3 or 0.2), performs significantly better ($p - value \ll 0.05$) than KW-Race with a fixed α.

Table 5.20. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}(i,(k-i)^3)\forall i = 0,\dots,k-1$.

	0.3 correlation					0.6 correlation				
Alpha	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR
0.00001	0.470	0.383	0.481	0.471	0.469	0.413	0.341	0.423	0.420	0.413
0.001	0.468	0.384	0.481	0.473	0.468	0.409	0.342	0.423	0.419	0.409
0.01	0.465	0.385	0.483	0.474	0.465	0.406	0.341	0.422	0.420	0.406
0.1	0.456	0.385	0.482	0.473	0.456	0.396	0.339	0.422	0.417	0.396
0.3	0.436	0.421	0.495	0.510	0.415	0.352	0.359	0.436	0.430	0.348

Table 5.21. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}(i,(i+1)^4)\forall i = 0,\dots,k-1$.

	0.3 correlation					0.6 correlation				
Alpha	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR	*KW*_Race	OCBA	CBA	EBA	*KW*_RaceRR
0.00001	0.425	0.072	0.076	0.442	0.425	0.331	0.078	0.110	0.341	0.330
0.001	0.425	0.072	0.076	0.442	0.425	0.331	0.078	0.110	0.341	0.330
0.01	0.426	0.072	0.077	0.443	0.426	0.330	0.077	0.111	0.342	0.330
0.1	0.419	0.074	0.078	0.444	0.419	0.328	0.077	0.110	0.341	0.328
0.3	0.448	0.203	0.211	0.545	0.356	0.346	0.150	0.176	0.390	0.327

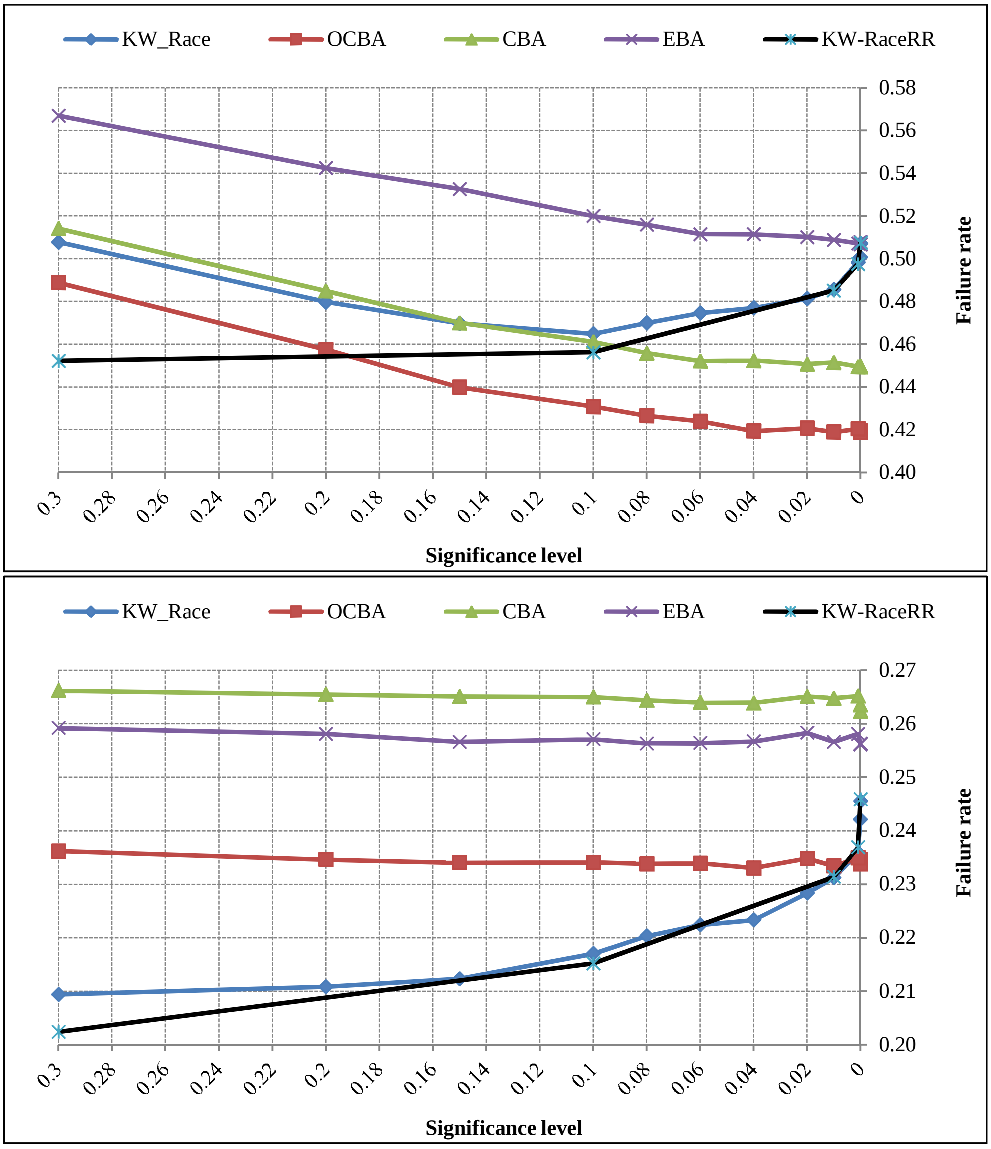


Figure 5.22. ECs based on f_r at 0.0 (above) and 0.9 (below) correlation levels. Data are drawn from $\sim\mathcal{N}(i,(k-i)^3)\forall i=0,\dots,k-1$.

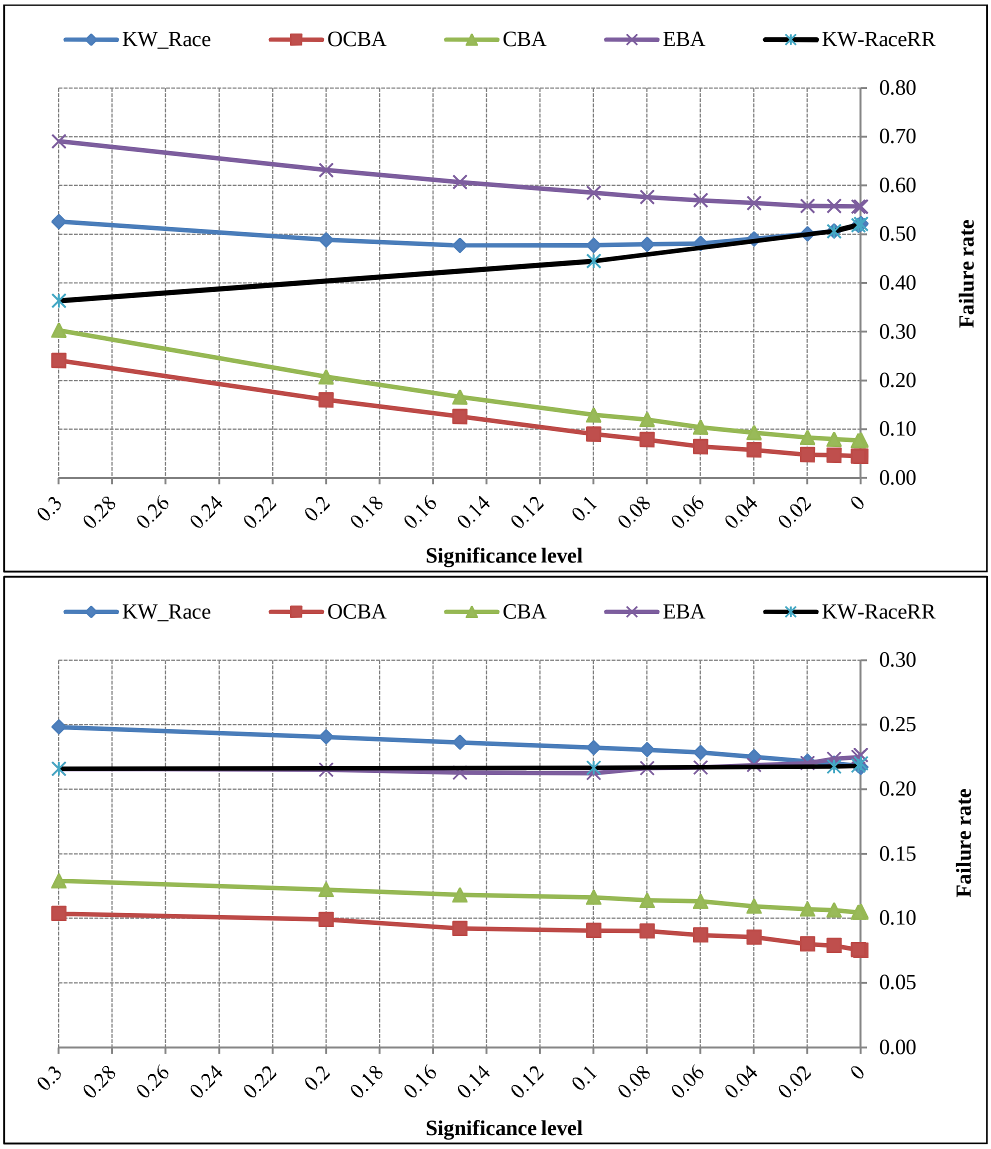


Figure 5.23. ECs based on f_r at 0.0 (above) and 0.9 (below) correlation levels. Data are drawn from $\sim\mathcal{N}(i,(i+1)^4)\forall i = 0,\dots,k-1$.

5.3.2.3 *Summary*

The results presented in this section show that having KW-Race set the budget does not affect its performance ranking among OCBA, CBA, and EBA. That is, if the budget is fixed and KW-RaceRR performs worse than OCBA, it will again perform worse if it specifies the budget, given the same means, variances, and correlations. Specifically, KW-Race remains better if the variances are equal, or are randomly selected, and it is worse if the variances are exponentially increasing or decreasing (except under high correlation). This pattern was observed in *Set*-1 before when KW-RaceRR used a fixed budget.

When α is small enough, KW-Race, and other Racing algorithms in general, sample all systems almost equally, eventually degrading their performance and moving it towards EBA's. KW-RaceRR improves over KW-Race by adapting α_0, and shows a somewhat stable performance over a wide range of α_0. Nonetheless, there is always some α_0 at which KW-RaceRR performs similar to EBA. More importantly, as presented before in Branke and Elomari (2013), KW-RaceRR at a high α_0 (e.g. 0.3 or 0.2) achieves significantly better ($p - value \ll 0.05$) results than KW-Race at any fixed α. The same conclusions hold for *A*-Race_2Way and A-RaceR_2Way, as shown in Appendix A2.

Table 5.22. A summary of the best performing algorithms (based on *PICS*) in *Set*-2 experiments.

Case	Distribution	α range	Correlation	Best algorithm
1	$\sim\mathcal{N}(i,6^2)\forall i = 0,\ldots,k-1$	0.3-0.02	0.0	KW-Race
		0.01-0.00001[3]	0.0	OCBA
		0.3-0.00001	0.9	KW-Race
2	$\sim\mathcal{N}(i,(k-i)^3)\forall i = 0,\ldots,k-1$	0.3-0.00001	0.0	OCBA
		0.3-0.01	0.9	KW-Race
		0.001-0.00001[3]	0.9	OCBA
3	$\sim\mathcal{N}(i,(i+1)^4)\forall i = 0,\ldots,k-1$	0.3-0.00001	0.0 and 0.9	OCBA
4	$\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$ *Set*-A	0.3-0.2 (low budget)	0.0	OCBA
		0.1-0.00001	0.0	KW-Race
		0.3-0.00001	0.9	KW-Race
	$\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$ *Set*-B	0.3-0.02	0.0	KW-Race
		0.01-0.00001[3]	0.0	OCBA
		0.3-0.08	0.9	KW-Race
		0.01-0.00001[3]	0.9	OCBA

[3] KW-Race starts behaving like EBA for these α values.

5.3.3 Results for *Set*-3

Based on the findings of *Set*-1 experiments, KW-RaceRR and OCBA outperformed all other algorithms in most cases within the available budget. Recall that under high correlation 2-way Racing algorithms achieve better results if the budget is small, but due to time limitation, they are not run here. *Set*-3 experiments consist of testing KW-RaceRR and OCBA on $k = 50$ systems and non-normal distributions with 0.0 and 0.9 correlation levels. Performance comparisons are based on $PICS$ only.

The settings are similar to *Set*-1. However, the variances cannot be *exponentially* increasing or decreasing due to the large number of systems; instead, they linearly increase and decrease as: $\sim\mathcal{D}\big(i, 3(k-i)\big)\forall i = 0, \dots, k-1$ and $\sim\mathcal{D}\big(i, 4(i+1)\big)\forall i = 0, \dots, k-1$. The equal variances case remains unchanged, and the random case now draws number from $\sim\mathcal{D}\big(U(0,15), U(20,40)\big)$. The exact values are in Table 5.23. Finally, the non-normal distributions are Weibull(1, 8), or Gamma(5, 6). See Figure 5.24.

Table 5.23. Means and variances drawn from $\sim\mathcal{D}\big(U(0,15), U(20,40)\big)$.

System	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Mean	0.10	1.36	1.43	1.77	2.19	2.36	2.40	2.57	3.04	3.07	3.50	3.50	3.79
Variance	27.76	31.06	28.96	31.56	28.46	33.42	29.66	31.75	36.38	27.59	33.14	29.08	32.74
System	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
Mean	3.83	3.97	3.98	3.99	4.03	4.42	4.42	4.49	4.80	4.86	5.63	5.71	5.81
Variance	26.01	31.08	32.16	34.44	27.82	33.45	31.42	31.91	32.56	29.58	25.59	34.12	24.86
System	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**
Mean	6.02	6.26	6.57	6.91	7.05	7.36	7.59	7.72	7.88	8.26	8.44	8.57	9.19
Variance	27.05	29.43	28.45	34.13	33.43	32.48	30.03	30.75	32.84	32.23	24.69	26.68	27.74
System	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**		
Mean	9.55	9.61	9.86	9.86	10.28	10.34	10.35	10.36	10.56	10.69	10.89		
Variance	33.73	27.22	26.00	25.41	23.90	33.30	36.66	29.22	28.92	35.63	33.14		

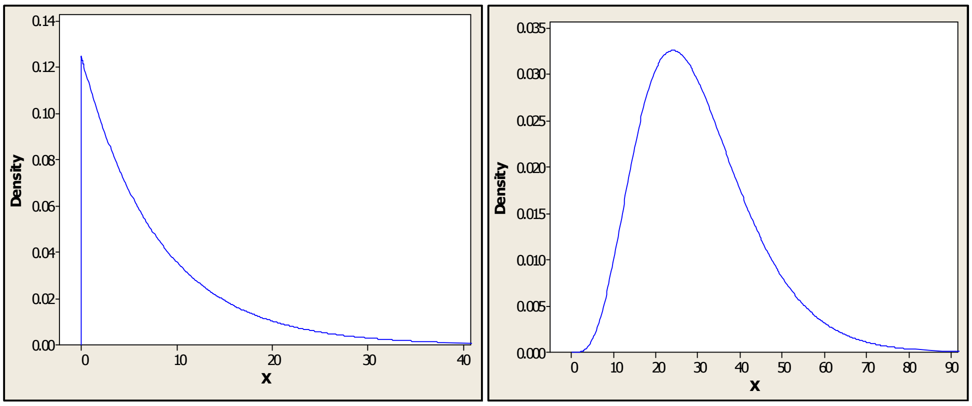


Figure 5.24. Weibull (left) and Gamma (right) distributions used in *Set*-3 experiments.

5.3.3.1 *Case* 1*: Monotonically increasing means with equal variances*

Figure 5.25 shows that, under 0.0 correlation, both algorithms perform closely to each other, over all three distributions, with KW-RaceRR reaching better $PICS$ values by the end of the run, and performing slightly worse when the budget is small. This was observed before in *Set*-1 where KW-RaceRR distributed Δ more evenly among the surviving systems, at the beginning of the run, such that it obtained more accurate estimates of their parameters before discarding them. The same pattern can be seen here in the corresponding survival, Figures 5.26-5.28. The legend on the figures does not display all systems for better viewing.

Note that the survival plots of KW-RaceRR are smoother, and do not display an oscillating pattern. This is because each additional sample allocated by KW-RaceRR goes to a different system (recall that it equally distributes $\Delta = number\ of\ survivors$ in each iteration), while OCBA may allocate anything between 0 and Δ to a single system, hence the oscillations. This pattern was observed to a less extent in *Set*-1 because *k* was smaller.

At 0.9 KW-RaceRR converges faster, then slows down, and half way through the run it, and OCBA, reach almost a zero $PICS$. This is clearer in non-normal distributions. The corresponding survival plots show that KW-RaceRR samples the third best more often in the beginning before discarding it. As this links with faster convergence, it means that the third best is a crucial system at that point, and a good estimate of its parameters is required for correct selection. Figure 5.29 compares the survival plots of KW-RaceRR across distributions. Note how the third best survives the longest with Normal data, and the least with Weibull data. See also Table 5.24.

Table 5.24. Deviation of the estimated means, of systems 0-2, from their nominal values. The numbers are based on KW-RaceRR allocations by the end of the run under 0.9 correlation.

		Normal	Weibull
System	Nominal mean	Deviation from nominal	Deviation from nominal
0	0	0.00	0.00
1	1	0.00	0.00
2	2	-0.01	0.18

In conclusion, increasing the number of systems and/or changing the distribution type has minor effect the performance of OCBA and KW-RaceRR, compared to *Set*-1 experiments. With 0.0 correlation, OCBA is still better if the budget is low, and KW-RaceRR is superior if the budget is high. With 0.9 correlation, KW-RaceRR is superior, but is more sensitive to the distribution type compared to OCBA. This has the effect of slowing its convergence half way through, but it reaches a near zero $PICS$ shortly afterwards, as does OCBA.

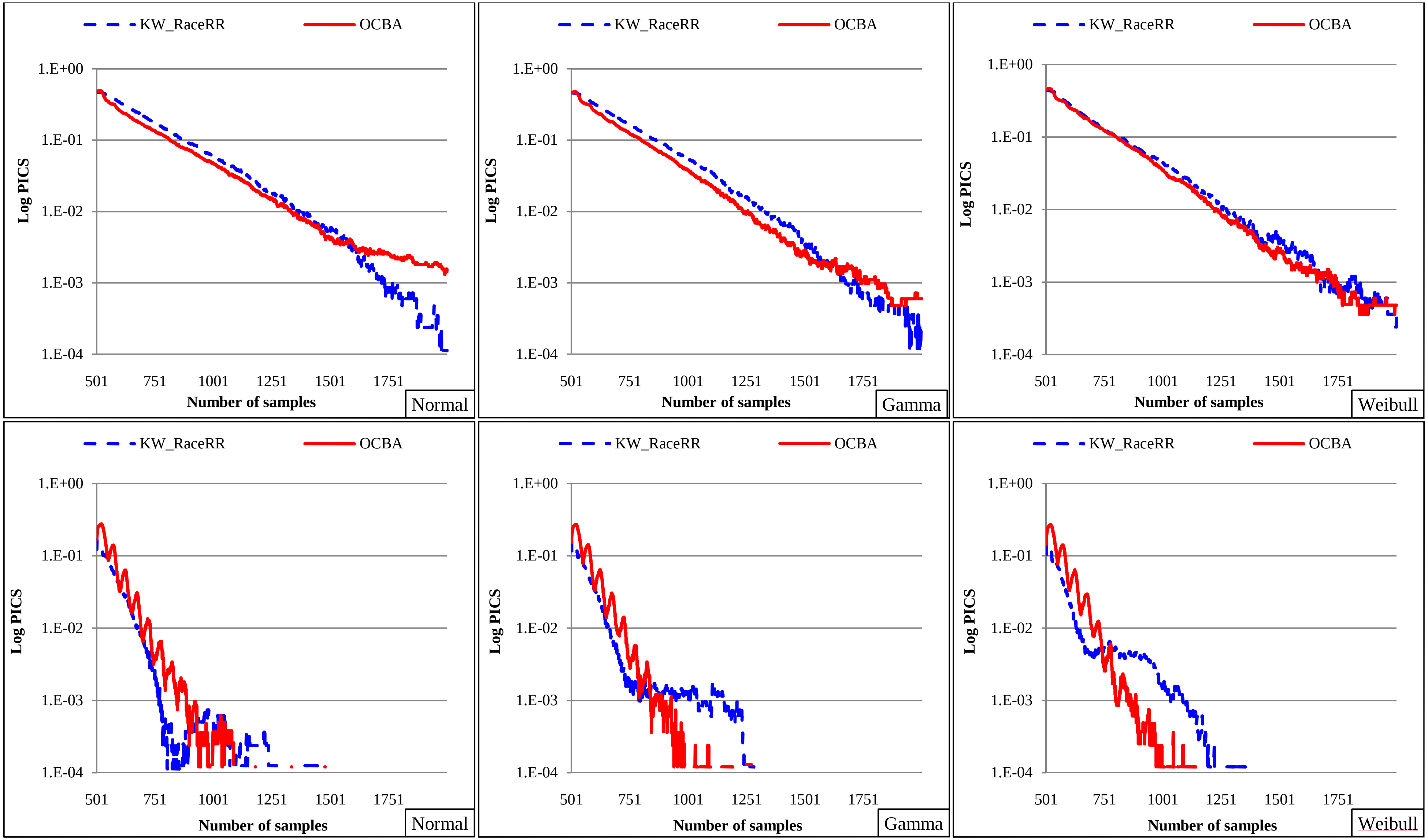


Figure 5.25. Comparison of different algorithms based on *PICS* under 0.0 (above) and 0.9 (below) correlation. $\sim\mathcal{D}(i,6^2)\forall i = 0,\dots,k-1$.

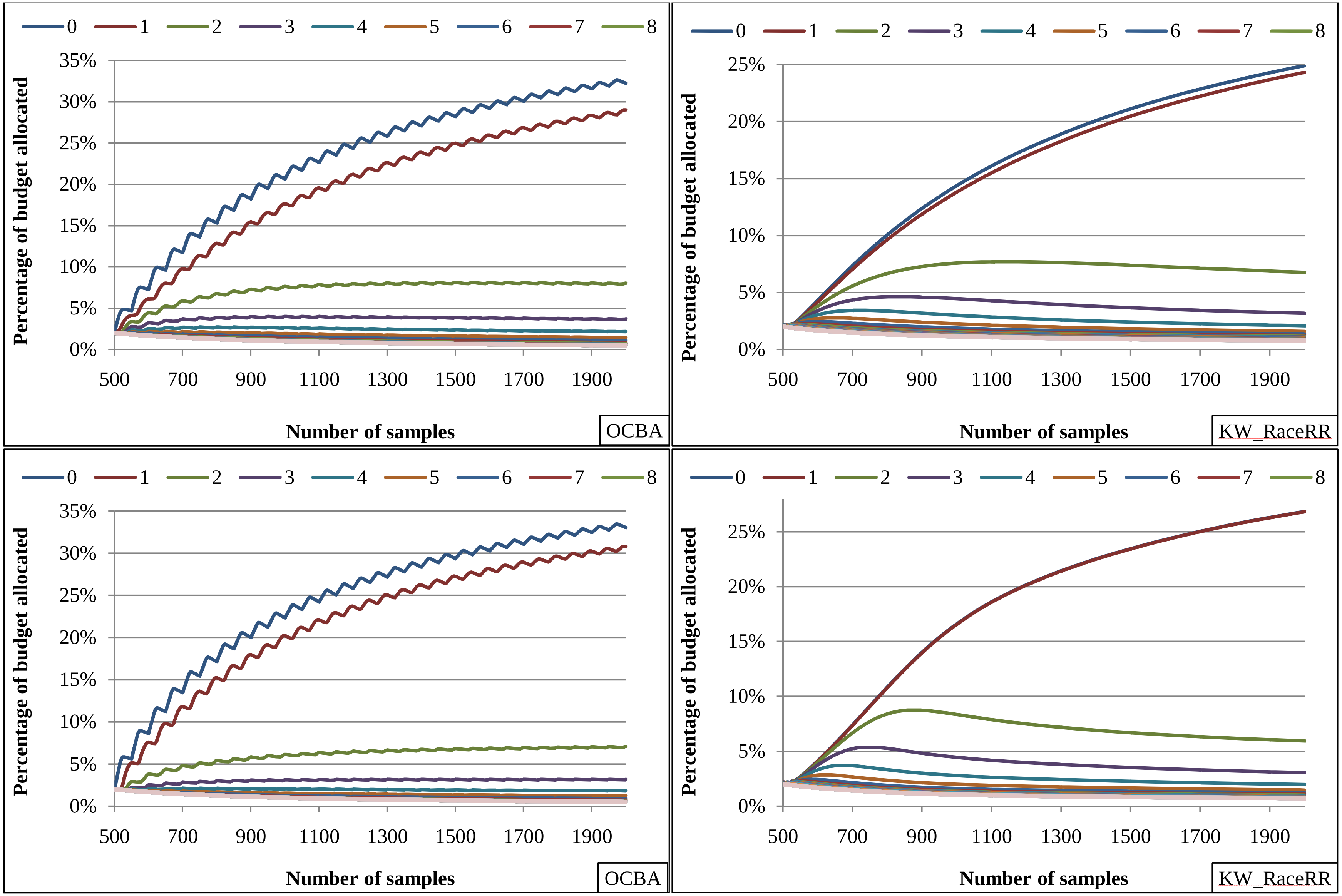


Figure 5.26. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{N}(i, 6^2)\forall i = 0, \dots, k-1$. Correlation = 0.0 (above) 0.9 (below).

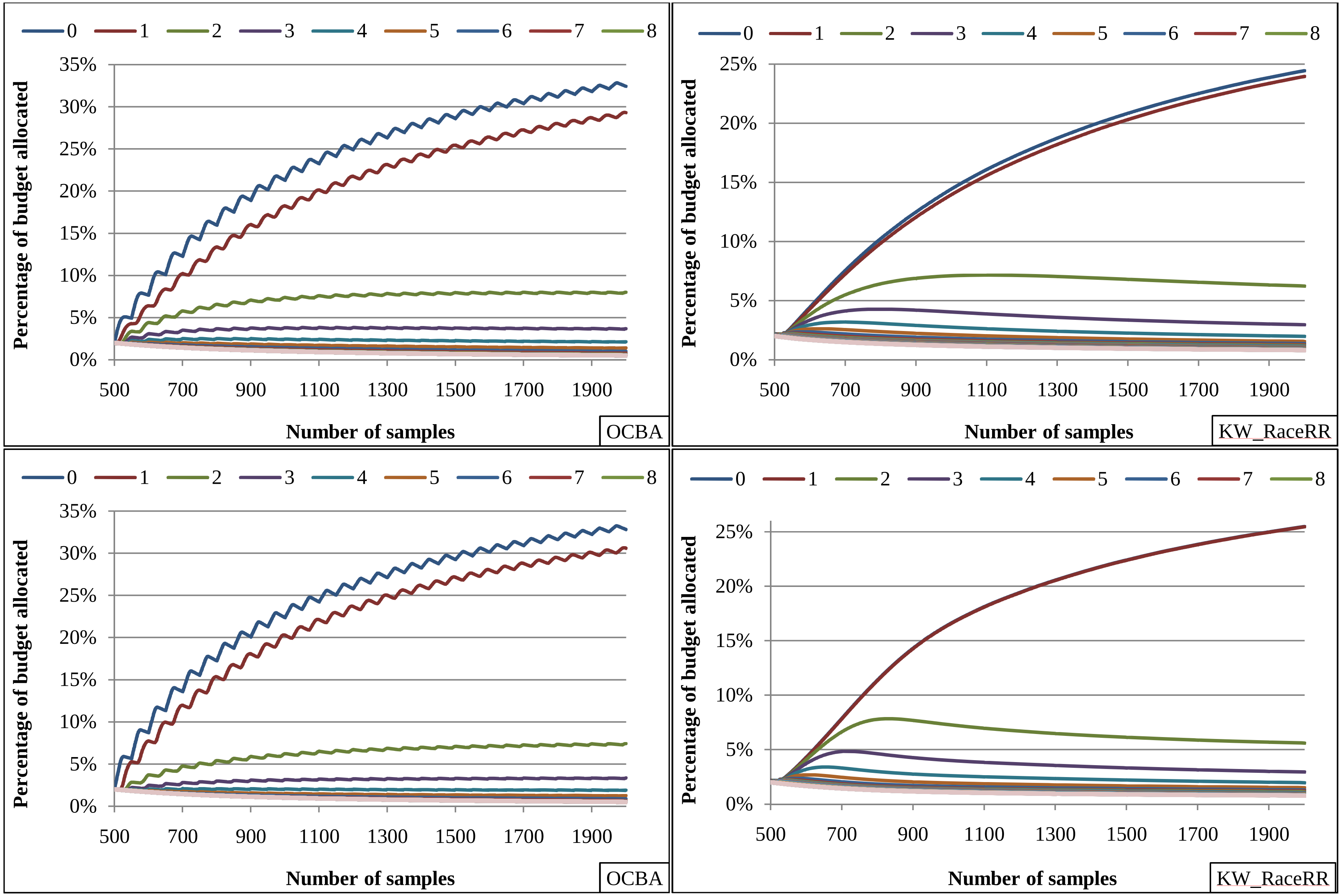


Figure 5.27. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{G}(i, 6^2)\forall i = 0, \ldots, k-1$. Correlation = 0.0 (above) 0.9 (below).

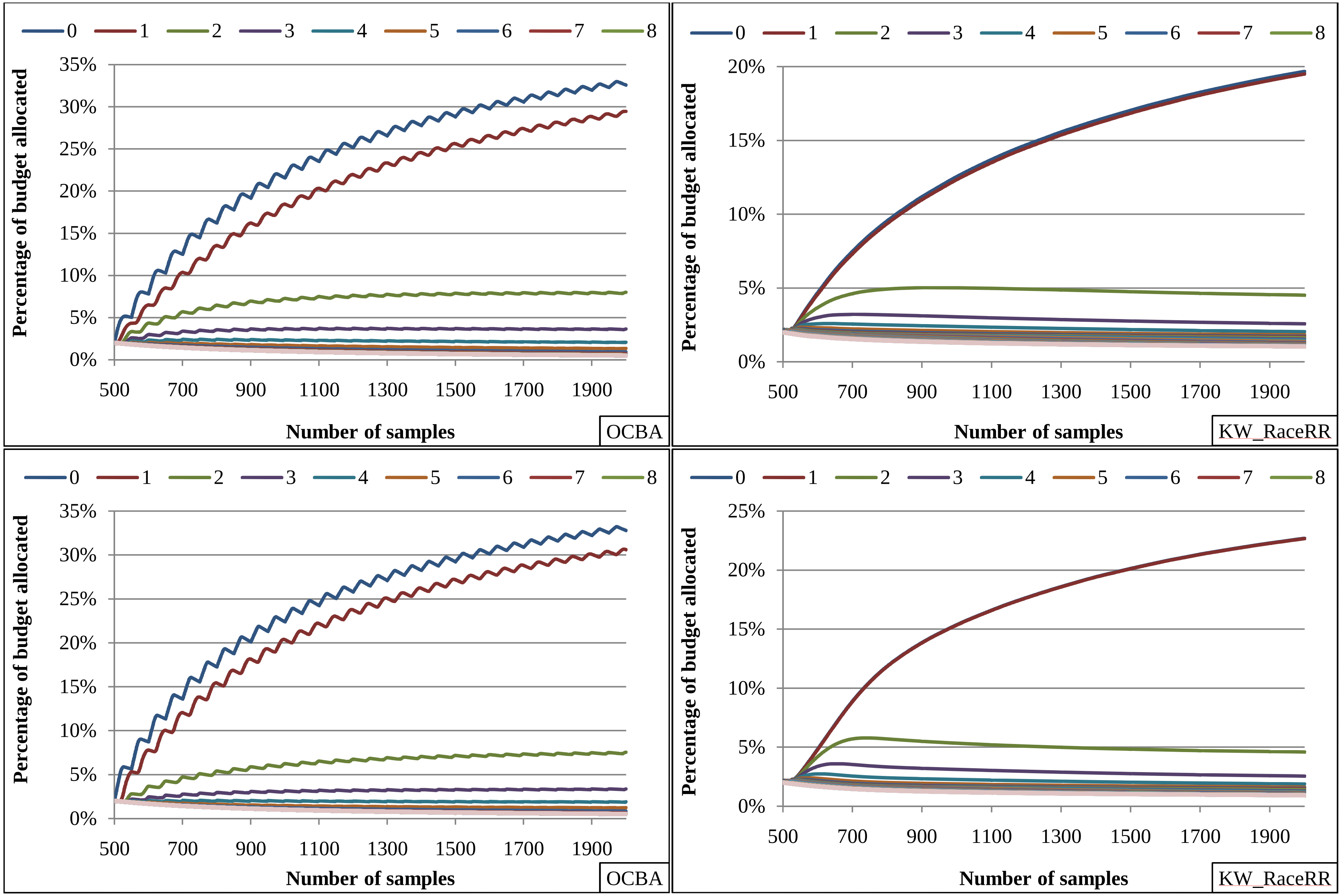


Figure 5.28. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{W}(i, 6^2)\forall i = 0, \dots, k-1$. Correlation = 0.0 (above) 0.9 (below).

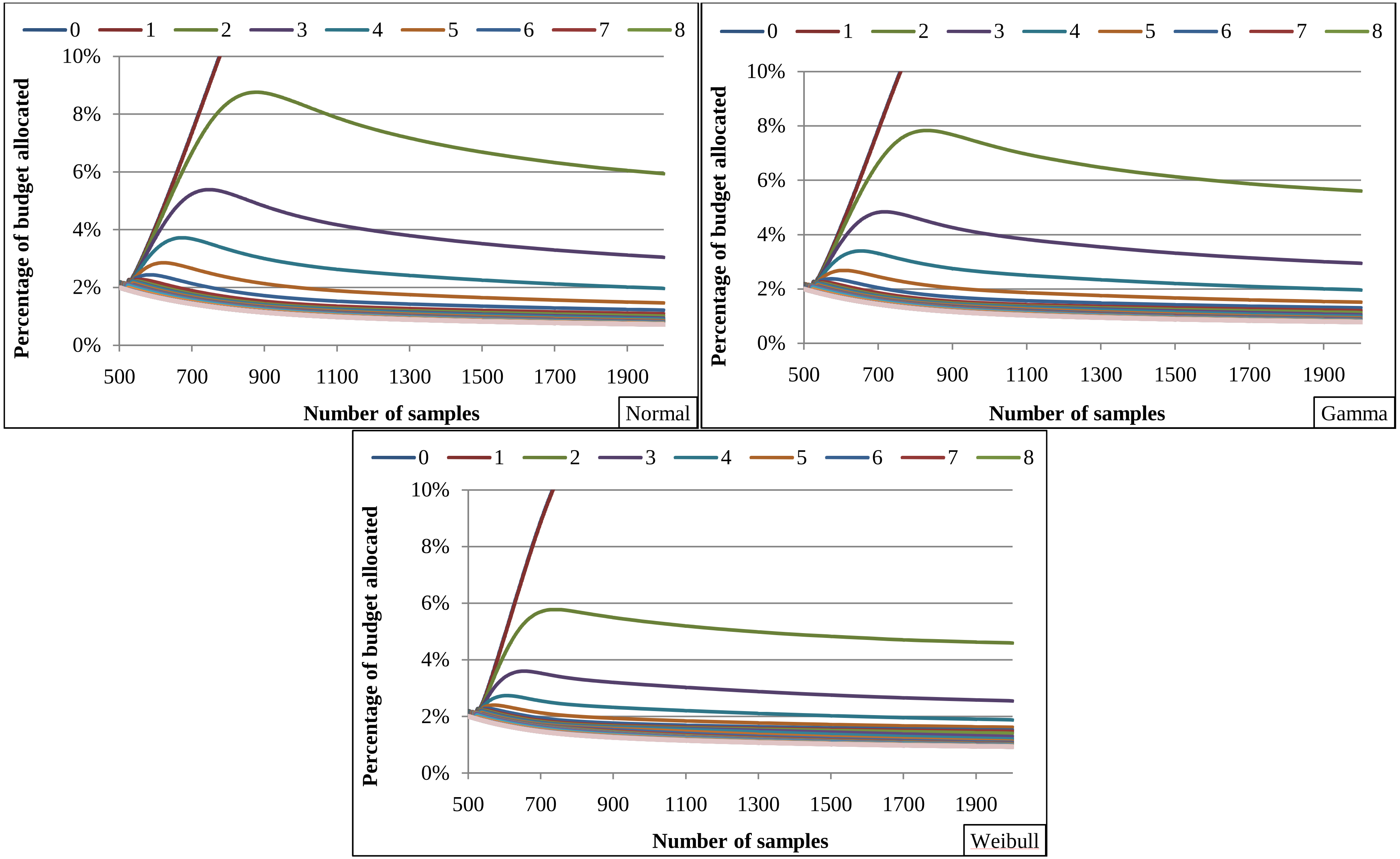


Figure 5.29. Cumulative budget allocated to each system by *KW*-RaceRR throughout the run. Correlation = 0.9. Compare the budget allocated to systems 2-5 across distributions.

5.3.3.2 *Case 2: Monotonically increasing means with linearly decreasing variances*

This Case and Case 3 are of importance, as they test if OCBA will still gain as much benefit by utilizing σ_i/σ_j if the variances are *linearly* changing, as opposed to the exponential change tested in *Set*-1. Figure 5.30 shows that, under 0.0 correlation, OCBA still outperforms KW-RaceRR with Normal and Gamma data, but not with Weibull data. Also, the difference in $PICS$ is much smaller here compared to the correspond case in *Set*-1. Under 0.9 correlation, both perform closely to each other, with KW-RaceRR reaching smaller $PICS$ by the end of the run. A finding consistent with the same case in *Set*-1.

These results indicate that OCBA does not benefit as much from utilizing σ_i/σ_j when the variances are linearly decreasing, because σ_i/σ_j is smaller now, and the corresponding ratios for systems i and j are smaller (especially that $\delta_{b,i}/\delta_{b,j}$ remains the same). Hence, the best systems are separated at a slower rate, compared to what was observed in *Set*-1. See the survival plots in Figures 5.31-5.33.

Relating the observed performance to the survival plots, OCBA is faster in discarding the third best with Normal and Gamma data, compared to Weibull data if correlation is 0.0. Meaning, a typical algorithm should not waste sampling effort system 2, and should focus on 0 and 1 as early as possible. Under 0.9 correlation, the situation is reversed, as the third best becomes critical and should not be discarded early on as OCBA does. Notice how KW-RaceRR keeps it in play throughout the run if the data is Normal, and starts discarding it half way through if the data follows a Weibull distribution. Indicating it is more sensitive to the distribution type than OCBA.

In conclusion, OCBA does not benefit as much, in terms of performance, from σ_i/σ_j when the variances are linearly decreasing, and KW-RaceRR becomes a close competitor. Their overall allocations and performance ranking, however, remain similar to *Set*-1 experiments. Under high correlation it is apparent that OCBA is more robust to the distribution type.

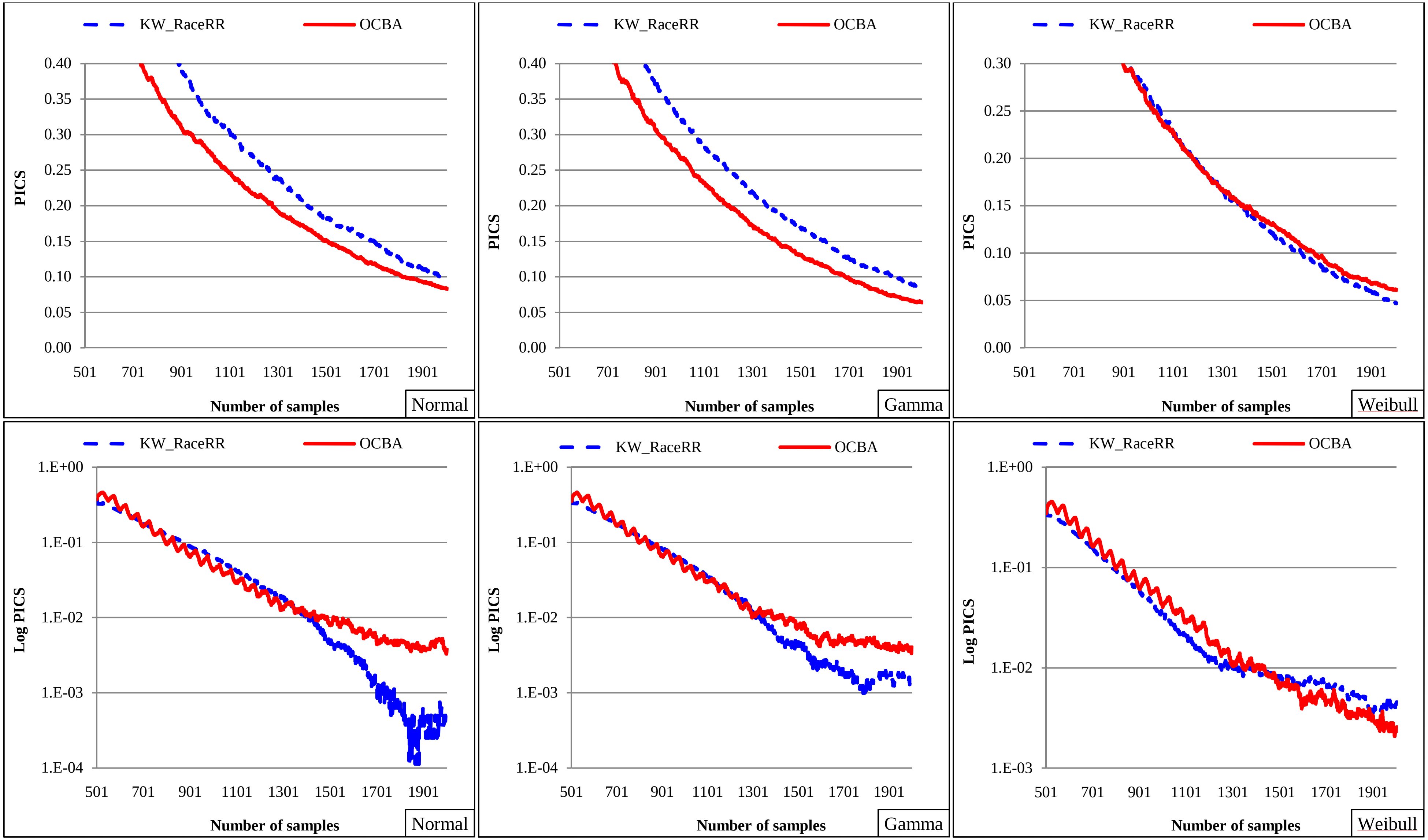


Figure 5.30. Comparison of different algorithms based on *PICS* under 0.0 (above) and 0.9 (below) correlation. $\sim\mathcal{D}\big(i,3(k-i)\big)\forall i=0,\dots,k-1$.

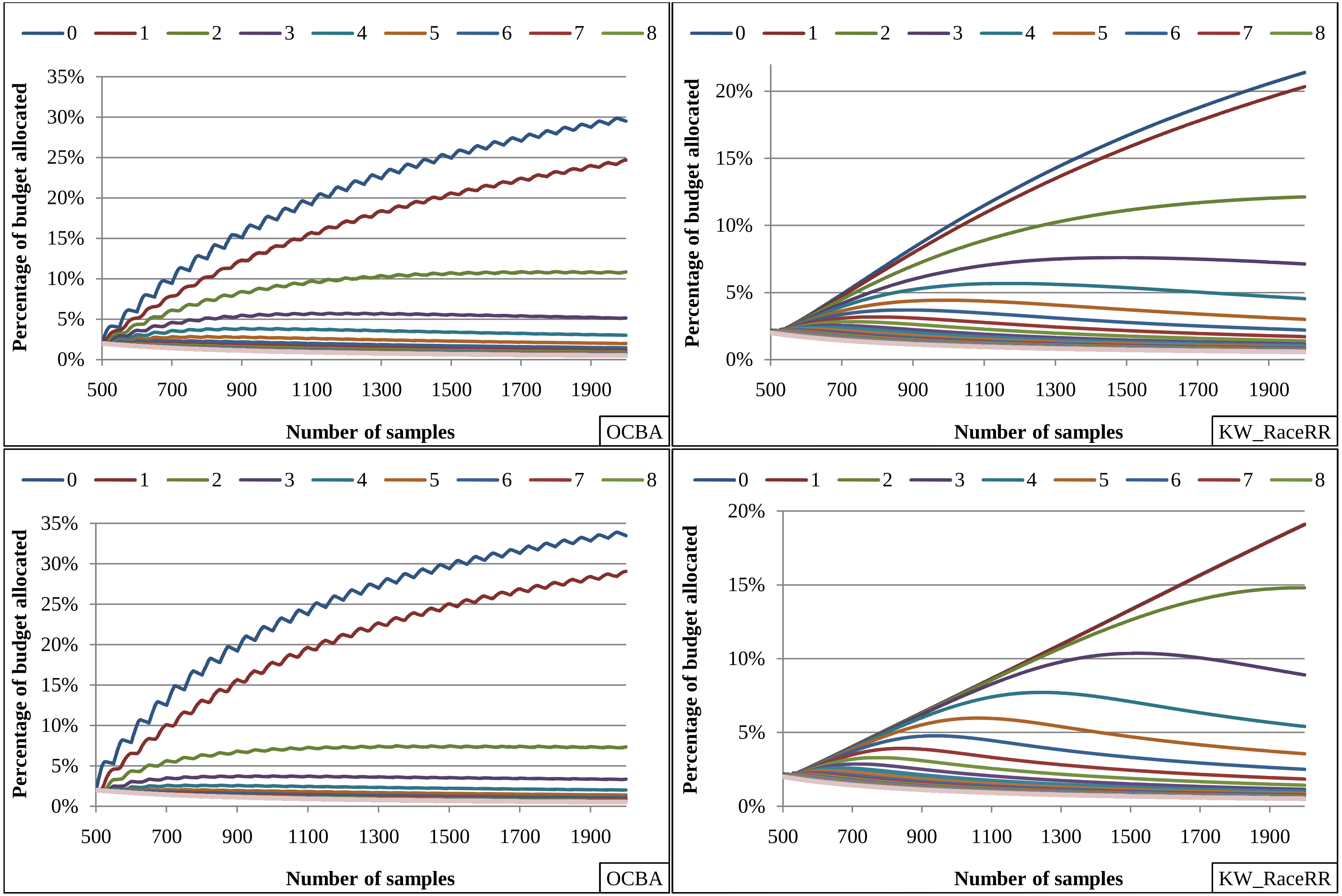


Figure 5.31. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{N}\big(i, 3(k-i)\big)\forall i = 0, \dots, k-1$. Correlation 0.0 (above) 0.9 (below).

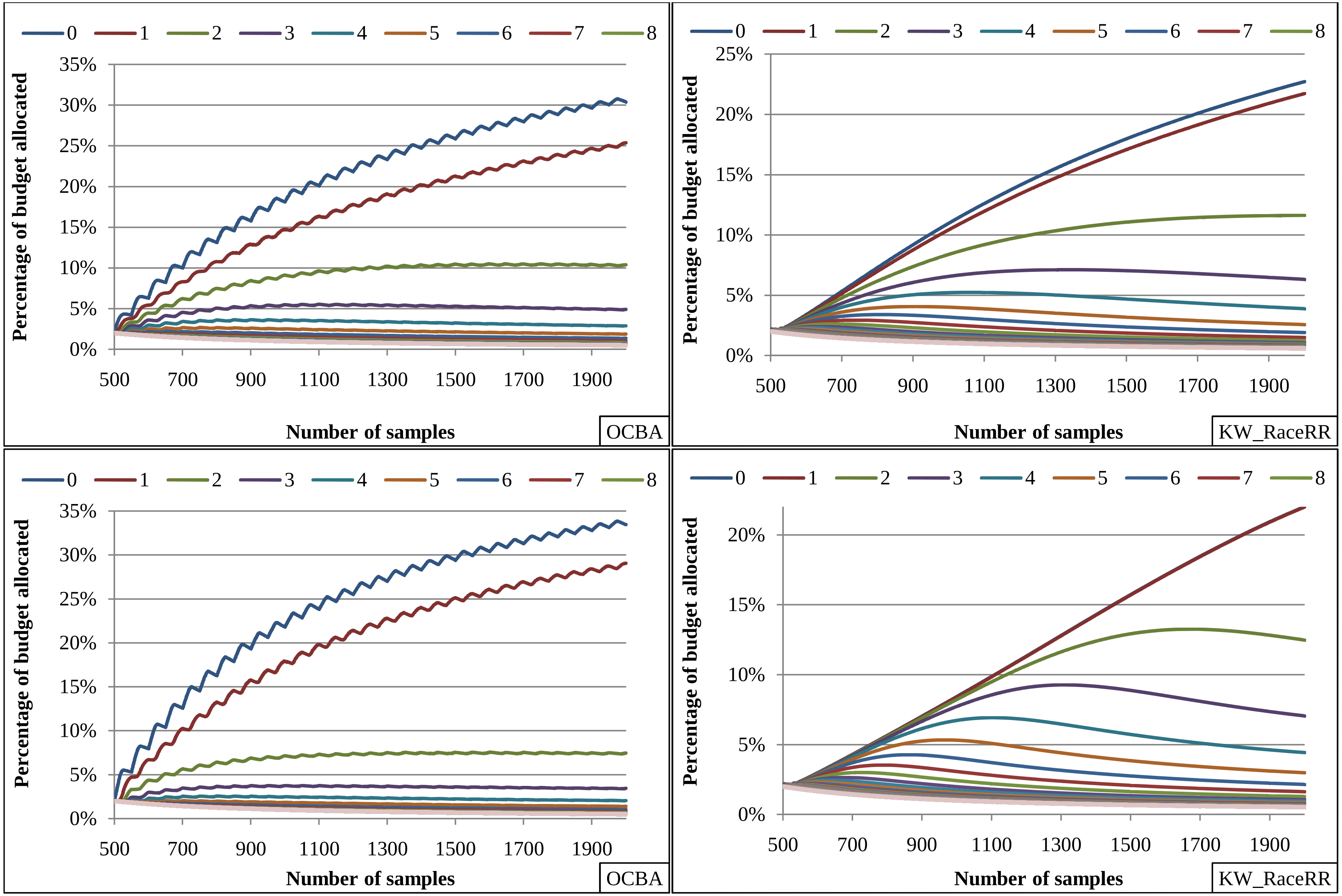


Figure 5.32. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{G}\big(i,3(k-i)\big)\forall i=0,\dots,k-1$. Correlation 0.0 (above) 0.9 (below).

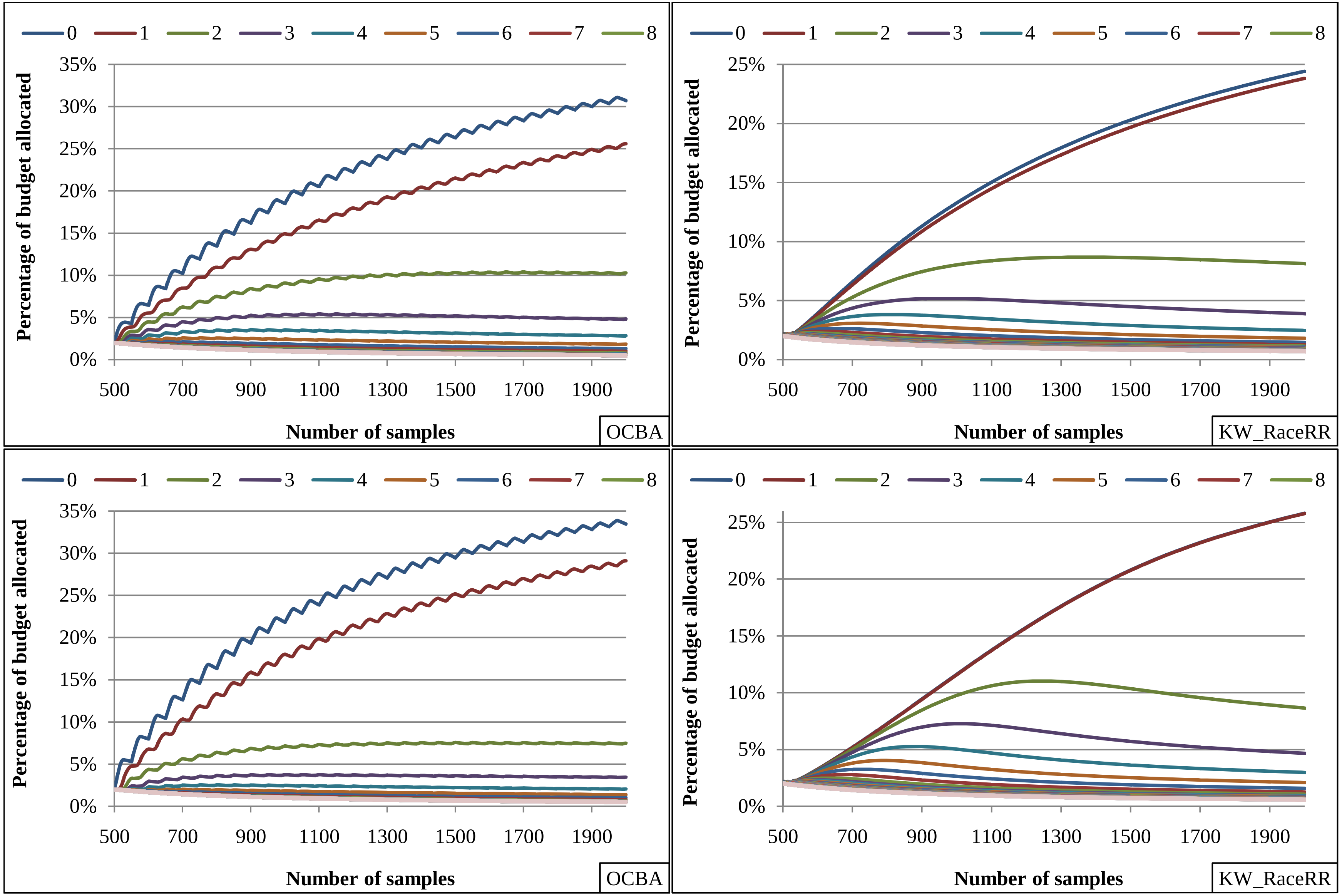


Figure 5.33. Cumulative budget allocated to each system throughout the run. ~$\mathcal{W}\big(i, 3(k-i)\big)\forall i = 0, \dots, k-1$. Correlation 0.0 (above) 0.9 (below).

5.3.3.3 *Case* 3*: Monotonically increasing means with linearly increasing variances*

Similar to what was observed in the corresponding case of *Set*-1, with no correlation, OCBA still outperforms KW-RaceRR regardless of the distribution type and the number of systems. See Figure 5.34. Nonetheless, it is clear that the difference in performance is much smaller here, as the variances are *linearly* increasing, and beyond sample 750 both algorithms reach almost a zero $PICS$. Additionally, under high correlation, KW-RaceRR achieves slightly better results at the beginning, which never occurred in *Set*-1 before. All of this shows how OCBA benefits much less from σ_i/σ_j if the variances are *linearly* increasing.

Interestingly, OCBA is not shifting the majority of the budget to the inferior systems as was observed in *Set*-1. In fact, systems 6-49 are almost never sampled after n_0. See Table 5.25 and Figure 5.35. This means a linearly increasing set of variances does not raise the ratios of the inferior systems high enough to allocate them more samples. The effect of the number of systems seems minor as systems 6-9, which were allocated a major part of the budget in *Set*-1, now receive almost nothing.

Table 5.25. Percentage of the budget allocated to each system by OCBA at the end of the run. Data $\sim\mathcal{D}(i,(i+1)^4)\forall i=0,\dots,k-1$.

System	0.0 correlation			0.9 correlation		
	Normal	Gamma	Weibull	Normal	Gamma	Weibull
0	21%	21%	21%	21%	21%	21%
1	25%	25%	26%	26%	26%	26%
2	9%	9%	10%	9%	10%	10%
3	5%	6%	6%	5%	6%	6%
4	4%	4%	4%	4%	4%	4%
5	3%	3%	3%	3%	3%	3%
6-49	2%	2%	2%	2%	2%	2%

The survival plots, in Figures 5.36-5.38, show that OCBA is faster in separating between the best systems and inferior ones, compared to KW-RaceRR. Plus, the best systems are allocated more of the budget. Such a distribution works for OCBA's advantage if correlation is 0.0, but not if it is high, because the samples taken from each system become more consistent, and a better estimate of the their parameters is required for a correct selection. This was observed in *Set*-1 as well.

In conclusion, OCBA does not benefit as much, in terms of performance, from σ_i/σ_j when the variances are linearly increasing, and KW-RaceRR becomes a close competitor. OCBA's overall allocations and performance ranking, do change compared to *Set*-1 experiments. The most noticeable difference is the focus of the budget on the best systems, not the ones with the highest variance. Under high correlation KW-RaceRR performs slightly better at the beginning, and reaches a near zero $PICS$ half way through the run where its performance overlaps with OCBA's. Additionally, it showed to be more sensitive to the distribution type.

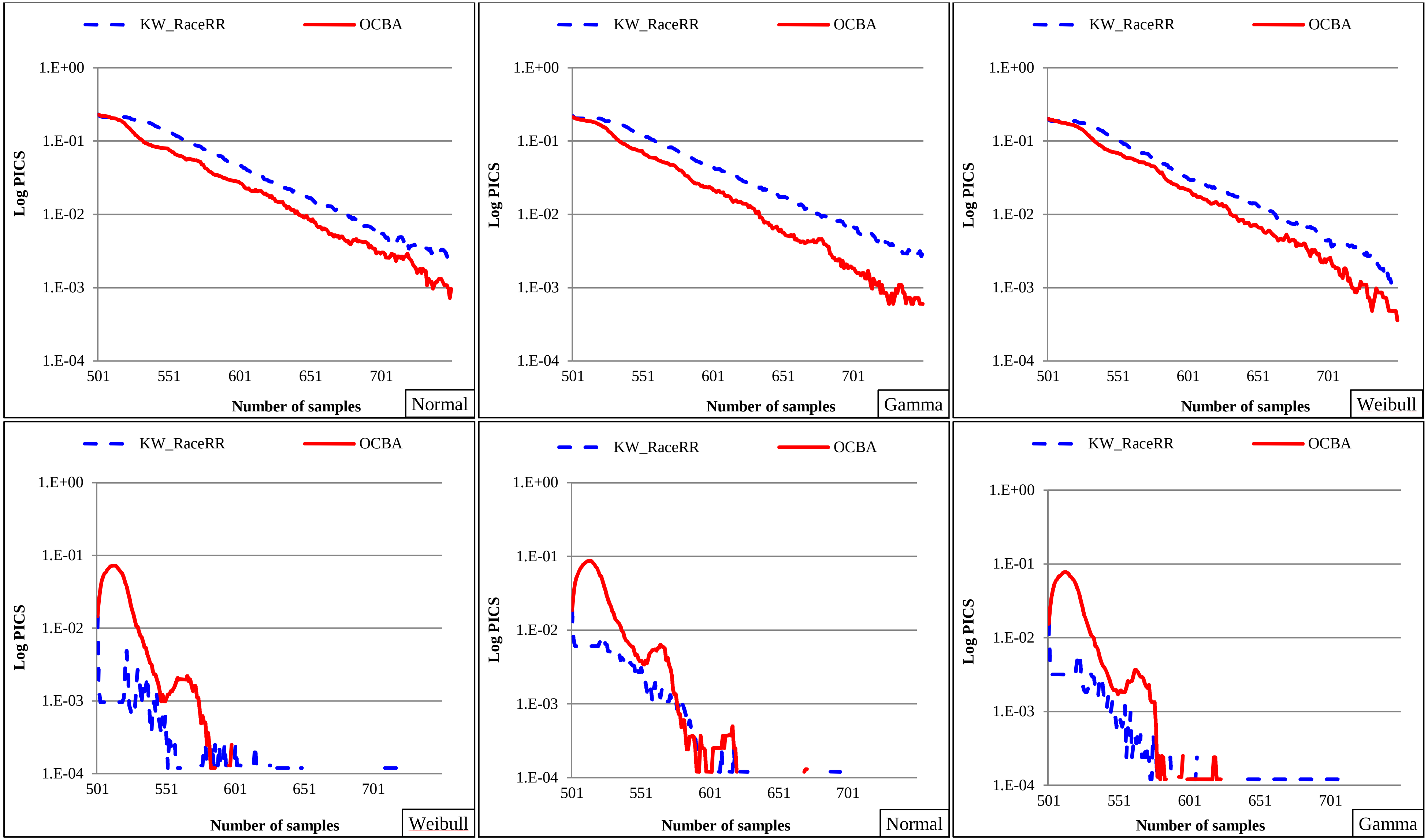


Figure 5.34. Comparison of different algorithms based on *PICS* under 0.0 (above) and 0.9 (below) correlation. $\sim\mathcal{D}\big(i, 4(i+1)\big)\forall i = 0, \dots, k-1$.

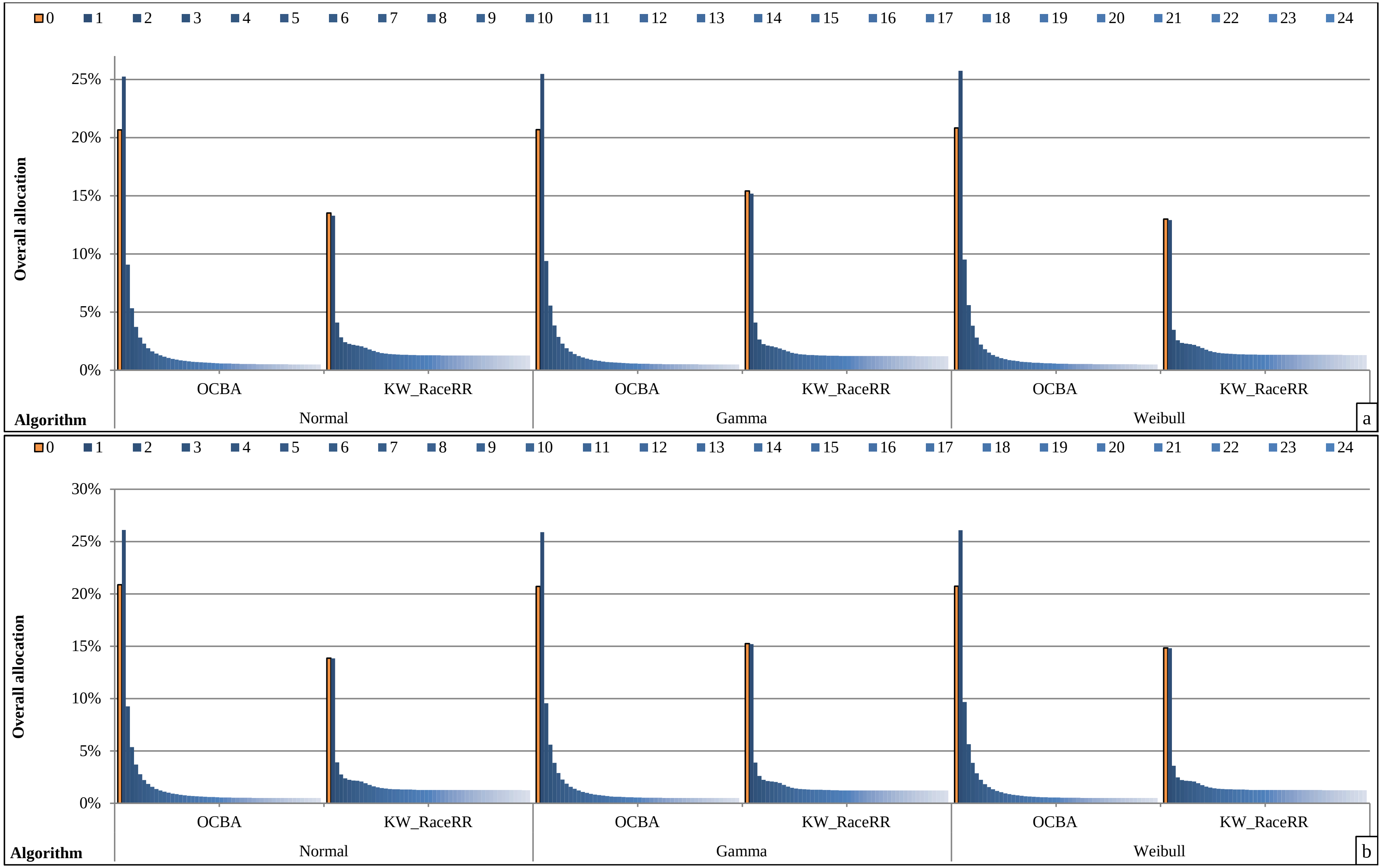


Figure 5.35. Overall allocation of different algorithms at 0.0 (a) and 0.9 (b) correlation. $\sim\mathcal{D}\big(i,4(i+1)\big)\forall i=0,\dots,k-1$. The numbers are averaged over all replications.

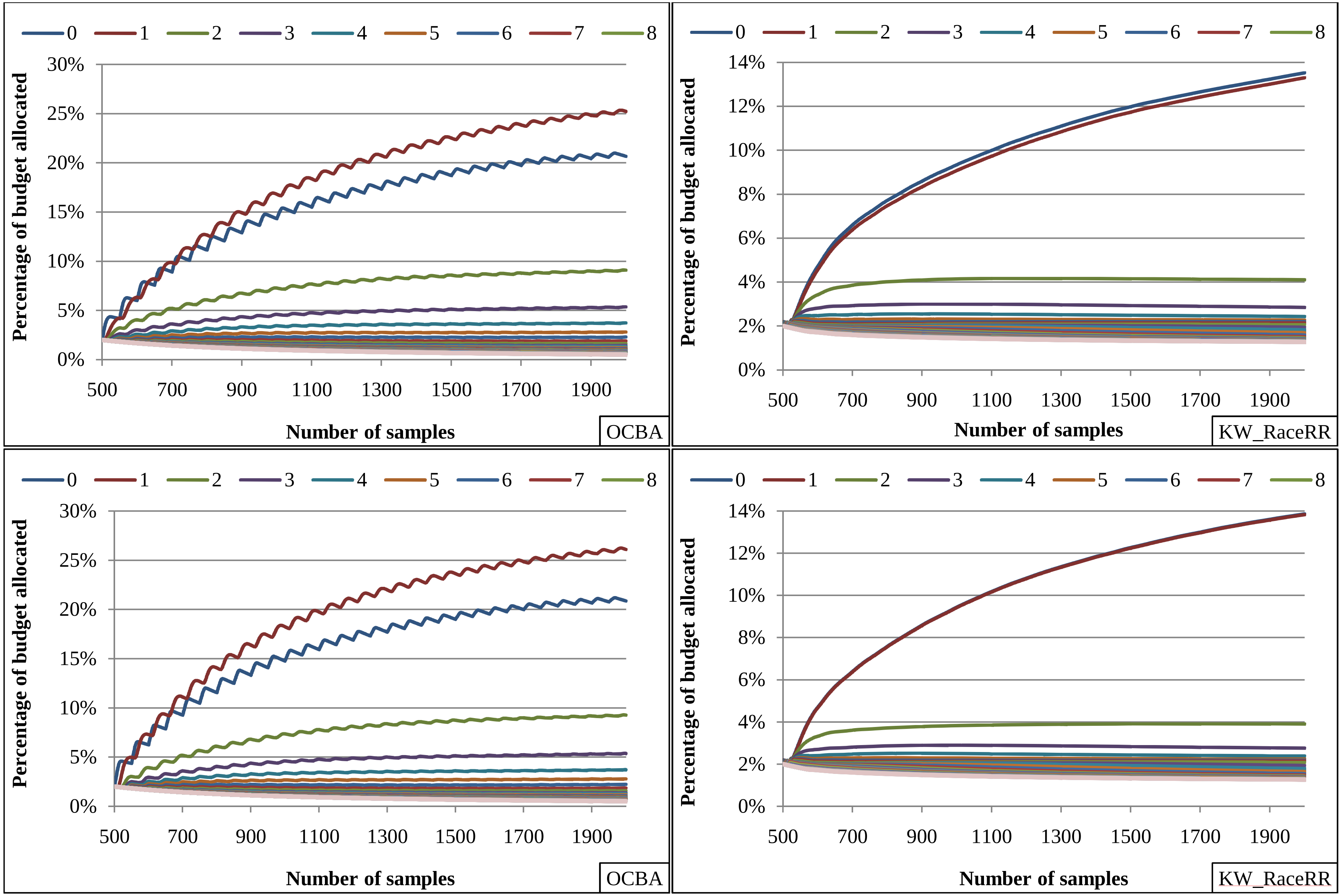


Figure 5.36. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{N}\big(i, 4(i+1)\big)\forall i = 0, \dots, k-1$. Correlation 0.0 (above) 0.9 (below).

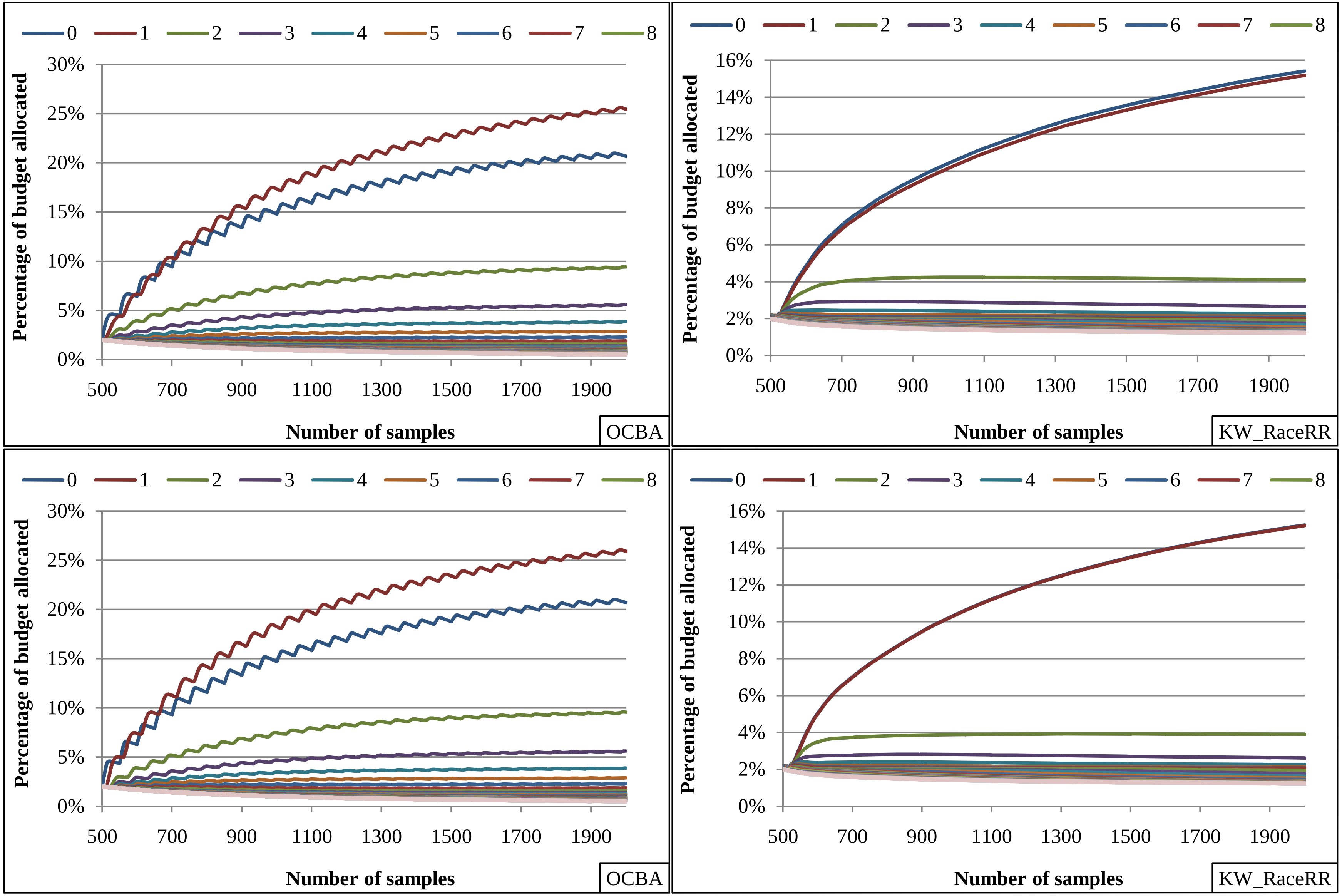


Figure 5.37. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{G}\big(i,4(i+1)\big)\forall i=0,\dots,k-1$. Correlation 0.0 (above) 0.9 (below).

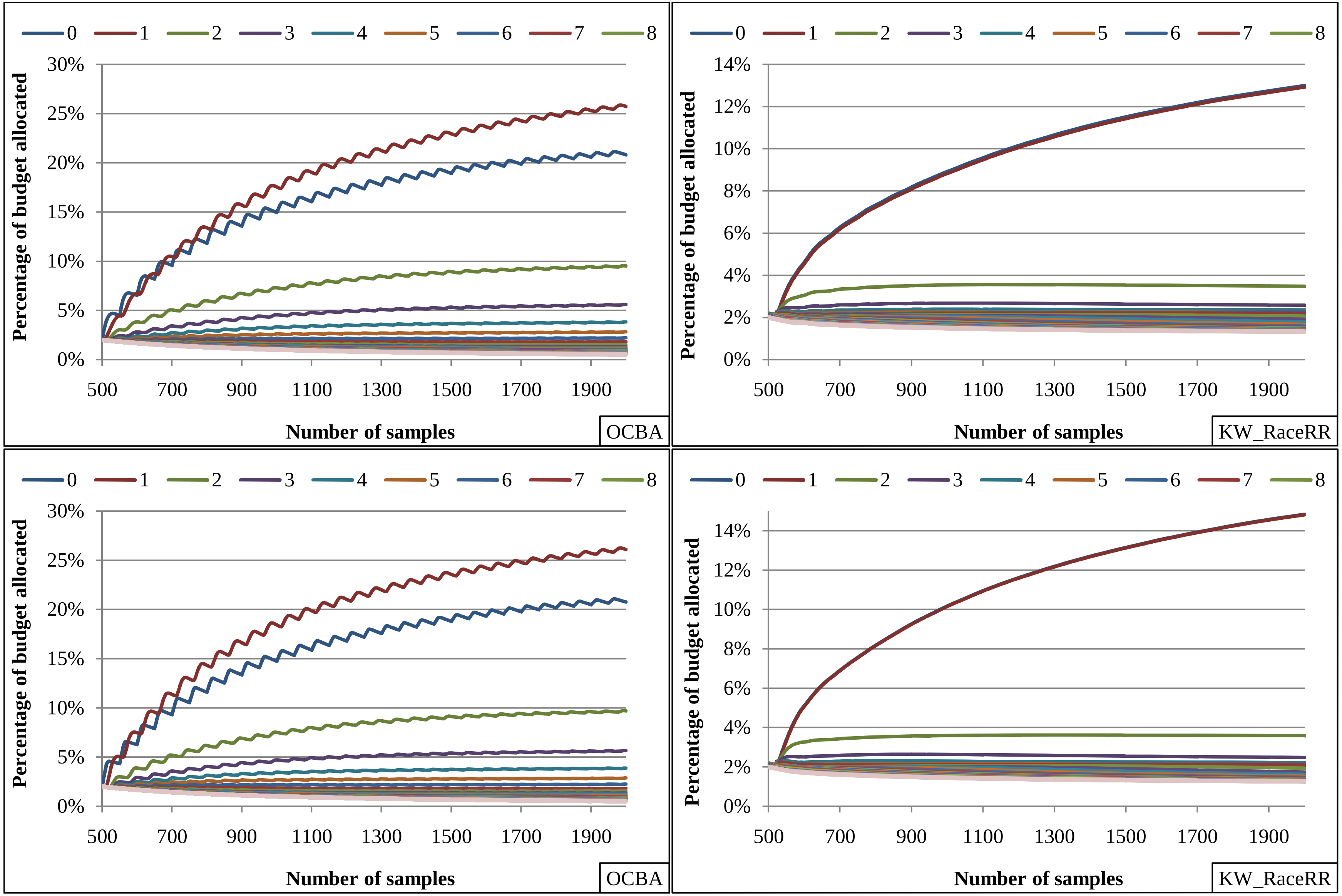


Figure 5.38. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{W}\big(i,4(i+1)\big)\forall i=0,\ldots,k-1$. Correlation 0.0 (above) 0.9 (below).

5.3.3.4 *Case* 4*: Means and variances are randomly selected*

Under 0.0 correlation, OCBA converges faster at the beginning, but slows down half way through the run, where KW-RaceRR starts achieving better results. See Figure 5.39. The corresponding survival plots, Figures 5.40-5.42, show that OCBA is faster in separating the best systems from the inferior ones, and focuses most of the budget on the top 3 or 4 systems. In comparison, KW-RaceRR distributes the budget more evenly between the best systems, and allocates more samples to the inferior ones. While this slows its convergence at the beginning, it allows for better estimates of the distributions' parameters, and eventually leads to better results at the end of the run. Figure 5.43 shows an example for Normal data, the other distributions are similar. The same behavior was observed in *Set*-1.

At 0.9 correlation both algorithms still allocate in a similar fashion to the independent case, which benefits KW-RaceRR more. Recall that under high correlation, the top systems require a more evenly spread budget among them to better estimate their parameters, and make a correct selection. Also note that both algorithms reach a near zero $PICS$ half way through, and their performances are indistinguishable from that point on. KW-RaceRR is once more performing differently *across* distributions. Looking at the allocations of third and fourth best when the distribution is Normal, compared to the Weibull, it is clear that KW-RaceRR starts allocating less samples to these systems earlier with Weibull data. This is linked with the slower convergence, indicating that a better estimate of their parameter is required for correct selection.

In conclusion, the random case is more affected by the increased of the number of systems, compared to *Set*-1, in terms of OCBA performing much better during the early stages of the run. However, as KW-RaceRR invests in more accurately estimating the parameters of the best distributions, it converges faster and outperforms OCBA by the end of the run. Under high correlation, KW-RaceRR is more sensitive to the distribution type as observed from its performance with Weibull data.

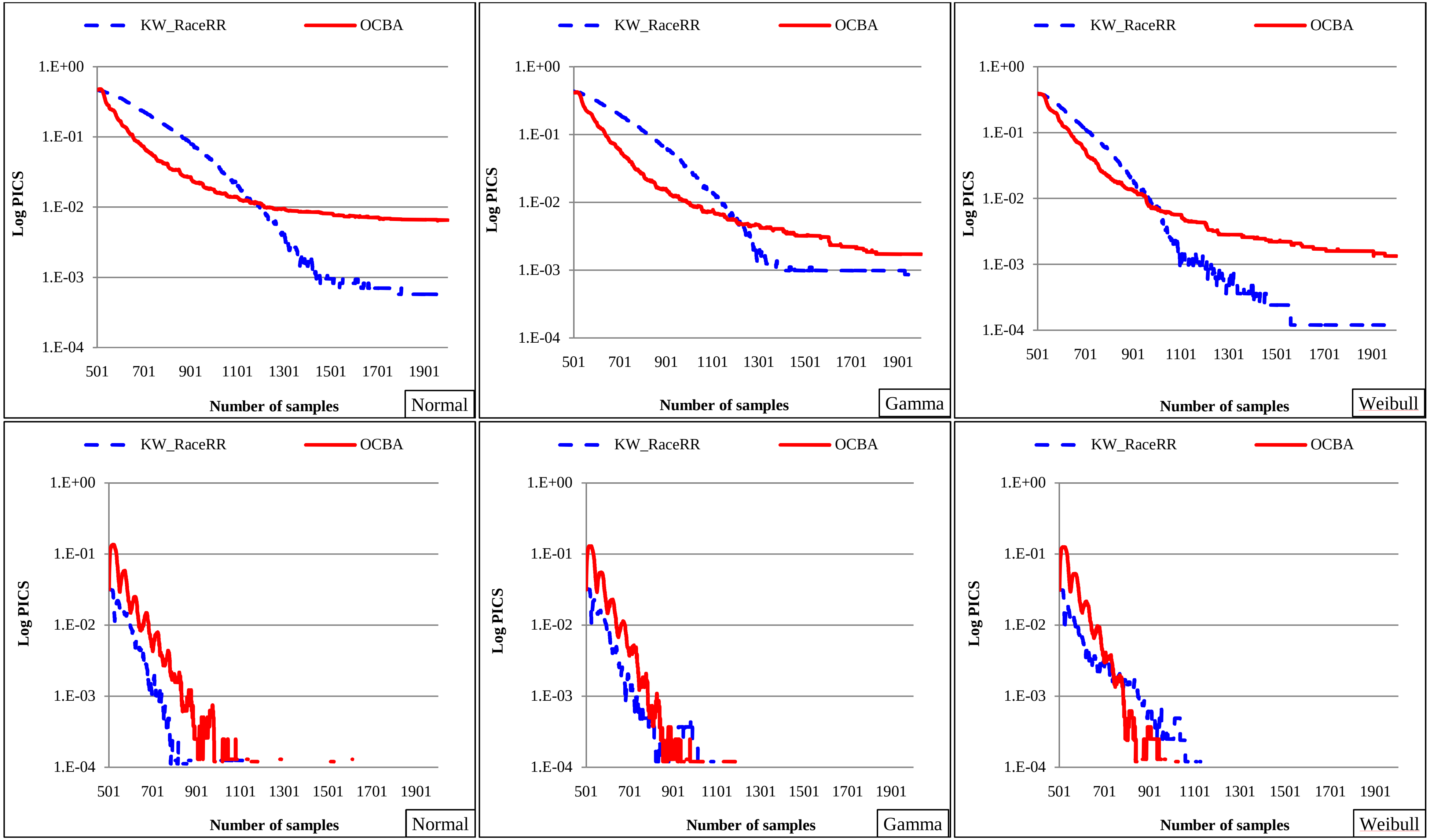


Figure 5.39. Comparison of different algorithms based on $PICS$ under 0.0 (above) and 0.9 (below) correlation. $\sim\mathcal{D}\big(U(0,15), U(20,40)\big)$.

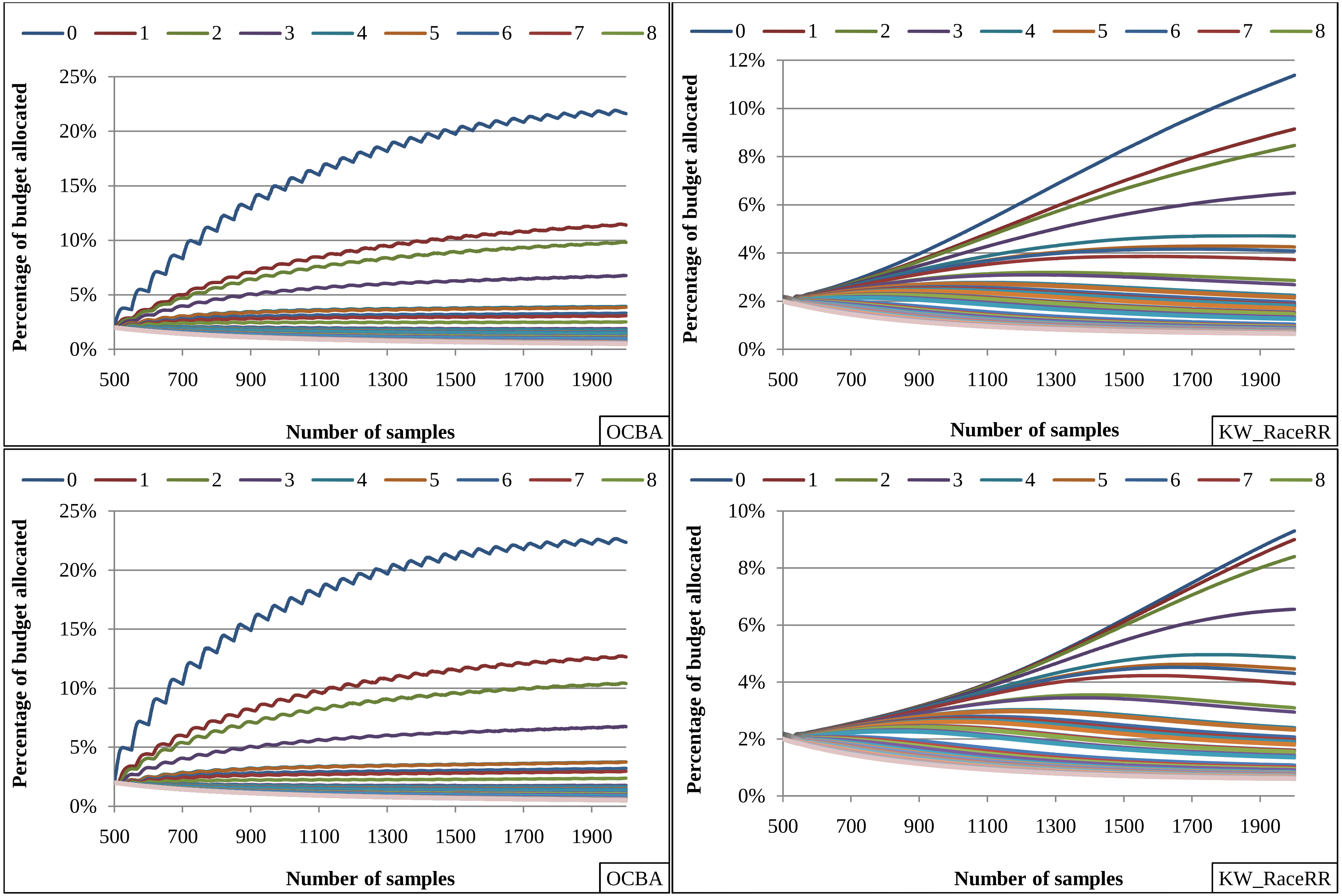


Figure 5.40. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{N}\big(U(0,15),U(20,40)\big)$. Correlation 0.0 (above) 0.9 (below).

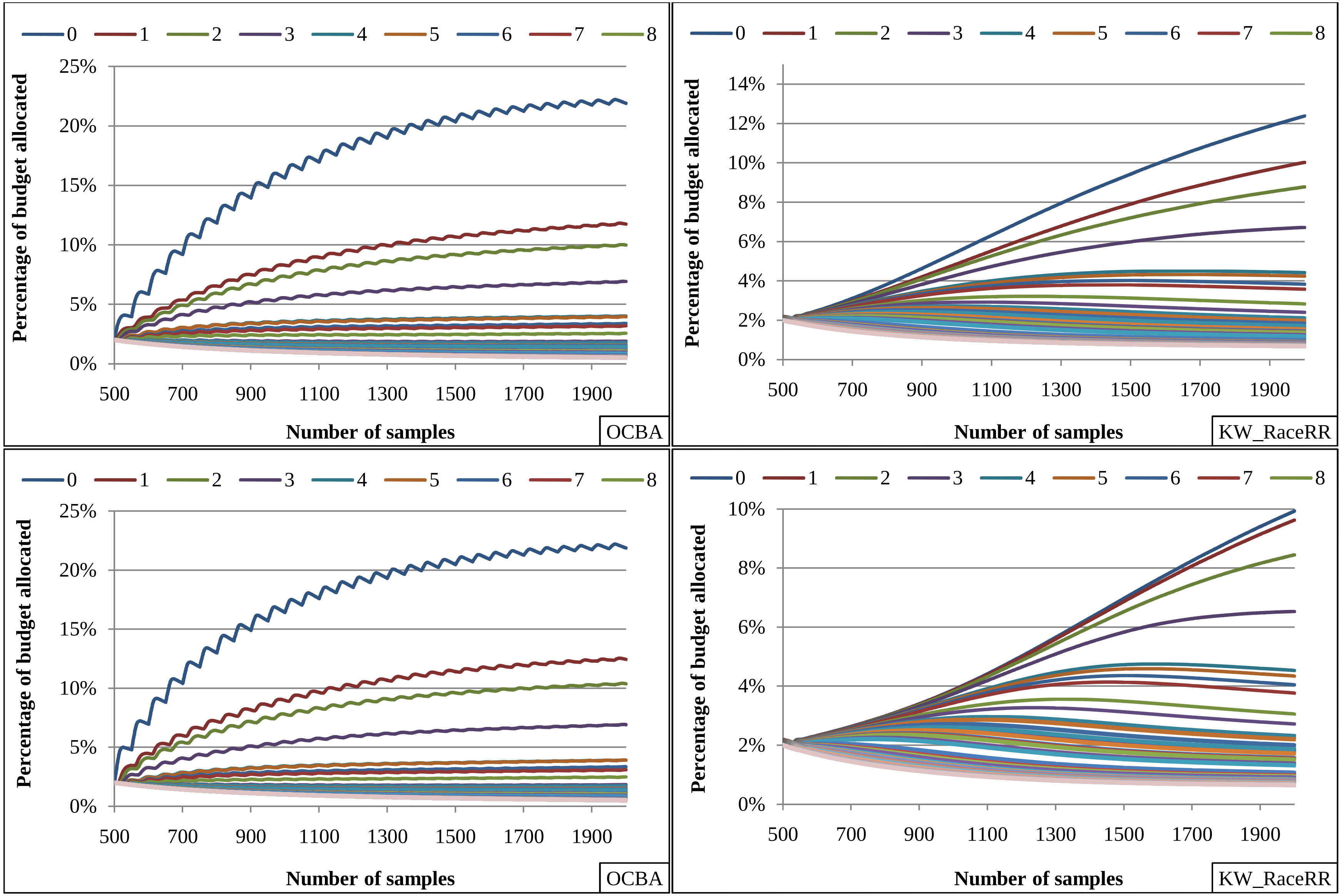


Figure 5.41. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{G}\big(U(0,15), U(20,40)\big)$. Correlation 0.0 (above) 0.9 (below).

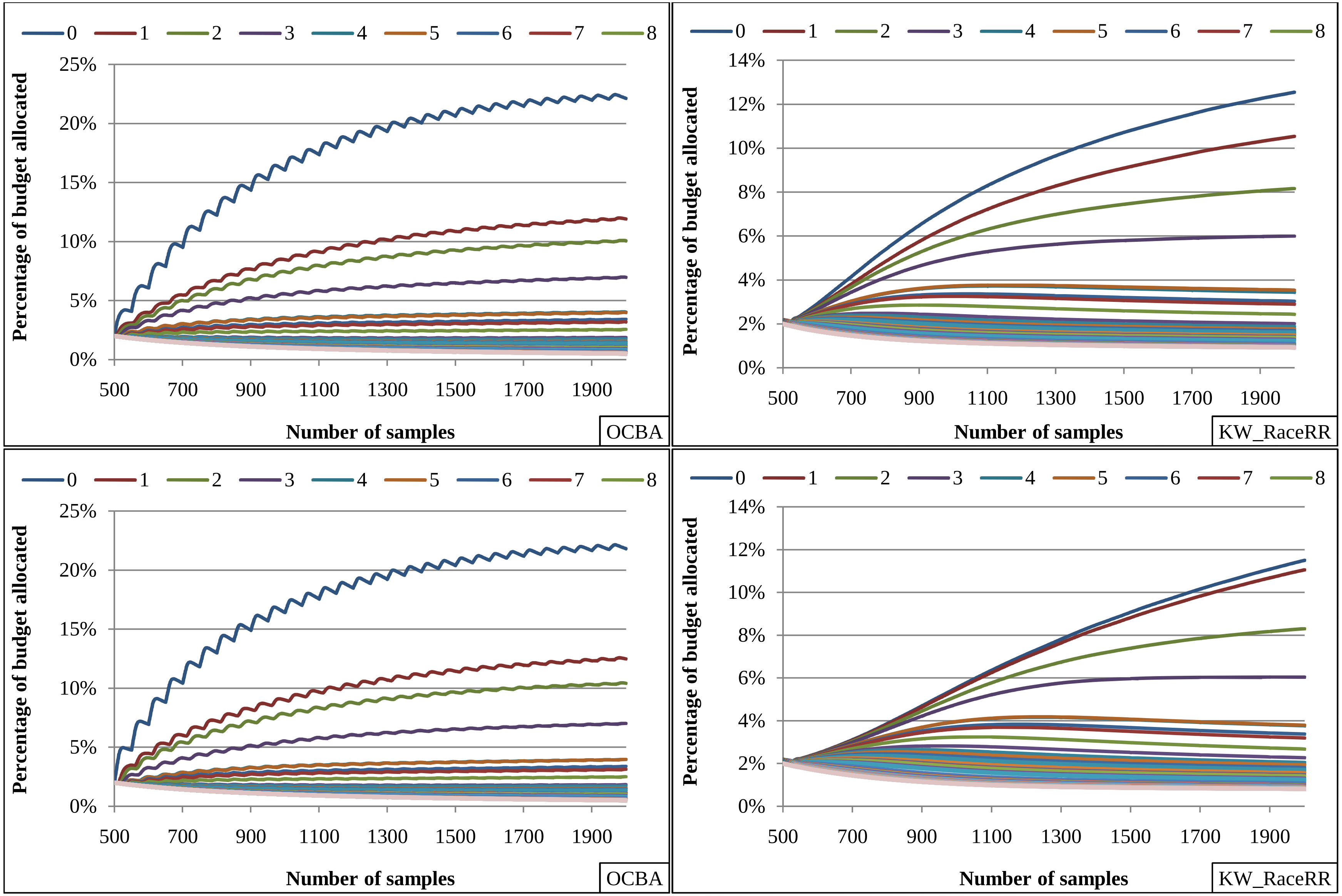


Figure 5.42. Cumulative budget allocated to each system throughout the run. $\sim\mathcal{W}\big(U(0{,}15), U(20{,}40)\big)$. Correlation 0.0 (above) 0.9 (below).

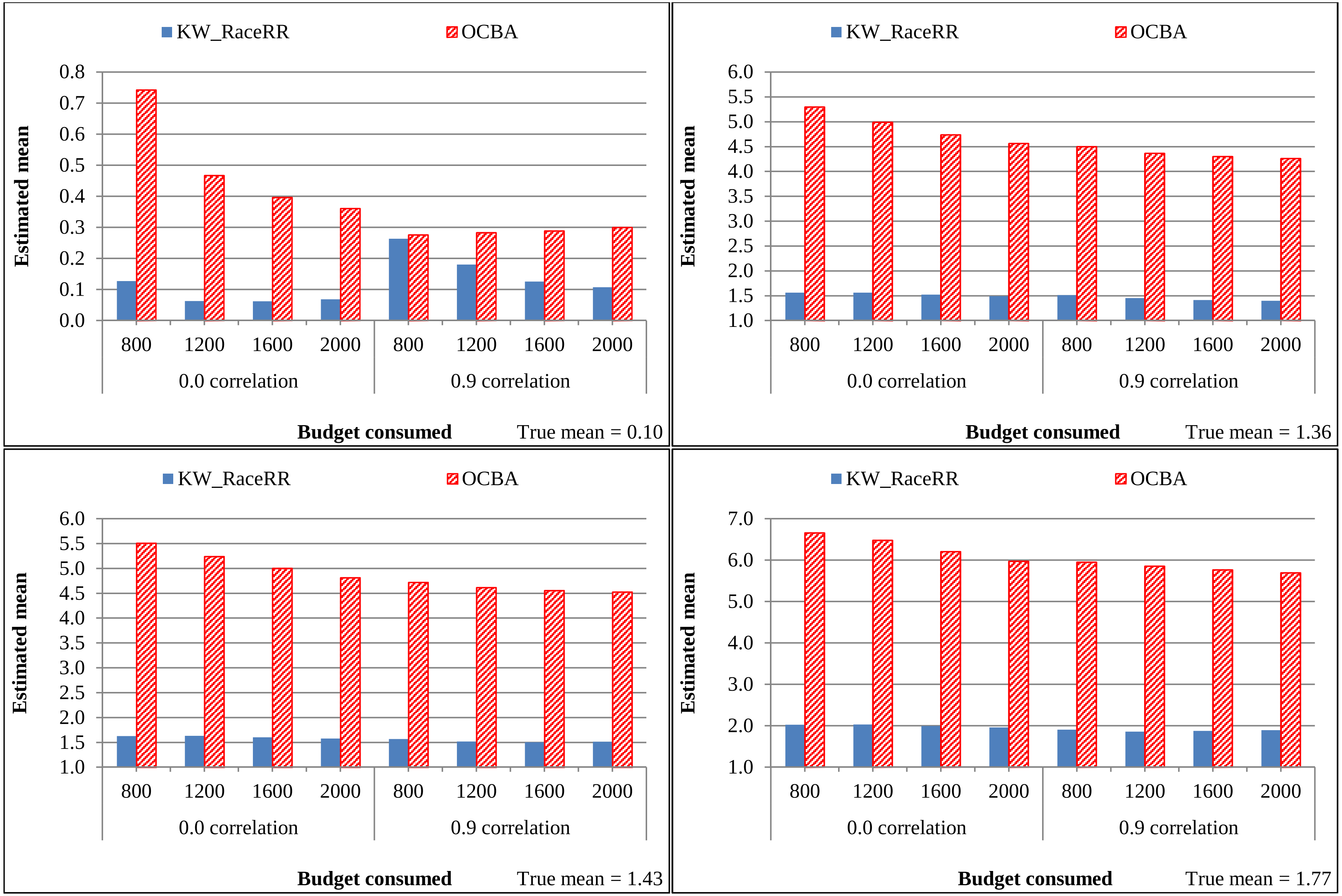


Figure 5.43. Estimated mean of the best four systems under 0.0 correlation at selected points throughout the run. $\sim\mathcal{N}\big(U(0{,}15), U(20{,}40)\big)$.

5.3.3.5 *Summary*

In comparison to *Set*-1, increasing the number of systems and/or changing the distribution type has minor effect on the allocations made by KW-RaceRR and OCBA. The exception is that of the linearly increasing variances, where OCBA did not allocate the majority of the budget to the inferior systems (largest variance), as their ratios were not high enough. Instead it allocated the majority to the best ones, similar to KW-RaceRR.

In terms of the $PICS$, KW-RaceRR is still superior if the budget is high, and OCBA is better for lower budgets. This applies for the equal variances case and the random case, with no correlation. Under high correlation, the distribution type (Weibull in specific) affects KW-RaceRR by slowing its convergence roughly after 700 samples, although it reaches a near zero $PICS$ afterwards. OCBA seems to be more robust against the distribution type.

The cases of *linearly* increasing or decreasing variances are of importance, as they show how little OCBA benefits from utilizing σ_i/σ_j, and its performance becomes much closer to KW-RaceRR, compared to *Set*-1. In addition, if correlation is high and the variances are increasing, KW-RaceRR performed better with a low budget mainly due to the more evenly distribution of the budget among the systems. See Table 5.26.

Table 5.26. A summary of the best performing algorithms (based on *PICS*) in *Set*-3 experiments.

Case	Means and variances	Budget	Distribution type	Correlation	Best algorithm
1	$\sim\mathcal{D}(i,6^2)\forall i=0,\dots,k-1$	≥ 1500	$\mathcal{N},\mathcal{G},\mathcal{W}$[4]	0.0	KW-RaceRR
		≤ 1200	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.0	OCBA
		≤ 700	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	KW-RaceRR
		≥ 1500	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	Indistinguishable
2	$\sim\mathcal{D}\big(i,3(k-i)\big)\forall i=0,\dots,k-1$	Entire	$\mathcal{N},\mathcal{G}$	0.0	OCBA
		≥ 1500	$\mathcal{W}$	0.0	KW-RaceRR
		≥ 1500	$\mathcal{N},\mathcal{G}$	0.9	KW-RaceRR
		Entire	$\mathcal{W}$	0.9	Indistinguishable
3	$\sim\mathcal{D}\big(i,4(i+1)\big)\forall i=0,\dots,k-1$	Entire	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.0	OCBA
		≤ 500	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	KW-RaceRR
		≥ 700	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	Indistinguishable
4	$\sim\mathcal{D}\big(U(0,15),U(20,40)\big)$	≤ 1200	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.0	OCBA
		≥ 1500	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.0	KW-RaceRR
		≤ 700	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	KW-RaceRR
		≥ 1000	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	Indistinguishable

[4] $\mathcal{N} = Normal\ distribution,\ \mathcal{G} = Gamma\ distribution, \mathcal{W} = Weibull\ distribution$

5.4 Conclusion

This chapter compared OCBA, CBA, and EBA to four Racing algorithms, two based on a 1-way ANOVA test: A-Race_1WUB and KW-Race, and two based on a 2-way ANOVA test: A-Race_2Way and F-Race. KW-Race and A-Race_2Way were implemented with and without the reset idea. The chosen performance measures were $PICS$ and $\mathbb{E}[OC]$, calculated experimentally over 100,000 replications. Using such measures required exact knowledge of the parameter settings' performance distribution, which is not feasible and depends on the problem domain and optimization algorithm. Hence, they were simulated by probability distributions (systems) with pre-determined means, variances, and correlations.

Three sets of experiments were conducted: *Set*-1 had a fixed training/sampling budget, which meant that Racing algorithms had to use the reset idea to consume the whole budget. *Set*-2 had A-Race_2Way and KW-Race specify the budget for OCBA, CBA, and EBA, which meant that Racing algorithms were run without the reset idea. The main objectives of *Set*-2 were to see if allowing a Racing algorithm to specify the budget gives it an advantage, and to show how the reset idea adapts the significance level and improves on standard Racing. Finally, *Set*-3 ran the same experiments as *Set*-1, but used non-normal distributions and an increased the number of systems.

Results for *Set*-1 revealed that the best performing algorithms overall are OCBA and KW-RaceRR, even with correlated data. Though if the budget is low (roughly ≤ 500) and correlation is 0.9, 2-way Racing is better. CBA was not competitive due to its inability to accurately estimate a covariance matrix, which eventually degraded its performance. OCBA is superior to KW-RaceRR when the variances are exponentially increasing or decreasing, because OCBA makes use of the ratio between σ_i/σ_j when distributing the budget. KW-RaceRR, on the other hand, performs better than OCBA on all other cases, if the budget is high (roughly ≥ 1000). It invests more samples to better estimate the parameters of the systems closest to the best. While this slows its convergence at the beginning, it enables it to outperform OCBA later on.

Results for *Set*-2 showed that *KW*-Race and *A*-Race_2Way (the algorithms setting the budget) perform better\worse than OCBA under the same conditions in *Set*-1. Indicating that allowing Racing to specify the budget does not change its performance ranking among OCBA, CBA, and EBA. In addition, it was shown how Racing with reset improves over standard Racing algorithms by adapting the α_0 parameter online. Such adaptation allowed Racing with reset to also have a stable performance over a wide range of α_0 values.

Results for *Set*-3 were mainly concerned with the robustness of the best performing algorithms of *Set*-1 against normality and the number of systems. OCBA and *KW*-RaceRR (the only two algorithms tested in *Set*-3) displayed similar allocation patterns to *Set*-1. The same holds, generally, for their performances. The two main outcomes of *Set*-3 were: first, OCBA does not benefit as much from utilizing σ_i/σ_j if the variances are linearly increasing or decreasing, and does not outperform *KW*-RaceRR by a large amount. In fact, it is slightly worse if correlation is 0.9 and the variances are increasing. Second, OCBA is more robust than *KW*-RaceRR to the distribution type, mainly under high correlation and Weibull data.

In summary, the Ranking and Selection literature and the Algorithm Configuration literature now have a number of new algorithms to choose from, such that better performance is achieved under various conditions. Some, like *KW*-RaceRR, are new to both fields and showed to be competitive to the currently used algorithms.

CHAPTER 6 Conclusion

The topic of this thesis was the algorithm configuration problem. That is, setting the values of an algorithm's parameters such that it performs best with respect to some performance metric(s). The methods used to set those parameters were termed as configurators. They were broadly classified as online or offline, the major difference being whether the parameters are learnt while solving the problem, or after the problem is solved over a representative training set. This work introduced three new offline configurators to *efficiently* learn the best algorithm parameters.

6.1. Summary of contributions

6.1.1. Meta-Optimization with a Flexible Budget

The first contribution concerned meta-optimizers, where the configurator iteratively *modifies* its solutions (parameter settings) to reach better ones. Current meta-optimization methods can only find the best parameter values for a pre-determined computational budget for the optimization algorithm. If similar problem instances are to be solved with different budgets, the whole process has to be repeated from scratch with each change, which may be computationally infeasible.

A more efficient method was presented here, the Flexible Budget method, which can discover the best parameter settings for any computational budget, less than a specified maximum n_{max}, without re-running the meta-optimization process. It works by utilizing the entire convergence curve to calculate a rank-based utility of a parameter setting, at every evaluation point up to n_{max}. Such information is used by the configurator to find the best parameter values for any budget. Three versions of the method were developed, differing in how the utility is calculated. The first used the Length of the convergence curve (L), the second used the Area Under the Curve (AUC), and the third used the Area Lost (AL) by not selecting a specific parameter setting.

The experimental validation consisted of comparing the solution quality of the best parameter settings found by the Flexible Budget method, to those of the standard *Fixed* Budget method (that had to be re-run for each different budget). The comparison was done for various computational budgets. The expectation was that the same solution quality can be achieved, but with less computational effort (a single run). The savings increase with the number of different budgets the problem needs to be solved within (discretization). Three sets of experiments were carried out. The first, *Set*-1, involved eight multimodal optimization functions, all in 5 dimensions, used for training and testing. Each of the other two had five training instances and four other testing instances. The instances were taken from the hump, *Set*-2, and quadratic, *Set*-3, family of functions. Sets 2 and 3 were in 5, 10, and 15 dimensions.

Results for *Set*-1 showed that the Flexible Budget method achieved significantly *better* results on 62% of the test functions and computational budgets chosen, 38% of the time the difference in its performance was insignificant, and it never performed worse. These numbers are based on the AL version of the Flexible Budget method, which was not the best or the worst performing in *Set*-1. Computational savings were about 60% of the effort required by the Fixed Budget method, for the chosen discretization levels.

Results for *Set*-2 varied with the dimensionality of the functions. In general, for the AL version, the Flexible Budget method performed significantly better 30%-45% of the time, 36%-60% of the time there was no significant difference in performance, and 0%-20% it was significantly worse than the Fixed Budget method. Though in the latter case the difference were minor and, depending on the application, may be acceptable compared to the savings gained. The savings ranged from 66% (5 and 10 dimensions) to 70% (15 dimensions) of the effort required by the Fixed Budget method, for the chosen discretization levels.

Results for *Set*-3 also varied with dimensionality. The AL version showed significant improvement 38%-100% of the time, 0%-49% there was no significant difference in performance, and 0%-13% of the time it performed significantly worse (again minor). Time savings were at 64% of the effort required by the Fixed Budget method for all dimensions.

As for which version of the Flexible Budget to use, the experiments conducted here point to AL when the training set differs from the testing set and to AUC when they are similar, but this is limited to the settings used in this thesis.

6.1.2. Racing with a self-adaptive significance level and one-way Racing with an intelligent budget allocation

The second and third contributions are summarized together, as they share the same experimental setup and evaluation. Both are offline configurators that *select* the best out of a set of parameter settings. Racing algorithms, as configurators, were shown to be sensitive to their hyper-parameter α, in terms of incorrectly selecting an inferior parameter setting as the best. Moreover, their termination criterion is either to stop when the computational budget is consumed, or when the “perceived” best is identified. In situations where the budget is large, and there is no advantage of terminating early, Racing is not guaranteed to make use of the entire budget, and premature termination will waste a lot of the computational budget available.

The second contribution was Racing with reset, which tackled both of these issues. Whenever a race terminates before the budget is consumed, the algorithm rolls back to the iteration when the first dropout occurred, lowers α by a factor, then re-applies the statistical tests on *all* the parameter settings. This reset is repeated till the budget is consumed. In this fashion α changes online and any parameter setting that was incorrectly discarded at any point, now has the potential to compete in the race again.

The third contribution is an extension to the second. Racing with reset was originally applied to F-Race, the most widely used Racing algorithm to tune parameters. It relies on a two-way ANOVA test, which dictates having an equal number of samples for each of the competing parameter settings whenever the tests are applied. More importantly, every time a dropout occurs, the remaining parameter settings *must* all be sampled the same number of times (once is the default). This has two drawbacks, one, it does not allow for a more intelligent allocation of the budget in each iteration. And two, it does not allow Racing with reset to use all previously collected data when it rolls back to the iteration when the first dropout occurred.

To overcome these two issues, one-way Racing algorithms were introduced, that do not require an equal number of samples for each surviving parameter setting. Namely, they were: KW-Race and A-Race_1WUB. The first is non-parametric and relies on the Kruskal-Wallis test, while the second is parametric and relies on an unbalanced ANOVA.

The experimental validation consisted of comparing the developed Racing algorithms to those used in the literature, basically F-Race and its parametric version A-Race_2W. In addition, EBA and two algorithms from the Simulation Optimization literature were added, OCBA and CBA, as they address a similar problem of selecting the best simulation system from a set. The performance measures were the Probability of Incorrect Selection ($PICS$) and the Expected Opportunity Cost $\mathbb{E}[OC]$, calculated experimentally over 100,000 replications. Given these performance measures, the parameter settings were simulated with probability distributions (systems) with pre-determined means, variances, and correlations.

Three sets of experiments were conducted. *Set*-1 had a pre-determined budget for all algorithms, which meant that Racing had to use the reset idea. *Set*-2 had Racing set the budget for OCBA, CBA, and EBA to see if it provides it with any advantage. Also, Racing with reset was added to *Set*-2 to show the improvement in performance over its standard version. Finally, *Set*-3 was similar to *Set*-1, but used a larger number of systems and different probability distributions.

The results indicated that no one algorithm outperformed the rest on all test cases and performance measures. Yet, it was shown under which conditions each algorithm performed best. In general, OCBA and *KW*-RaceRR were the best over most cases, with OCBA being slightly better if the budget is low and/or the variances are increasing or decreasing. Under high correlation, two-way Racing algorithms become competitive, especially for low budgets, and outperformed the rest in some cases. A more detailed summary is presented in Tables 6.1-6.3.

Table 6.1. A summary of the best performing algorithms (based on PICS) in *Set*-1 experiments.

Case	Distribution	Budget	Correlation	Best algorithm
1	$\sim\mathcal{N}(i,6^2)\forall i=0,\dots,k-1$	≥ 700	0.0 or 0.9	*KW*-RaceRR
		≤ 500	0.0	OCBA
		≤ 350	0.9	*F*-RaceR
2	$\sim\mathcal{N}(i,(k-i)^3)\forall i=0,\dots,k-1$	Throughout	0.0	OCBA
		Throughout	0.9	*A*-RaceR_2W
3	$\sim\mathcal{N}(i,(i+1)^4)\forall i=0,\dots,k-1$	Throughout	0.0 or 0.9	OCBA
4	$\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$ *Set*-A and *Set*-B	High	0.0 or 0.9	*KW*-RaceRR
		Low	0.0	OCBA
		Low	0.9	*A*-RaceR_2W

Table 6.2. A summary of the best performing algorithms (based on PICS) in *Set*-2 experiments.

Case	Distribution	α range	Correlation	Best algorithm
1	$\sim\mathcal{N}(i,6^2)\forall i=0,\dots,k-1$	0.3-0.02	0.0	*KW*-Race
		0.01-0.00001	0.0	OCBA
		0.3-0.00001	0.9	*KW*-Race
2	$\sim\mathcal{N}(i,(k-i)^3)\forall i=0,\dots,k-1$	0.3-0.00001	0.0	OCBA
		0.3-0.01	0.9	*KW*-Race
		0.001-0.00001	0.9	OCBA
3	$\sim\mathcal{N}(i,(i+1)^4)\forall i=0,\dots,k-1$	0.3-0.00001	0.0 and 0.9	OCBA
4	$\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$ *Set*-A	0.3-0.2 (low budget)	0.0	OCBA
		0.1-0.00001	0.0	*KW*-Race
		0.3-0.00001	0.9	*KW*-Race
	$\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$ *Set*-B	0.3-0.02	0.0	*KW*-Race
		0.01-0.00001	0.0	OCBA
		0.3-0.08	0.9	*KW*-Race
		0.01-0.00001	0.9	OCBA

Table 6.3. A summary of the best performing algorithms (based on PICS) in *Set*-3 experiments.

Case	Means and variances	Budget	Distribution type	Correlation	Best algorithm
1	$\sim\mathcal{D}(i,6^2)\forall i = 0,\dots,k-1$	≥ 1500	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.0	KW-RaceRR
		≤ 1200	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.0	OCBA
		≤ 700	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	KW-RaceRR
		≥ 1500	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	Indistinguishable
2	$\sim\mathcal{D}\big(i,3(k-i)\big)\forall i = 0,\dots,k-1$	Entire	$\mathcal{N},\mathcal{G}$	0.0	OCBA
		≥ 1500	$\mathcal{W}$	0.0	KW-RaceRR
		≥ 1500	$\mathcal{N},\mathcal{G}$	0.9	KW-RaceRR
		Entire	$\mathcal{W}$	0.9	Indistinguishable
3	$\sim\mathcal{D}\big(i,4(i+1)\big)\forall i = 0,\dots,k-1$	Entire	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.0	OCBA
		≤ 500	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	KW-RaceRR
		≥ 700	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	Indistinguishable
4	$\sim\mathcal{D}\big(U(0{,}15),U(20{,}40)\big)$	≤ 1200	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.0	OCBA
		≥ 1500	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.0	KW-RaceRR
		≤ 700	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	KW-RaceRR
		≥ 1000	$\mathcal{N},\mathcal{G},\mathcal{W}$	0.9	Indistinguishable

As for trying to intelligently allocate Δ within a one-way Racing algorithm, using OCBA, results showed that it only improves performance whenever OCBA is superior to Racing. That is, if the variances are increasing or decreasing. At which point one can just use OCBA by itself. However, this does not mean that no better performance can be achieved by intelligently allocating Δ, it only means that OCBA was not the best choice.

6.2. Limitations, extensions, and future research

All the work done in this thesis was experimental, and while a vast number of scenarios were tested, the conclusions are limited to the chosen settings. If one is faced with a different situation, the results presented here may serve as a guideline as to which algorithm the user can start testing.

Other experiments and extensions, which could not be implemented within the available time frame, include: for the Flexible Budget method, running the same experiments on more test functions and real optimization problems. Using multi-objective optimization methods during training, with each instance being a different objective, instead of averaging performance. Finally, using the best parameter settings suggested by the Flexible Budget method, at each evaluation point, as a schedule to change the parameters online, then comparing it to algorithms that adapt parameters based on feedback from the search. For computational budget allocators, the same experiments can be run with data drawn from other distributions, possibly discrete, and from real optimization problems. MAB solvers (like UCB1) and $\mathcal{KN}++$ (Kim and Nelson, 2006) can be added to the comparison, with some modifications. The algorithms can be tested under dynamic conditions where the distributions change over time. Finally, one-way Racing can be combined with other methods to intelligently allocate Δ, for instance the two-stage procedure of Rinott (1978).

Areas which received little, or no, attention thus far, and require additional research, in the author's opinion, include: determining the number of applications and replications required to properly evaluate an operator in online control. Whether adapting, or controlling, hyper-parameters of the configurator would enable it to discover new (better) parameter settings, ones which it could not have found with its default hyper-parameters, or will it have no advantage.

Reference list

1. Adenso-Diaz, B. and Laguna, M., 2006. Fine tuning of algorithms using fractional experimental designs and local search. *Operations Research,* 54 (1), pp.99-114.

2. Ansotegui, C., Sellmann, M. and Tierney, K., 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In*:* Gent, I. P. eds. *Principles and Practice of Constraint Programming Conference.* Berlin Heidelberg: Springer, pp.142-157.

3. Auer, P. and Ortner, R., 2010. UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica,* 61 (1-2), pp.55-65.

4. Auer, P., Bianchi, N. C. and Fischer, P., 2002. Finite-time analysis of the multi-armed bandit problem. *Machine Learning,* 47 (2-3), pp.235-256.

5. Babic, D. and Hu, A. J., 2007. Structural abstraction of software verification conditions. In*:* Damm, W. and Hermanns, H. eds. *Computer Aided Verification.* Berlin Heidelberg: Springer, pp.366-378.

6. Balaprakash, P., Birattari, M., Stützle, T. and Dorigo, M., 2009. Adaptive sample size and importance sampling in estimation-based local search for the probabilistic traveling salesman problem. *European Journal of Operational Research,* 199 (1), pp.98-110.

7. Bartz-Beielstein, T., Friese, M., Zaefferer, M., Naujoks, B., Flasch, O., Konen, W. and Koch, P., 2011. Noisy optimization with sequential parameter optimization and optimal computational budget allocation. In*:* Krasnogor, N. ed. *Genetic and Evolutionary Computation Conference.* Dublin, 12-16 July. New York: ACM.

8. Bartz-Beielstein, T., Lasarczyk, C. W. G. and Preuss, M., 2005. Sequential parameter optimization. In*:* Mckay, B. ed. *IEEE Congress on Evolutionary Computation.* Edinburgh, 2-4 Sep. Piscataway: IEEE Press.

9. Battiti, R. and Campigotto, P., 2008. Reinforcement learning and reactive search: an adaptive MAX-SAT solver. In*:* Ghallab, M., Spyropoulos, C. D., Fakotakis, N. and Avouris, N. eds. *European Conference on Artificial Intelligence.* Amsterdam: IOS Press.

10. Battiti, R. and Campigotto, P., 2011. An investigation of reinforcement learning for reactive search optimization. In*:* Hamadi, Y., Monfroy, E. and Saubion, F. eds. *Autonomous Search.* Berlin Heidelberg: Springer, pp.131-160.

11. Battiti, R. and Protasi, M., 1997. Reactive search, a history-sensitive heuristic for MAX-SAT. *Journal of Experimental Algorithmics,* 2 (1), pp.9-17.

12. Battiti, R., Mascia, F. and Brunato, M., 2009. Metrics, landscapes and features. *Operations Research Computer Science Interfaces Series,* 45 (1), pp.1-14.

13. Bertsimas, D. and Niño-Mora, J., 2000. Restless bandits, linear programming relaxations, and a primal-dual index heuristic. *Operations Research,* 48 (1), pp.80-90.

14. Birattari, M., 2005. On the estimation of the expected performance of a metaheuristic on a class of instances how many instances, how many runs? TR/IRIDIA/2004-001, Universite Libre de Bruxelles

15. Birattari, M., Balaprakash, P. and Dorigo, M., 2007. The ACO/F-Race algorithm for combinatorial optimization under uncertainty. In: Doerner, K., Gendreau, M., Greistorfer, P., Gutjahr, W., Hartl, R. and Reimann, M. eds. *Metaheuristics.* Berlin: Springer, pp.189-203.

16. Birattari, M., Yuan, Z., Balaprakash, P. and Stützle, T., 2009. F-Race and Iterated F-Race: An overview. TR/IRIDIA/2009-018. Universit´e Libre de Bruxelles

17. Birattari, M., Yuan, Z., Balaprakash, P. and Stützle, T., 2010. F-Race and iterated F-Race: An overview. In*:* Bartz-Beielstein, T., Chiarandini, M., Paquete, L. and Preuss, M. eds. *Experimental Methods for the Analysis of Optimization Algorithms.* Berlin: Springer, pp.311-336.

18. Blanchet, J., Jingchen, L. and Zwart, B., 2008. Large deviations perspective on ordinal optimization of heavy-tailed systems. In*:* Mason, S., Hill, R., Moench, L. and Rose, O. eds. *Winter Simulation Conference.* Miami, 7-10 Dec. New York: ACM.

19. Blum, C. and Socha, K., 2005. Training feed-forward neural networks with ant colony optimization: an application to pattern classification. In*:* Nedjah, N., Mourelle, L. M., Vellasco, M. M. B. R., Abraham, A. and Köppen, M. eds. *Hybrid Intelligent Systems.* Rio de Janeiro, 6-9 Nov. Washington, D.C.: IEEE Computer Society.

20. Bontempi, G., 2011. An optimal stopping strategy for online calibration in local search. In*:* Coello, C. a. C. ed. *Learning and Intelligent OptimizatioN.* Berlin Heidelberg: Springer, pp.106-115.

21. Botee, H. M. and Bonabeau, E., 1998. Evolving ant colony optimization. *Advances in Complex Systems,* 1 (2-3), pp.149-160.

22. Box, G. E. P. and Andersen, S. L., 1954. *Robust Tests for Variances and Effect of non-Normality and Variance Heterogeneity on Standard Tests.* Raleigh: University of North Carolina.

23. Bradley, R. A., 1952. Corrections for non-normality in the use of the two-sample t- and F-test at high significance levels. *The Annals of Mathematical Statistics,* 23 (1), pp.103-113.

24. Branke, J. and Elomari, J. A., 2012. Meta-optimization for parameter tuning with a flexible computing budget. In*:* Soule, T. ed. *Genetic and Evolutionary Computation Conference.* Philadelphia, 7-11 July. New York: ACM.

25. Branke, J. and Elomari, J., 2013. Racing with a fixed budget and a self-adaptive significance level. In*:* Pardalos, P. and Nicosia, G. eds. *Learning and Intelligent OptimizatioN.* Catania, 7-11 Jan.

26. Branke, J., Chick, S. E. and Schmidt, C., 2007. Selecting a selection procedure. *Management Sciences,* 53 (12), pp.1916-1932.

27. Brantley, M. W., Lee, L. H., Chun Hung, C. and Chen, A., 2008. Optimal sampling in design of experiment for simulation-based stochastic optimization. In*:* Gao, Y. and Reveliotis, S. eds. *IEEE International Conference on Automation Science and Engineering.* Arlington, 23-26 Aug. Piscataway: IEEE Press.

28. Bruss, F. T., 2000. Sum the odds to one and stop. *The Annals of Probability,* 28 (3), pp.1384-1391.

29. Bubeck, S., Munos, R. and Stoltz, G., 2011. Pure exploration in finitely-armed and continuous-armed bandits. *Theoretical Computer Science,* 412 (19), pp.1832-1852.

30. Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Mccollum, B., Ochoa, G., Parkes, A. J. and Petrovic, S., 2011. The cross-domain heuristic search challenge; an international research competition. In*:* Coello, C. a. C. ed. *Learning and Intelligent OptimizatioN.* Berlin Heidelberg: Springer, pp.631-634.

31. Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E. and Qu, R., 2013. Hyper-heuristics: A Survey of the state of the art. *Journal of the Operational Research Society,* 64 (9), pp.1-30.

32. Burke, E. K., Mccollum, B., Meisels, A., Petrovic, S. and Qu, R., 2007. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research,* 176 (1), pp.177-192.

33. Burke, E., Silva, J. D. and Soubeiga, E., 2005. Multi-objective hyper-heuristic approaches for space allocation and timetabling. In*:* Ibaraki, T., Nonobe, K. and Yagiura, M. eds. *Metaheuristics: Progress as Real Problem Solvers.* Berlin: Springer, pp.129-158.

34. Caelen, O. and Bontempi, G., 2005. How to allocate a restricted budget of leave-one-out assessments for effective model selection in machine learning: a comparison of state-of-the-art techniques. In*:* Verbeeck, K., Tuyls, K., Nowé, A., Manderick, B. and Kuijpers, B. eds. *Belgain-Dutch Conference on Artificial Intelligence*. Brussels, 17-18 Oct. Brussels: Koninklijke Vlaamse Academie van Belgie voor Wetenschappen en Kunsten.

35. Chakhlevitch, K. and Cowling, P., 2008. Hyperheuristics: Recent developments. In*:* Cotta, C., Sevaux, M. and Sörensen, K. eds. *Adaptive and Multilevel Metaheuristics.* Berlin Heidelberg: Springer, pp.3-29.

36. Chen, C. H. and Lee, L. H., 2010. *Stochastic Simulation Optimization: An Optimal Computing Budget Allocation.* Singapore: World Scientific Publishing.

37. Chen, C. H., Lin, J., Yucesan, E. and Chick, S. E., 2000. Simulation budget allocation for further enhancing the efficiency of ordinal optimization. *Discrete Event Dynamic Systems,* 10 (3), pp.251-270.

38. Chen, C. H., 1995. An effective approach to smartly allocate computing budget for discrete event simulation. *IEEE Conference on Decision and Control.* New Orleans, 13-15 Dec. Washington, D.C.: IEEE Press.

39. Chen, C. H., Yucesan, E., Dai, L. and Chen, H. C., 2010. Efficient Computation of Optimal Budget Allocation for Discrete Event Simulation Experiment. *IIE Transactions,* 42 (1), pp.60-70.

40. Chen, C. H., He, D., Fu, M. and Lee, L. H., 2008. Efficient simulation budget allocation for selecting an optimal subset. *INFORMS Journal on Computing,* 20 (4), pp.579-595.

41. Chen, E. J. and Lee, L. H., 2009. A multi-objective selection procedure of determining a Pareto set. *Computers and Operations Research,* 36 (6), pp.1872-1879.

42. Chen, H. C., Dai, L., Chen, C. H. and Yucesan, E., 1997. New development of optimal computing budget allocation for discrete event simulation. In*:* Andradottir, S., Healy, K. J., Withers, D. H. and Nelson, B. L. eds. *Winter Simulation Conference.* Atlanta, 8-11 Dec. Piscataway: IEEE Press.

43. Chiarandini, M. and Stutzle, T., 2007. Stochastic local search algorithms for graph set T-colouring and frequency assignment. *Constraints,* 12 (3), pp.371-403.

44. Chiarandini, M., Birattari, M., Socha, K. and Rossi-Doria, O., 2006. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling,* 9 (5), pp.403-432.

45. Chick, S. E. and Inoue, K., 2001. New two-stage and sequential procedures for selecting the best simulated system. *Operations Research,* 49 (5), pp.732-743.

46. Clerc, M. and Kennedy, J., 2002. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation,* 6 (1), pp.58-73.

47. Cochran, W. G., 1968. Errors of measurement in statistics. *Technometrics,* 10 (4), pp.637-666.

48. Conover, W. J., 1999. *Practical Nonparametric Statistics.* 3rd ed. New York: John Wiley & Sons.

49. Cortez, P., Rocha, M. and Neves, J., 2001. A meta-genetic algorithm for time series forecasting. In*:* Torgo, L. ed. *Artificial Intelligence Techniques for Financial Time Series Analysis.* Oporto, Portugal, 17-20-Dec.

50. Cowling, P. I., Kendall, G. and Soubeiga, E., 2001. A hyperheuristic approach to scheduling a sales summit. In*:* Burke, E. and Erben, W. eds. *Practice and Theory of Automated Timetabling Conference.* Konstanz, 16-18 Aug. Berlin Heidelberg: Springer.

51. Coy, S. P., Golden, B. L., Runger, G. C. and Wasil, E. A., 2001. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics,* 7 (1), pp.77-97.

52. Dacosta, L., Fialho, A., Schoenauer, M. and Sebag, M., 2008. Adaptive operator selection with dynamic multi-armed bandits. In*:* Ryan, C. and Keijzer, M. eds. *Genetic and Evolutionary Computation Conference.* Atlanta, New York: ACM.

53. Daniel, W. W., 1990. *Applied Nonparametric Statistics.* 2nd ed. Boston: PWS-Kent Publishing.

54. Dayanik, S., Powell, W. and Yamazaki, K., 2008. Index policies for discounted bandit problems with availability constraints. *Advances in Applied Probability,* 40 (2), pp.377-400.

55. De Jong, K., 2007. Parameter setting in EAs: a 30 year perspective. In*:* Lobo, F. G., Lima, C. F. and Michalewicz, Z. eds. *Parameter Setting in Evolutionary Algorithms.* Berlin Heidelberg: Springer, pp.1-18.

56. Deb, K., 1999. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation,* 7 (3), pp.205-230.

57. Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation,* 6 (2), pp.182-197.

58. Di Gaspero, L. and Roli, A., 2008. Stochastic local search for large-scale instances of the haplotype inference problem by pure parsimony. *Journal of Algorithms,* 63 (3), pp.55-69.

59. Di Tollo, G., Lardeux, F., Maturana, J. and Saubion, F., 2011. From adaptive to more dynamic control in evolutionary algorithms. In*:* Merz, P. and Hao, J.K. eds. *Evolutionary Computation in Combinatorial Optimization.* Berlin: Springer, pp.130-141.

60. Drugan, M. and Thierens, D., 2011. Generalized adaptive pursuit algorithm for genetic pareto local search algorithms. In*:* Krasnogor, N. ed. *Genetic and Evolutionary Computation Conference.* Dublin, 12-16 July. New York: ACM.

61. Dudewicz, E. J. and Dalal, S. R., 1975. Allocation of observations in ranking and selection with unequal variances. *Sankhyā: The Indian Journal of Statistics,* 37 (1), pp.28-78.

62. Egon, S. P., 1931. The analysis of variance in cases of non-normal variation. *Biometrika,* 23 (1-2), pp.114-133.

63. Eiben, A. E. and Smit, S. K., 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation,* 1 (1), pp.19-31.

64. Eiben, A. E., Hinterding, R. and Michalewicz, Z., 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation,* 3 (2), pp.124-141.

65. Elashoff, J. D., 1969. Analysis of covariance: a delicate instrument. *American Educational Research Journal,* 6 (3), pp.383-401.

66. Fawcett, C., Hoos, H. H. and Chiarandini, M., 2009. An automatically configured modular algorithm for post enrollment course timetabling. TR-2009-15. University of British Columbia

67. Fialho, A., 2010. *Adaptive Operator Selection for Optimization.* PhD. Universit´e Paris-Sud.

68. Fialho, A., Da Costa, L., Schoenauer, M. and Sebag, M., 2008. Extreme value based adaptive operator selection. In*:* Rudolph, G., Jansen, T., Lucas, S., Poloni, C. and Beume, N. eds. *Parallel Problem Solving from Nature.* Berlin Heidelberg: Springer, pp.175-184.

69. Fialho, A., Da Costa, L., Schoenauer, M. and Sebag, M., 2010. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence,* 60 (1-2), pp.25-64.

70. Francesca, G., Pellegrini, P., Stützle, T. and Birattari, M., 2011. Off-line and on-line tuning: A study on operator selection for a memetic algorithm applied to the QAP. In*:* Merz, P. and Hao, J.K. eds. *Evolutionary Computation in Combinatorial Optimization.* Berlin Heidelberg: Springer, pp.203-214.

71. Friedman, M., 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association,* 32 (200), pp.675-701.

72. Fu, M. C., Hu, J. Q., Chen, C. H. and Xiong, X., 2004. Optimal computing budget allocation under correlated sampling. In*:* Ingalls, R., Rossetti, M., Smith, J. and B. Peters eds. *Winter Simulation Conference.* Washington D.C., 5-8 Dec.

73. Fu, M. C., Hu, J. Q., Chen, C. H. and Xiong, X., 2007. Simulation allocation for determining the best design in the presence of correlated sampling. *INFORMS Journal on Computing,* 19 (1), pp.101-111.

74. Gagliolo, M. and Schmidhuber, J., 2006. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence,* 47 (3), pp.295-328.

75. Gagliolo, M. and Schmidhuber, J., 2011. Algorithm portfolio selection as a bandit problem with unbounded losses. *Annals of Mathematics and Artificial Intelligence,* 61 (2), pp.49-86.

76. Gagliolo, M., Zhumatiy, V. and Schmidhuber, J., 2004. Adaptive online time allocation to search algorithms. In*:* Boulicaut, J.F., Esposito, F., Giannotti, F. and Pedreschi, D. eds. *Machine Learning.* Berlin Heidelberg: Springer, pp.134-143.

77. Games, P. A. and Lucas, P. A., 1966. Power of the analysis of variance of independent groups on non-normal and normally transformed data. *Educational and Psychological Measurement* 26 (2), pp.311-327.

78. Gaspero, L. and Urli, T., 2012. Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In*:* Hamadi, Y. and Schoenauer, M. eds. *Learning and Intelligent OptimizatioN.* Berlin Heidelberg: Springer, pp.384-389.

79. Gaspero, L., Tollo, G., Roli, A. and Schaerf, A., 2007. Hybrid local search for constrained financial portfolio selection problems. In*:* Hentenryck, P. and Wolsey, L. eds. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems.* Berlin Heidelberg: Springer, pp.44-58.

80. Gittins, J. C., 1979. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society,* 41 (2), pp.148-177.

81. Gittins, J., Glazebrook, K. and Weber, R., 2011, *Multi-armed Bandit Allocation Indices.* 2nd ed. West Sussex: Wiley.

82. Glass, G. V., Peckham, P. D. and Sanders, J. R., 1972. Consequences of failure to meet assumptions underlying the fixed effects analyses of variance and covariance. *Review of educational research,* 42 (3), pp.237-288.

83. Glynn, P. and Juneja, S., 2004. A large deviations perspective on ordinal optimization. In*:* Ingalls, R. G. and Rossetti, M. D. eds. *Winter Simulation Conference.* Washington, D.C., 5-8 Dec. Piscataway: IEEE Press.

84. Gong, W., Fialho, L. and Cai, Z., 2010. Adaptive strategy selection in differential evolution. In*:* Pelikan, M. ed. *Genetic and Evolutionary Computation Conference.* Portland, 7-11 July. New York: ACM.

85. Grefenstette, J. J., 1986. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems Man and Cybernetics,* 16 (1), pp.122-128.

86. Hansen, N. and Kern, S., 2004. Evaluating the CMA evolution strategy on multimodal test functions. In*:* Yao, X., Burke, E., Lozano, J., Smith, J., Merelo-Guervós, J., Bullinaria, J., Rowe, J., Tiňo, P., Kabán, A. and Schwefel, H.P. eds. *Parallel Problem Solving from Nature.* Berlin: Springer, pp.282-291.

87. Hansen, N. and Ostermeier, A., 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation,* 9 (2), pp.159-195.

88. Hansen, N., 2006. The CMA evolution strategy: A comparing review. In*:* Lozano, J., Larrañaga, P., Inza, I. and Bengoetxea, E. eds. *Towards a New Evolutionary Computation.* Berlin Heidelberg: Springer, pp.75-102.

89. Hansen, N., Muller, S. D. and Koumoutsakos, P., 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation,* 11 (1), pp.1-18.

90. He, D., Lee, L. H., Chen, C. H., Fu, M. C. and Wasserkrug, S., 2010. Simulation optimization using the cross-entropy method with optimal computing budget allocation. *ACM Transactions on Modeling and Computer Simulation,* 20 (1), pp.1-22.

91. Herdy, M., 1992. Reproductive isolation as strategy parameter in hierarchically organized evolution strategies. In*:* Manner, R. and Manderick, B. eds. *Parallel Problem Solving from Nature.* Brussels, Belgium, 28-30 Sep. New York: Elsevier.

92. Hill, T., 2011. Knowing when to stop. *American Scientist* 97 (1), pp.136-133.

93. Hinkley, D. V., 1971. Inference about the change-point from cumulative sum tests. *Biometrika*, 58 (3), pp.509.

94. Hordijk, W., 1996. A measure of landscapes. *Evolutionary Computation*, 4 (4), pp.335-360.

95. Horsnell, G., 1953. The effect of unequal group variances on the F-test for the homogeneity of group means. *Biometrika*, 40 (1-2), pp.128-136.

96. Hsu, T.C. and Feldt, L. S., 1969. The effect of limitations on the number of criterion score values on the significance level of the F-test. *American Educational Research Journal*, 6 (4), pp.515-527.

97. Hutter, F., 2009b. *Automated Configuration of Algorithms for Solving Hard Computational Problems.* PhD. University of British Columbia.

98. Hutter, F., Babic, D., Hoos, H. H. and Hu, A. J., 2007a. Boosting verification by automatic tuning of decision procedures. In*:* Baumagartner, J. and Sheeran, M. eds. *Formal Methods in Computer Aided Design.* Los Alamitos: IEEE Computer Society, pp.27-34.

99. Hutter, F., Bartz-Beielstein, T., Hoos, H., Leyton-Brown, K. and Murphy, K. P., 2010a. Sequential model-based parameter optimization: An experimental investigation of automated and interactive approaches. In*:* Bartz-Beielstein, T., Chiarandini, M., Paquete, L. and Preuss, M. eds. *Experimental Methods for the Analysis of Optimization Algorithms.* Berlin Heidelberg: Springer, pp.363-414.

100. Hutter, F., Hoos, H. and Leyton-Brown, K., 2010b. Automated configuration of mixed integer programming solvers. In*:* Lodi, A., Milano, M. and Toth, P. eds. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems.* Berlin Heidelberg: Springer, pp.186-202.

101. Hutter, F., Hoos, H. and Stutzle, T., 2007b. Automatic algorithm configuration based on local search. *National Conference on Artificial Intelligence*, Vancouver. Palo Alto: AAAI Press.

102. Hutter, F., Hoos, H. H. and Leyton-Brown, K., 2011. Sequential model-based optimization for general algorithm configuration. In*:* Coello, C. a. C. eds. *Learning and Intelligent OptimizatioN.* Berlin Heidelberg: Springer, pp.507-523.

103. Hutter, F., Hoos, H. H., Leyton-Brown, K. and Murphy, K. P., 2009b. An experimental investigation of model-based parameter optimisation: SPO and beyond. In*:* Rothlauf, F. ed. *Genetic and Evolutionary Computation Conference.* Montreal, 8-12 July. New York: ACM.

104. Hutter, F., Hoos, H., Leyton-Brown, K. and Murphy, K., 2010c. Time-bounded sequential parameter optimization. In*:* Blum, C. and Battiti, R. eds. *Learning and Intelligent OptimizatioN.* Berlin: Springer, pp.281-298.

105. Hutter, F., Hoos, H., Leyton-Brown, K. and Stützle, T., 2009a. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research,* 36 (1), pp.267-306.

106. Igel, C., Heidrich-Meisner, V. and Glasmachers, T., 2008. Shark. *Journal of Machine Learning Research,* 9 (1), pp.993-996.

107. Jones, T., 1995. Crossover, mutation, and population-based search. In*:* Eshelman, L. J. ed. *Genetic Algorithms Conference.* Pittsburgh, 15-19 July. San Francisco: Morgan Kaufmann.

108. Kallel, L., Naudts, B. and Reeves, C. R., 2001. Properties of fitness functions and search landscapes. In*:* Kallel, L., Naudts, B. and Rogers, A. eds. *Theoretical Aspects of Evolutionary Computing.* Berlin Heidelberg: Springer, pp.175-206.

109. Kauffman, S. and Levin, S., 1987. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology,* 128 (1), pp.11-45.

110. Khudabukhsh, A., Xu, L., Hoos, H. and Leyton-Brown, K., 2009. SATenstein: Automatically building local search SAT solvers from components. In: Boutilier, C. ed. *International Joint Conference on Artificial Intelligence*. Pasadena, 11-17 July. San Francisco: Morgan Kaufmann.

111. Kim, S. H. and Nelson, B. L., 2006. Selecting the best system. In*:* Shane, G. H. and Barry, L. N. eds. *Handbooks in Operations Research and Management Science.* San Francisco: Elsevier, pp.501-534.

112. Kim, S. H., 2013. Statistical ranking and selection. In*:* Gass, S. I. and Fu, M. C. eds. *Encyclopedia of Operations Research and Management Science.* New York: Springer.

113. Kleinberg, R., Niculescu-Mizil, A. and Sharma, Y., 2010. Regret bounds for sleeping experts and bandits. *Machine Learning,* 80 (2-3), pp.245-272.

114. Knowles, J. and Corne, D., 2002. Towards landscape analyses to inform the design of a hybrid local search for the multi-objective quadratic assignment problem. In*:* Abraham, A., Ruiz-Del-Solar, J. and Köppen, M. eds. *Soft Computing Systems: Design, Management and Applications.* Lansdale: IOS Press, pp.271-279.

115. Koza, J. R., 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge: The MIT press.

116. Kruskal, W. H. and Wallis, W. A., 1952. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association,* 47 (260), pp.583-621.

117. Lasarczyk, C. W. G., 2007. *Genetische programmierung einer algorithmischen Chemie.* PhD. Technische Universität Dortmund.

118. Lee, L. H., Chew, E. P., Teng, S. and Goldsman, D., 2004. Optimal computing budget allocation for multi-objective simulation models. In*:* Rossetti, M. D. and Ingalls, R. G. eds. *Winter Simulation Conference.* Washington, D.C., 5-8 Dec. New York: ACM.

119. Lenne, R., Solnon, C., Stutzle, T., Tannier, E. and Birattari, M., 2008. Reactive stochastic local search algorithms for the genomic median problem. In*:* Hemert, J. V. and Cotta, C. eds. *Evolutionary Computation in Combinatorial Optimization.* Berlin Heidelberg: Springer, pp.266-276.

120. Lindawati, Lau, H. and Lo, D., 2011. Instance-based parameter tuning via search trajectory similarity clustering. In*:* Coello, C. C. ed. *Learning and Intelligent OptimizatioN.* Berlin Heidelberg: Springer, pp.131-145.

121. Lindawati, Yuan, Z., Lau, H. C. and Zhu, F., 2013. Automated parameter tuning framework for heterogeneous and large instances: Case study in quadratic assignment problem. *Learning and Intelligent OptimizatioN.* Catania, 7-11 Jan.

122. Lindquist, E. F., 1953. *Design and Analysis of Experiments in Education and Psychology.* Oxford: Houghton Mifflin.

123. Loo Hay, L., Ek Peng, C. and Suyan, T., 2007. Finding the Pareto set for multi-objective simulation models by minimization of expected opportunity cost. In*:* Henderson, S., Biller, B., Hsieh, M. H. and Shortle, J. eds. *Winter Simulation Conference* Washington, D.C., 9-12 Dec. Piscataway: IEEE Press.

124. Lourenco, H. R., Martin, O. and Stutzle, T., 2002. Iterated local search. In*:* Kochenberger, F. G. G. ed. *Handbook of Metaheuristics.* Norwell: Kluwer Academic Publishers, pp.321–353.

125. Lunney, G. H., 1970. Using analysis of variance with a dichotomous dependent variable: an empirical study. *Journal of Educational Measurement,* 7 (4), pp.263-269.

126. Marascuilo, L. A. and Mcsweeney, M., 1977, *Nonparametric and Distribution-Free Methods for the Social Sciences.* California: Brooks/Cole Publishing.

127. Maron, O. and Moore, A., 1994. Hoeffding races: accelerating model selection search for classification and function approximation. In*:* Cowan, J. D., Tesauro, G. and Alspector, J. eds. *Advances in Neural Information Processing Systems.* San Francisco: Morgan Kaufmann, pp.59-66.

128. Maturana, J. and Saubion, F., 2008. A compass to guide genetic algorithms. In*:* Rudolph, G., Jansen, T., Lucas, S., Poloni, C. and Beume, N. eds. *Parallel Problem Solving from Nature.* Berlin Heidelberg: Springer, pp.256-265.

129. Maturana, J., Fialho, A., Saubion, F., Schoenauer, M. and Sebag, M., 2009a. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In*:* Greensted, A. ed. *IEEE Congress on Evolutionary Computation.* Trondheim, 18-21 May. Piscataway: IEEE Press.

130. Maturana, J., Lardeux, F. and Saubion, F., 2009b. Controlling behavioral and structural parameters in evolutionary algorithms. In*:* Collet, P., Monmarché, N., Legrand, P., Schoenauer, M. and Lutton, E. eds. *Artificial Evolution.* Strasbourg, 26-28 Oct. Berlin: Springer.

131. Maxwell, S. E. and Delaney, H. D., 2003. *Designing Experiments and Analyzing Data: A Model Comparison Perspective.* 2nd ed. New York: Routledge Academic.

132. Mcclymont, K. and Keedwell, E. C., 2011. Markov chain hyper-heuristic (MCHH): an online selective hyper-heuristic for multi-objective continuous problems. In*:* Krasnogor, N. ed. *Genetic and Evolutionary Computation Conference*, Dublin, 12-16 July. New York: ACM.

133. Meissner, M., Schmuker, M. and Schneider, G., 2006. Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics,* 7 (125), pp.121-133.

134. Mercer, R. E. and Sampson, J. R., 1977. Adaptive search using a reproductive meta-plan. *The International Journal of Systems and Cybernetics,* 7 (3), pp.215-228.

135. Merz, P. and Freisleben, B., 2000. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation,* 8 (1), pp.61-91.

136. Montgomery, D. and Runger, G., 2011, *Applied Statistics and Probability for Engineers.* 5th ed. New York: John Wiley & Sons.

137. Montgomery, D. C., 2001, *Design and Analysis of Experiments.* 5th ed. New York: John Wiley & Sons.

138. Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker and Berghe, G. V., 2012. An intelligent hyper-heuristic framework for CHeSC 2011. In*:* Hamadi, Y. and Schoenauer, M. eds. *Learning and Intelligent OptimizatioN.* Berlin Heidelberg: Springer, pp.461-466.

139. Nannen, V. and Eiben, A. E., 2006. A method for parameter calibration and relevance estimation in evolutionary algorithms. In: Keijzer, M. ed. *Genetic and Evolutionary Computation Conference*. Seattle, 8-12 July. New York: ACM.

140. Nannen, V. and Eiben, A. E., 2007. Relevance estimation and value calibration of evolutionary algorithm parameters. In*:* Sangal, R., Mehta, H. and Bagga, R. K. eds. *The International Joint Conference on Artificial Intelligence.* Hyderabad, 6-12 Jan. San Francisco: Morgan Kaufmann.

141. Nareyek, A., 2001. *Constraint-based agents: an architecture for constraint-based modeling and local-search-based reasoning for planning and scheduling in open and dynamic worlds.* Berlin: Springer.

142. Nareyek, A., 2004. Choosing search heuristics by non-stationary reinforcement learning. In*:* Mauricio, G. C. R., Jorge Pinho De, S. and Ana, V. eds. *Metaheuristics.* Norwell, USA: Kluwer Academic Publishers, pp.523-544.

143. Ochoa, G., Sebastien Verel, Fabio Daolio and Tomassini, M., 2011. Clustering of local optima in combinatorial fitness landscapes. In*:* Coello, C. a. C. ed. *Learning and Intelligent OptimizatioN.* Berlin Heidelberg: Springer, pp.454-457.

144. Ochoa, G., Tomassini, M., Verel, S. and Darabos, C., 2008. A study of NK landscapes' basins and local optima networks. In*:* Keijzer, M. ed. *Genetic and Evolutionary Computation Conference* Atlanta, 12-16 July. New York: ACM.

145. Oltean, M., 2005. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation,* 13 (3), pp.387-410.

146. O'neill, M. and Ryan, C., 2003. *Grammatical evolution: Evolutionary automatic programming in an arbitrary language.* Massachusetts: Kluwer Academic Publishers.

147. Paquete, L. and Stutzle, T., 2006. A study of stochastic local search algorithms for the biobjective quadratic assignment problem with correlated flow matrices. *European Journal of Operational Research,* 169 (3), pp.943-959.

148. Pedersen, M. E. H. and Chipperfield, A. J., 2008a. Local unimodal sampling. HL0801 Hvass Laboratories.

149. Pedersen, M. E. H. and Chipperfield, A. J., 2008b. Parameter tuning versus adaptation: proof of principle study on differential evolution. HL0803. Hvass Laboratories.

150. Pedersen, M. E. H. and Chipperfield, A. J., 2008c. Tuning differential evolution for artificial neural networks. HL0803. Hvass Laboratories.

151. Pellegrini, P., Stützle, T. and Birattari, M., 2010. Off-line vs. on-line tuning: A study on MAX–MIN ant system for the TSP. In*:* Dorigo, M., Birattari, M., Di Caro, G., Doursat, R., Engelbrecht, A., Floreano, D., Gambardella, L., Groß, R., Sahin, E., Sayama, H. and Stützle, T. eds. *Swarm Intelligence.* Berlin Heidelberg: Springer, pp.239-250.

152. Poli, R., Langdon, W. and Holland, O., 2005. Extending particle swarm optimisation via genetic programming. In*:* Keijzer, M., Tettamanzi, A., Collet, P., Van Hemert, J. and Tomassini, M. eds. *Genetic Programming.* Berlin Heidelberg: Springer, pp.142-142.

153. Pratt, J. W., 1964. Robustness of some procedures for the two-sample location problem. *Journal of the American Statistical Association,* 59 (307), pp.665-680.

154. Qin, A. K., Huang, V. L. and Suganthan, P. N., 2009. Differential Evolution Algorithm with Strategy Adaptation for Global Numerical Optimization. *IEEE Transactions on Evolutionary Computation,* 13 (2), pp.398-417.

155. Qu, H., Ryzhov, I. O. and Fu, M. C., 2012. Ranking and selection with unknown correlation structures. In*:* Laroque, C., Pasupathy, R. and Himmelspach, J. eds. *Winter Simulation Conference.* Berlin, 9-12 Dec. New York: ACM.

156. Qu, R. and Burke, E. K., 2008. Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society,* 60 (9), pp.1273-1285.

157. Rider, P. R., 1929. On the distribution of the ratio of mean to standard deviation in small samples from non-normal universes. *Biometrika,* 21 (1-4), pp.124-143.

158. Rinott, Y., 1978. On two-stage selection procedures and related probability-inequalities. *Communications in Statistics - Theory and Methods,* 7 (8), pp.799-811.

159. Rönkkönen, J., Li, X., Kyrki, V. and Lampinen, J., 2011. A framework for generating tunable test functions for multimodal optimization. *Soft Computing,* 15 (9), pp.1689-1706.

160. Ross, B. J., 2002. Searching for Search Algorithms: Experiments in Meta-search. CS-02-23. Brock University

161. Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L. and Stutzle, T., 2003. A comparison of the performance of different metaheuristics on the timetabling problem. In*:* Burke, E. and Causmaecker, P. eds. *Practice and Theory of Automated Timetabling IV.* Berlin Heidelberg: Springer, pp.329-351.

162. Runka, A., 2009. Evolving an edge selection formula for ant colony optimization. In*:* Rothlauf, F. ed. *Genetic and Evolutionary Computation Conference.* Montreal, 8-12 July. New York: ACM.

163. Rusmevichientong, P. and Tsitsiklis, J. N., 2010. Linearly parameterized bandits. *Mathematics of Operations Research,* 35 (2), pp.395-411.

164. Schaffer, J. D., Caruana, R. A., Eshelman, L. J. and Das, R., 1989. A study of control parameters affecting online performance of genetic algorithms for function optimization. In*:* Schaffer, J. D. ed. *International Conference on Genetic Algorithms.* George Mason University, 24-27 May. Fairfax: Morgan Kaufmann.

165. Scheffé, H., 1999. *The Analysis of Variance.* New York: Wiley Interscience.

166. Sechen, C. and Sangiovanni-Vincentelli, A., 1985. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits,* 20 (2), pp.510-522.

167. Shahookar, K. and Mazumder, P., 1990. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 9 (5), pp.500-511.

168. Sheskin, D., 2004. *Handbook of Parametric and Nonparametric Statistical Procedures.* 3rd ed. Florida: Chapman and Hall/CRC.

169. Shortle, J. F. and Chen, C. H., 2008. A preliminary study of optimal splitting for rare-event simulation. In*:* Mason, S., Hill, R., Moench, L. and Rose, O. eds. *Winter Simulation Conference.* Miami, 7-10 Dec. Piscataway: IEEE Press.

170. Shortleab, J. F., Chena, C.-H., Craina, B., Brodskyb, A. and Brodc, D., 2012. Optimal splitting for rare-event simulation. *IIE Transactions,* 44 (5), pp.352-367.

171. Smit, S. and Eiben, A., 2010a. Using entropy for parameter analysis of evolutionary algorithms. In*:* Bartz-Beielstein, T., Chiarandini, M., Paquete, L. and Preuss, M. eds. *Experimental Methods for the Analysis of Optimization Algorithms.* Berlin Heidelberg: Springer, pp.287-310.

172. Smit, S. K. and Eiben, A. E., 2010c. Parameter tuning of evolutionary algorithms: Generalist vs. Specialist. In*:* Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A., Goh, C.-K., Merelo, J., Neri, F., Preuß, M., Togelius, J. and Yannakakis, G. eds. *Applications of Evolutionary Computation.* Berlin Heidelberg: Springer, pp.542-551.

173. Smit, S. K. and Eiben, A. E., 2010. Beating the 'world champion' evolutionary algorithm via REVAC tuning. In: Nazaraf, S. ed. *IEEE Congress on Evolutionary Computation*. Barcelona, 18-23 July. Piscataway: IEEE Press.

174. Smith-Miles, K. and Lopes, L., 2011. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research,* 39 (5), pp.875–889.

175. Srivastava, A. B. L., 1959. Effect of non-normality on the power of the analysis of variance test. *Biometrika*, 46 (1-2), pp.114-122.

176. Stanhope, S. and Daida, J., 1998. Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment. In*:* Porto, V., Saravanan, N., Waagen, D. and Eiben, A. eds. *Evolutionary Programming VII.* Berlin Heidelberg: Springer, pp.693-702.

177. Stephenson, M., Amarasinghe, S., Martin, M. and O'reilly, U.-M., 2003. Meta optimization: improving compiler heuristics with machine learning. In*:* L, Michael. ed. *Programming Language Design and Implementation Conference.* San Diego, 9-11 Jun. New York: ACM.

178. Stützle, T., López-Ibáñez, M., Pellegrini, P., Maur, M., Montes De Oca, M., Birattari, M. and Dorigo, M., 2011. Parameter adaptation in ant colony optimization. In*:* Hamadi, Y., Monfroy, E. and Saubion, F. eds. *Autonomous Search.* Berlin Heidelberg: Springer, pp.191-215.

179. Tavares, J. and Pereira, F. B., 2011. Towards the development of self-ant systems. In*:* Krasnogor, N. ed. *Genetic and Evolutionary Computation Conference.* Dublin, 12-16 July. New York: ACM.

180. Tavares, J. and Pereira, F., 2012. Automatic design of ant algorithms with grammatical evolution genetic programming. In*:* Moraglio, A., Silva, S., Krawiec, K., Machado, P. and Cotta, C. eds. *Genetic Programming.* Berlin Heidelberg: Springer, pp.206-217.

181. Teng, S., Lee, L. H. and Chew, E. P., 2010. Integration of indifference-zone with multi-objective computing budget allocation. *European Journal of Operational Research,* 203 (2), pp.419-429.

182. Thierens, D., 2005. An adaptive pursuit strategy for allocating operator probabilities. In*:* Beyer, H. G. ed. *Genetic and Evolutionary Computation Conference.* Washington D.C., New York: ACM.

183. Tomassini, M., Vérel, S. and Ochoa, G., 2008. Complex-network analysis of combinatorial spaces: The NK landscape case. *Physical Review,* 78 (6), pp.11-21.

184. Wang, Y., Li, B., Weise, T., Wang, J., Yuan, B. and Tian, Q., 2011. Self-adaptive learning based particle swarm optimization. *Information Sciences*, 181 (20), pp.4515-4538.

185. Weber, R. R. and Gideon, W., 1990. On an index policy for restless bandits. *Journal of Applied Probability,* 27 (3), pp.637-648.

186. Weinberger, E. D., 1991. Local properties of Kauffman's N-k model: A tunably rugged energy landscape. *Physical Review A*, 44 (10), pp.6399-6413.

187. Weinberger, E., 1990. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63 (5), pp.325-336.

188. Whitacre, J. M., Pham, T. Q. and Sarker, R. A., 2006. Use of statistical outlier detection method in adaptive evolutionary algorithms. In*:* Cattolico, M. ed. *Genetic and Evolutionary Computation Conference.* Seattle, New York: ACM.

189. White, T., Pagurek, B. and Oppacher, F., 1998. ASGA: Improving the ant system by integration with genetic algorithms. In*:* Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H. and Riolo, R. L. eds. *Genetic Programming Conference.* Wisconsin, 22-25 July. San Francisco: Morgan Kaufmann.

190. Whittle, P., 1988. Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability,* 25 (1), pp.287-298.

191. Wilcox, R. R., 1987. New designs in analysis of variance. *Annual Review of Psychology,* 38 (1), pp.29-60.

192. Wong, Y. Y., Lee, K. H., Leung, K. S. and Ho, C. W., 2003. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications,* 7 (8), pp.506-515.

193. Xu, L., Hoos, H. H. and Leyton-Brown, K., 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In*:* Leake, D. ed. *Proceedings of the AAAI Conference on Artificial Intelligence.* Georgia, 11-15 July. Menlo Park: AAAI Press.

194. Xu, L., Hutter, F., Hoos, H. H. and Leyton-Brown, K., 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research,* 32 (1), pp.565-606.

195. Young, R. K. and Veldman, D. J., 1963. Heterogeneity and skewness in analysis of variance. *Perceptual and Motor Skills,* 16 (2), pp.588-588.

196. Yuan, B. and Gallagher, M., 2004. Statistical Racing techniques for improved empirical evaluation of evolutionary algorithms. In*:* Yao, X., Burke, E. K., Lozano, J. A., Smith, J., Merelo-Guervós, J. J., Bullinaria, J. A., Rowe, J. E., Tiňo, P., Kabán, A. and Schwefel, H.P. eds. *Parallel Problem Solving from Nature Conference.* Berlin Heidelberg: Springer, pp.172-181.

197. Yuan, B. and Gallagher, M., 2004. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In*:* Yao, X., Burke, E., Lozano, J., Smith, J., Merelo-Guervós, J., Bullinaria, J., Rowe, J., Tino, P., Kabán, A. and Schwefel, H. P. eds. *Parallel Problem Solving from Nature.* Berlin: Springer, pp.172-181.

198. Yuan, B. and Gallagher, M., 2007. Combining meta-EAs and Racing for difficult EA parameter tuning tasks. In*:* Lobo, F., Lima, C. and Michalewicz, Z. eds. *Parameter Setting in Evolutionary Algorithms.* Springer Berlin Heidelberg, pp.121-142.

199. Yuan, Z., Montes de Oca, M., Birattari, M. and Stützle, T., 2012. Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence,* 6 (1), pp.49-75.

200. Yuan, Z., Stützle, T. and Birattari, M., 2010. MADS/F-Race: Mesh adaptive direct search meets F-Race. In*:* García-Pedrajas, N., Herrera, F., Fyfe, C., Benítez, J. and Ali, M. eds. *Trends in Applied Intelligent Systems.* Springer Berlin Heidelberg, pp.41-50.

Appendix

A1. CBA vs. OCBA

The following hypothesis is made regarding CBA: CBA poorly estimates the covariance matrix, which degrades its performance. To experimentally validate this, CBA is run under two conditions: with an estimated covariance matrix, and with a fixed covariance matrix containing the nominal values. The results are then compared to the published ones in Fu *et al.* (2007), and to OCBA based on the *PICS*.

The experimental setup, as specified in Fu *et al.* (2007), is: $N = 500$ samples, $n_0 = 100$ samples, $k = 10$, $\Delta = 400$, (i.e. the algorithm is run once to allocate all what remains of the budget after n_0). Data are drawn from Normal distributions with monotonically increasing means and equal variances $\sim\mathcal{N}(i, 6^2)\forall i = 1, \dots, k$, and with randomly selected means and variances $\sim\mathcal{N}\big(U(1,k), U(10,24)\big)$. The system with the highest mean is considered the best. Finally, four correlation levels were used: 0.0, 0.2, 0.5, and 0.9. Note that OCBA was never run; instead, CBA was run with fixing the off-diagonal elements of the covariance matrix to zero. They called this algorithm the Independent Budget Allocation (IBA).

The results for the equal variances case are in Table A1.1 and Figure A1.1. Ten equally spaced correlation levels, from 0.0 to 0.9, are used here. It is clear that CBA can outperform OCBA only if the covariance matrix is fixed, or the correlation is high (≥ 0.7). More importantly, under high correlation CBA does not perform much better than EBA, which questions the need for it. Finally, IBA is not OCBA. That is why it cannot be claimed that CBA outperforms OCBA as the original paper does. The same observation is made for the random case, see Table A1.2. Though for this case CBA with a fixed covariance matrix was not run due to time limitations.

Table A1.1. Validation of CBA against published results based on *PICS*. $\sim\mathcal{N}(i, 6^2)\forall i = 1, \dots, k$.

Correlation	OCBA	CBA Est.	CBA Fix.	EBA	CBA Published	IBA Published
0.0	0.131	0.146	0.131	0.202	0.170	0.170
0.1	0.122	0.134	0.120	0.189		
0.2	0.108	0.127	0.107	0.171	0.133	0.141
0.3	0.101	0.119	0.095	0.151		
0.4	0.090	0.114	0.089	0.131		
0.5	0.077	0.099	0.074	0.108	0.068	0.097
0.6	0.068	0.079	0.058	0.081		
0.7	0.058	0.054	0.035	0.054		
0.8	0.050	0.024	0.024	0.024		
0.9	0.041	0.003	0.003	0.003	0.003	0.074

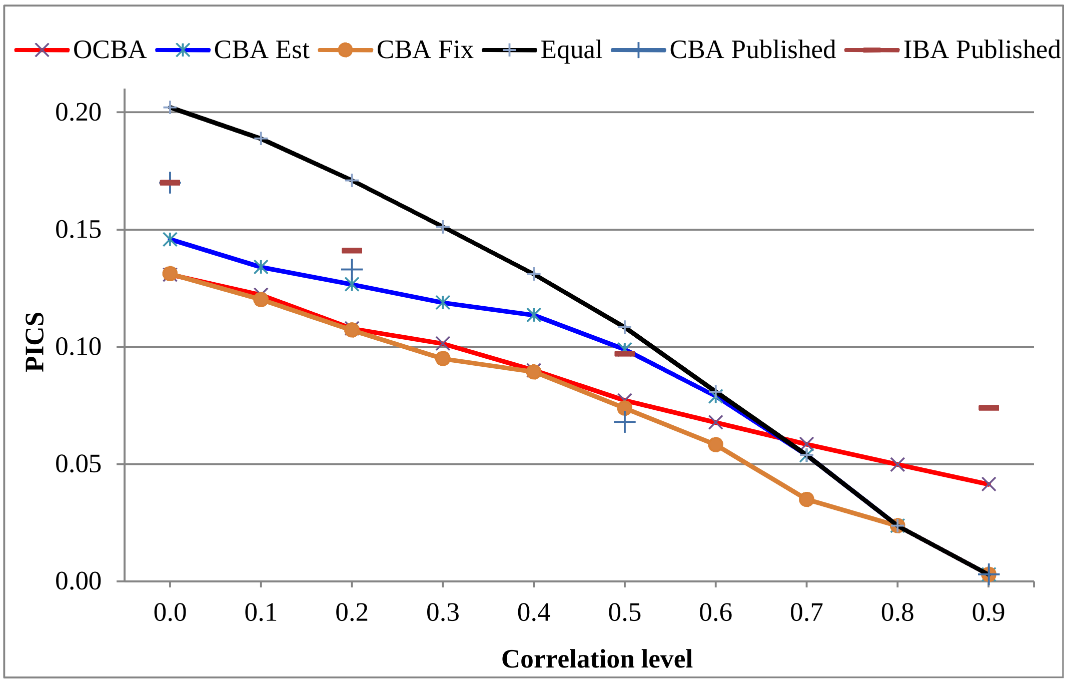


Figure A1.1. Validation of CBA against published results based on *PICS*. $\sim\mathcal{N}(i, 6^2)\forall i = 1, \dots, k$.

Table A1.2. Validation of CBA against published results based on *PICS*. $\sim\mathcal{N}\big(U(1,k), U(10{,}24)\big)$.

Correlation	OCBA	CBA Est.	EBA	CBA Published	IBA Published
0.0	0.254	0.274	0.323	0.303	0.303
0.1	0.248	0.273	0.306		
0.2	0.24	0.263	0.287	0.267	0.286
0.3	0.228	0.255	0.264		
0.4	0.211	0.235	0.240		
0.5	0.189	0.21	0.210	0.195	0.252
0.6	0.178	0.177	0.178		
0.7	0.165	0.137	0.137		
0.8	0.141	0.094	0.094		
0.9	0.102	0.039	0.039	0.041	0.178

Note that CBA better estimates high correlations compared to low ones. Figure A1.2 shows the estimated covariance between the best system (0) and all other systems (1-9) immediately after n_0. The numbers are for a randomly selected replication. It is clear that with no correlation, the covaraince are far from their nominal values (i.e. 0), compared to 0.9 correlation, where the covaraince are close to their nominal values of 32.4.

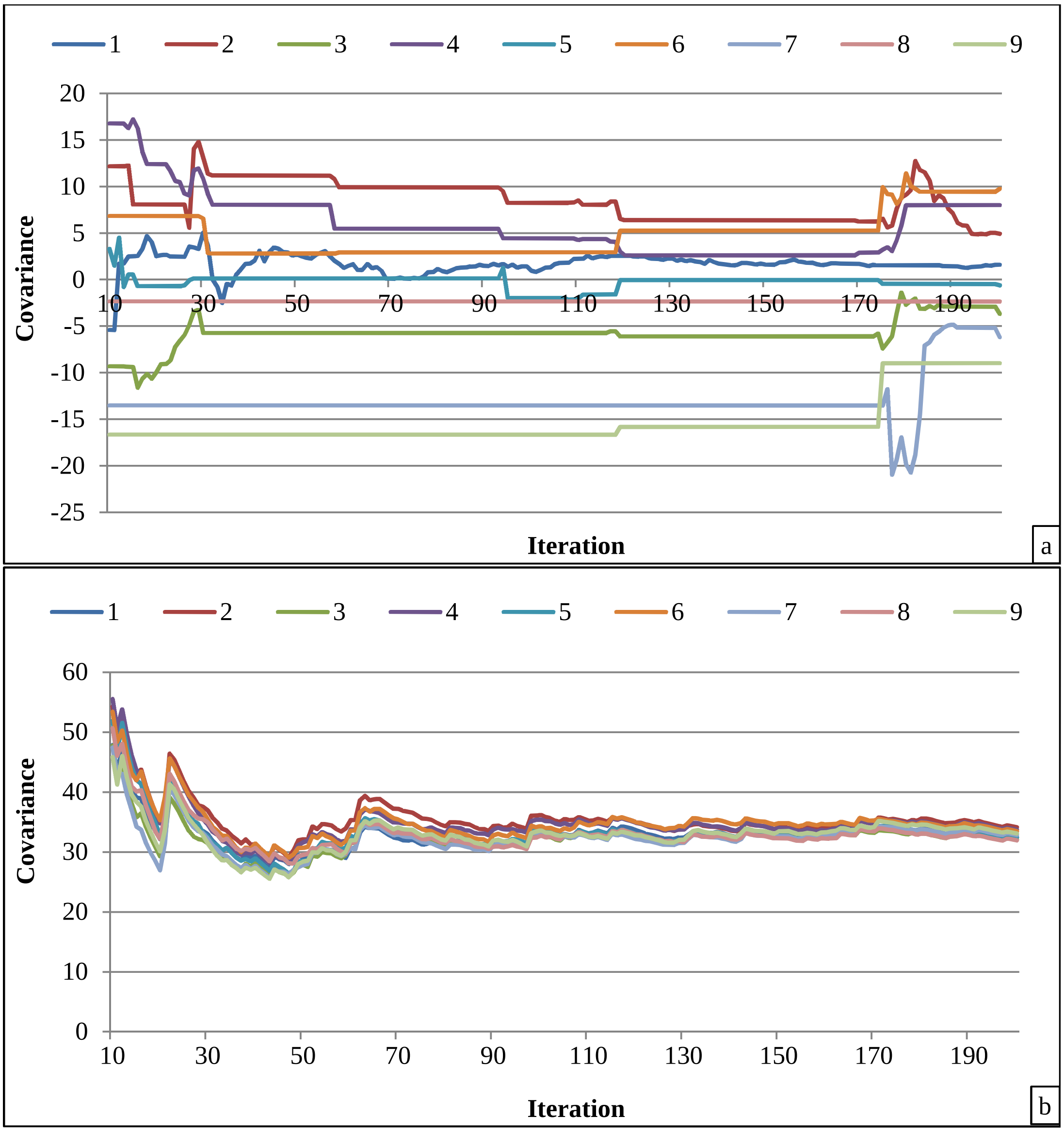


Figure A1.2. Estimation of the covariance between system 0 and all other systems. Estimation is made by CBA after n_0 at 0.0 (a) and 0.9 (b) correlation.

A2. *A*-Race_2Way setting the budget

This section briefly presents the results of *Set*-2 experiments when *A*-Race_2Way sets the budget for OCBA, CBA, and EBA. The conclusion is similar to the case when *KW*-Race sets the budget. That is, if *A*-Race_2Way sets the budget, it will not affect its performance ranking among OCBA, CAB, and EBA. Additionally, *A*-RaceR_2Way significantly improves ($p - value \ll 0.05$) over *A*-Race_2Way by adapting α_0, which enables a steady performance over a wide range of significance levels.

Figures A2.1- A2.6 show the f_r and $\mathbb{E}[OC]$ at 0.0 and 0.9 correlation for the same means and variances used in *Set*-1. The figures for the remaining correlation levels are not displayed as they follow the same pattern, but are summarized though in Tables A2.1- A2.5.

Table A2.1. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}(i, 6^2)\forall i = 0, \dots, k-1$.

	0.3 correlation					0.6 correlation				
Alpha	*A*_Race_2Way	OCBA	CBA	EBA	*A*_RaceR_2Way	*A*_Race_2Way	OCBA	CBA	EBA	*A*_RaceR_2Way
0.00001	0.009	0.006	0.041	0.194	0.009	0.000	0.003	0.033	0.141	0.000
0.001	0.003	0.007	0.045	0.256	0.005	0.000	0.005	0.058	0.201	0.000
0.01	0.005	0.012	0.067	0.303	0.005	0.003	0.017	0.107	0.239	0.000
0.1	0.054	0.078	0.164	0.352	0.003	0.035	0.081	0.206	0.276	0.000
0.3	0.184	0.216	0.283	0.367	0.003	0.126	0.185	0.267	0.284	0.000

Table A2.2. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}(i, (k-i)^3)\forall i = 0, \dots, k-1$.

	0.3 correlation					0.6 correlation				
Alpha	*A*_Race_2Way	OCBA	CBA	EBA	*A*_RaceR_2Way	*A*_Race_2Way	OCBA	CBA	EBA	*A*_RaceR_2Way
0.00001	0.469	0.382	0.482	0.471	0.469	0.411	0.342	0.424	0.416	0.411
0.001	0.458	0.383	0.480	0.480	0.457	0.389	0.343	0.423	0.426	0.389
0.01	0.429	0.380	0.479	0.501	0.429	0.353	0.344	0.422	0.447	0.353
0.1	0.417	0.414	0.494	0.546	0.407	0.333	0.368	0.462	0.506	0.323
0.3	0.500	0.502	0.544	0.589	0.414	0.434	0.461	0.520	0.547	0.328

Table A2.3. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}(i, (i+1)^4)\forall i = 0, \dots, k-1$.

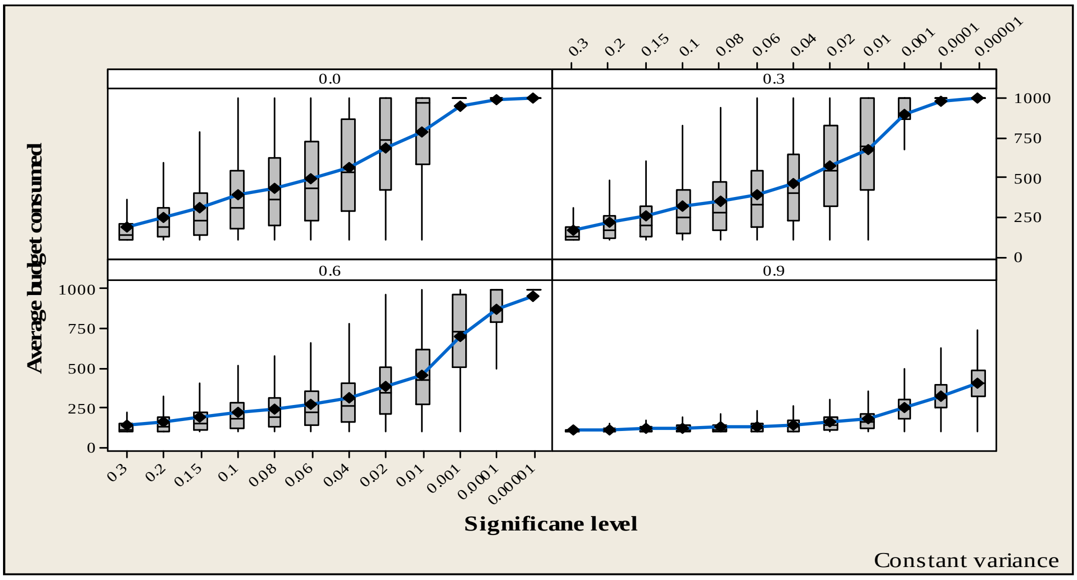
	0.3 correlation					0.6 correlation				
Alpha	*A*_Race_2Way	OCBA	CBA	EBA	*A*_RaceR_2Way	*A*_Race_2Way	OCBA	CBA	EBA	*A*_RaceR_2Way
0.00001	0.425	0.073	0.077	0.443	0.425	0.332	0.077	0.110	0.342	0.332
0.001	0.429	0.076	0.080	0.445	0.427	0.338	0.080	0.112	0.343	0.338
0.01	0.431	0.090	0.095	0.452	0.426	0.341	0.091	0.122	0.350	0.340
0.1	0.462	0.189	0.184	0.511	0.420	0.364	0.162	0.188	0.397	0.331
0.3	0.552	0.386	0.366	0.637	0.414	0.422	0.316	0.324	0.502	0.328

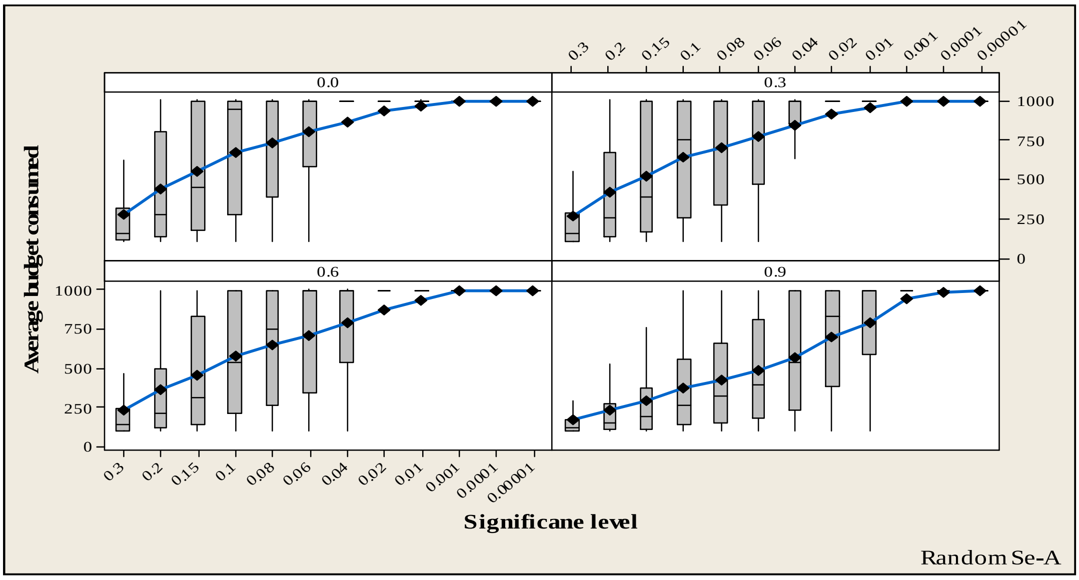
Table A2.4. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}\big(U(0,k),U(10{,}24)\big)$. *Set*-A.

	0.3 correlation					0.6 correlation				
Alpha	A_Race_2Way	OCBA	CBA	EBA	A_RaceR_2Way	A_Race_2Way	OCBA	CBA	EBA	A_RaceR_2Way
0.00001	0.274	0.262	0.338	0.449	0.274	0.196	0.187	0.322	0.351	0.196
0.001	0.262	0.262	0.339	0.492	0.262	0.174	0.188	0.325	0.402	0.173
0.01	0.248	0.262	0.343	0.521	0.244	0.164	0.192	0.342	0.447	0.160
0.1	0.277	0.307	0.407	0.563	0.239	0.205	0.264	0.420	0.520	0.151
0.3	0.432	0.447	0.505	0.580	0.231	0.378	0.413	0.511	0.539	0.144

Table A2.5. A summary of the f_r achieved by the competing algorithms at selected α points of an EC. Data are drawn from $\sim\mathcal{N}\big(U(0,k),U(10{,}24)\big)$. *Set*-B.

	0.3 correlation					0.6 correlation				
Alpha	A_Race_2Way	OCBA	CBA	EBA	A_RaceR_2Way	A_Race_2Way	OCBA	CBA	EBA	A_RaceR_2Way
0.00001	0.016	0.045	0.080	0.274	0.016	0.001	0.027	0.066	0.158	0.001
0.001	0.012	0.045	0.082	0.334	0.012	0.001	0.029	0.086	0.242	0.001
0.01	0.017	0.051	0.097	0.374	0.017	0.005	0.038	0.139	0.293	0.005
0.1	0.088	0.118	0.217	0.435	0.019	0.059	0.112	0.260	0.368	0.005
0.3	0.263	0.285	0.356	0.453	0.049	0.191	0.239	0.356	0.392	0.026





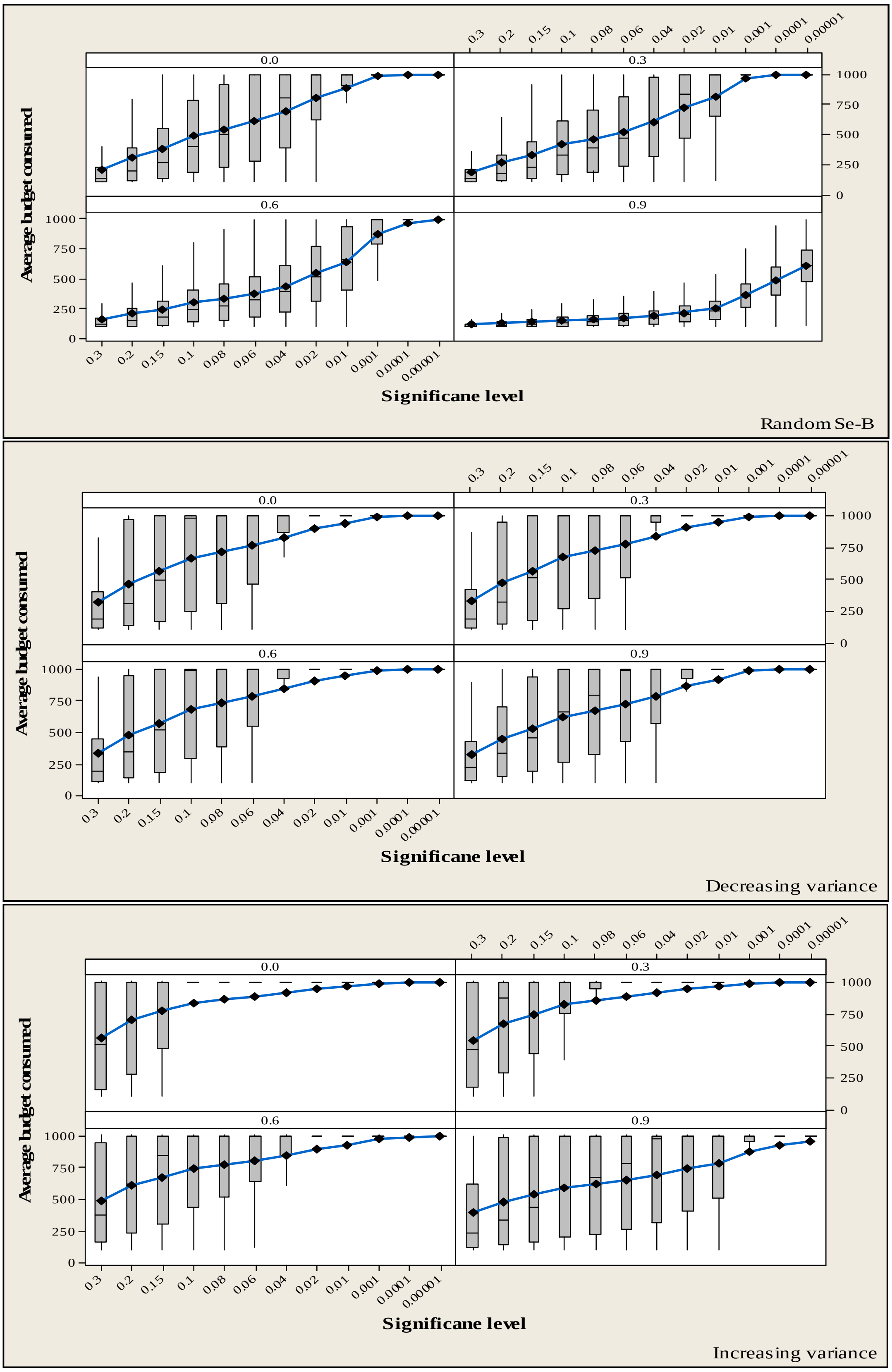


Figure A2.1. Box plots of the budget consumed by *A*-Race_2W at various α values, and different correlation levels. The connecting line is that of the means.

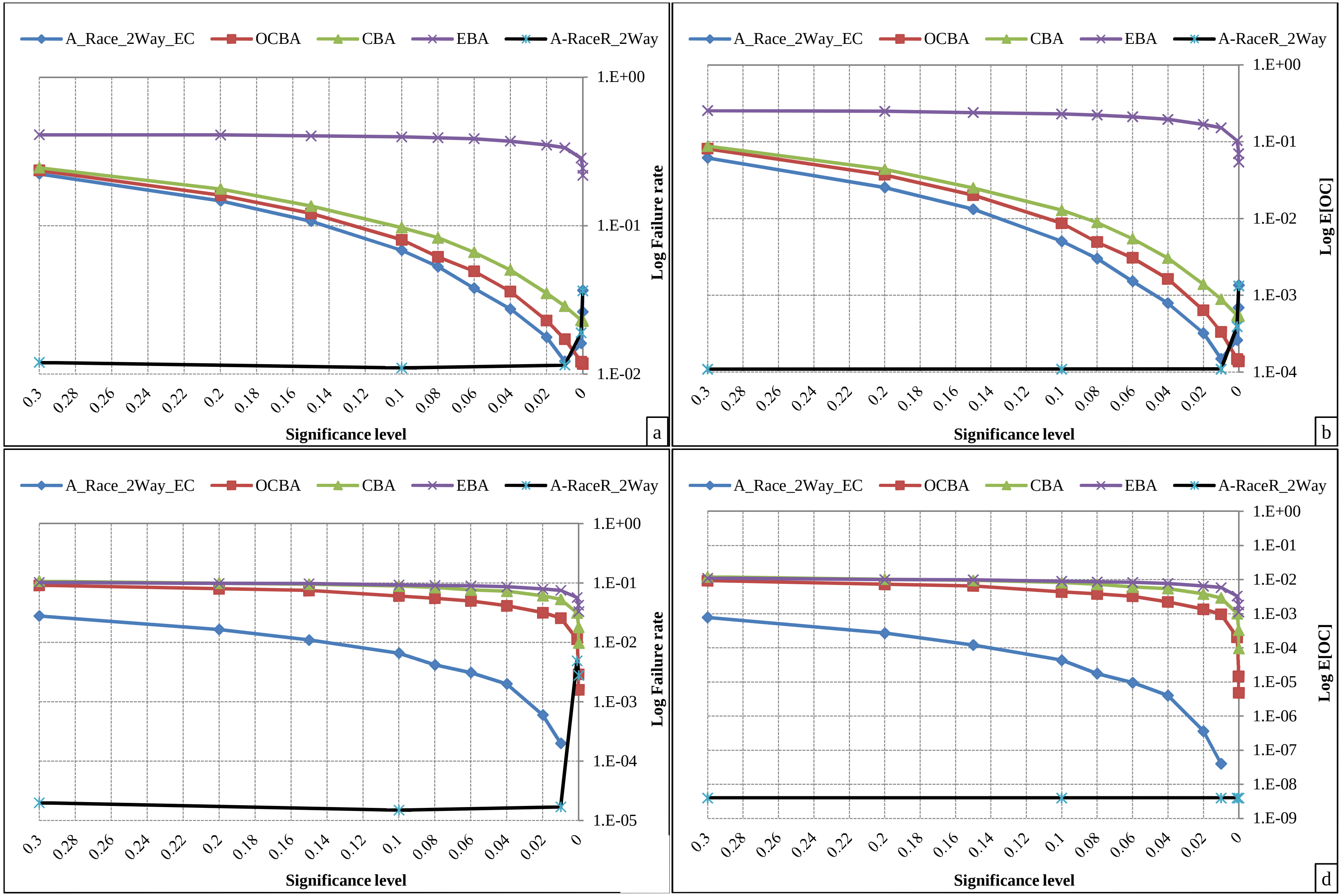


Figure A2.2. ECs based on f_r and *E[OC]* at 0.0 (a and b) and 0.9 (c and d) correlation levels. Data are drawn from $\sim\mathcal{N}(i, 6^2)\forall i = 0, \dots, k-1$.

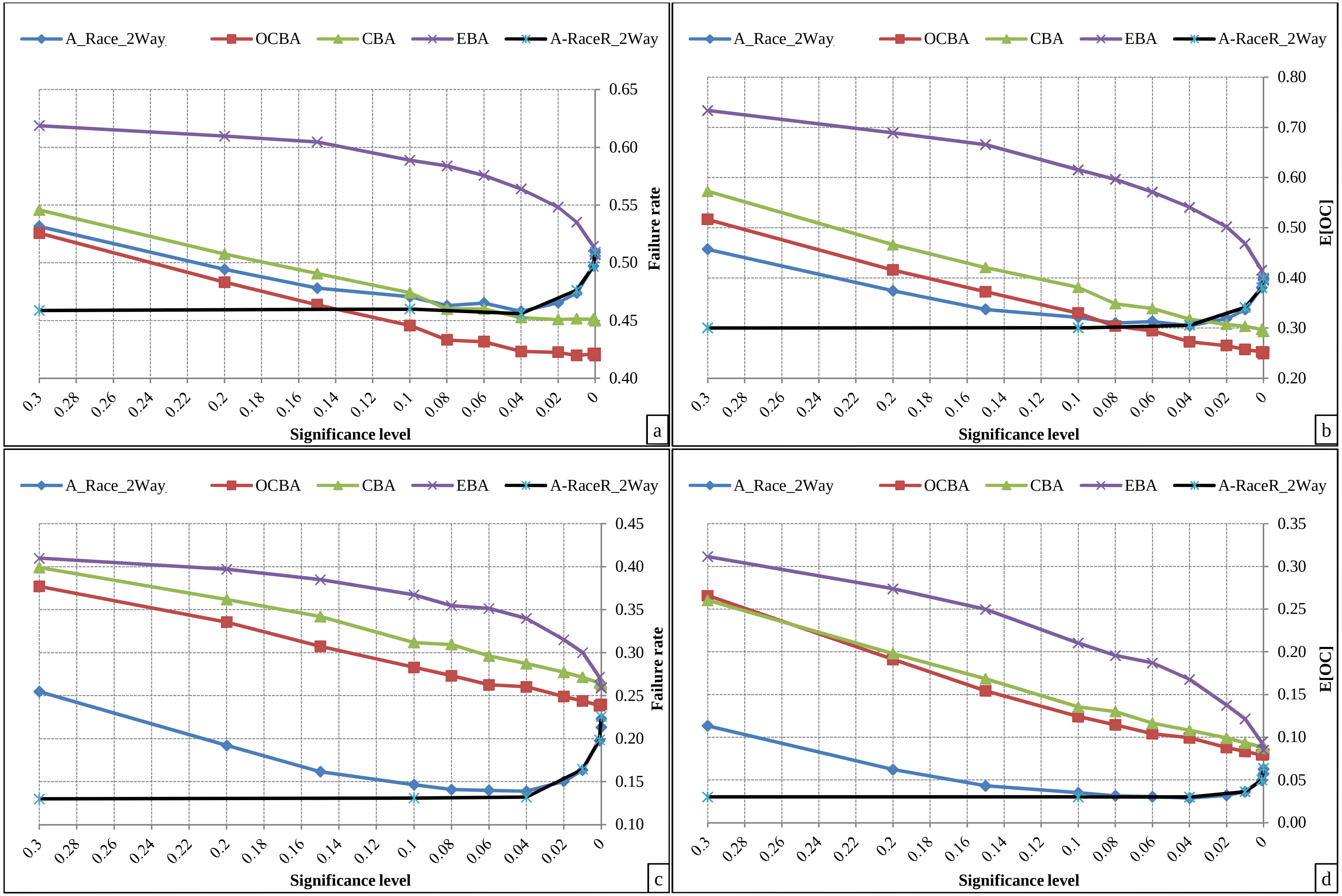


Figure A2.3. ECs based on f_r and $E[OC]$ at 0.0 (a and b) and 0.9 (c and d) correlation levels. Data are drawn from $\sim\mathcal{N}(i,(k-i)^3)\forall i = 0,\dots,k-1$.

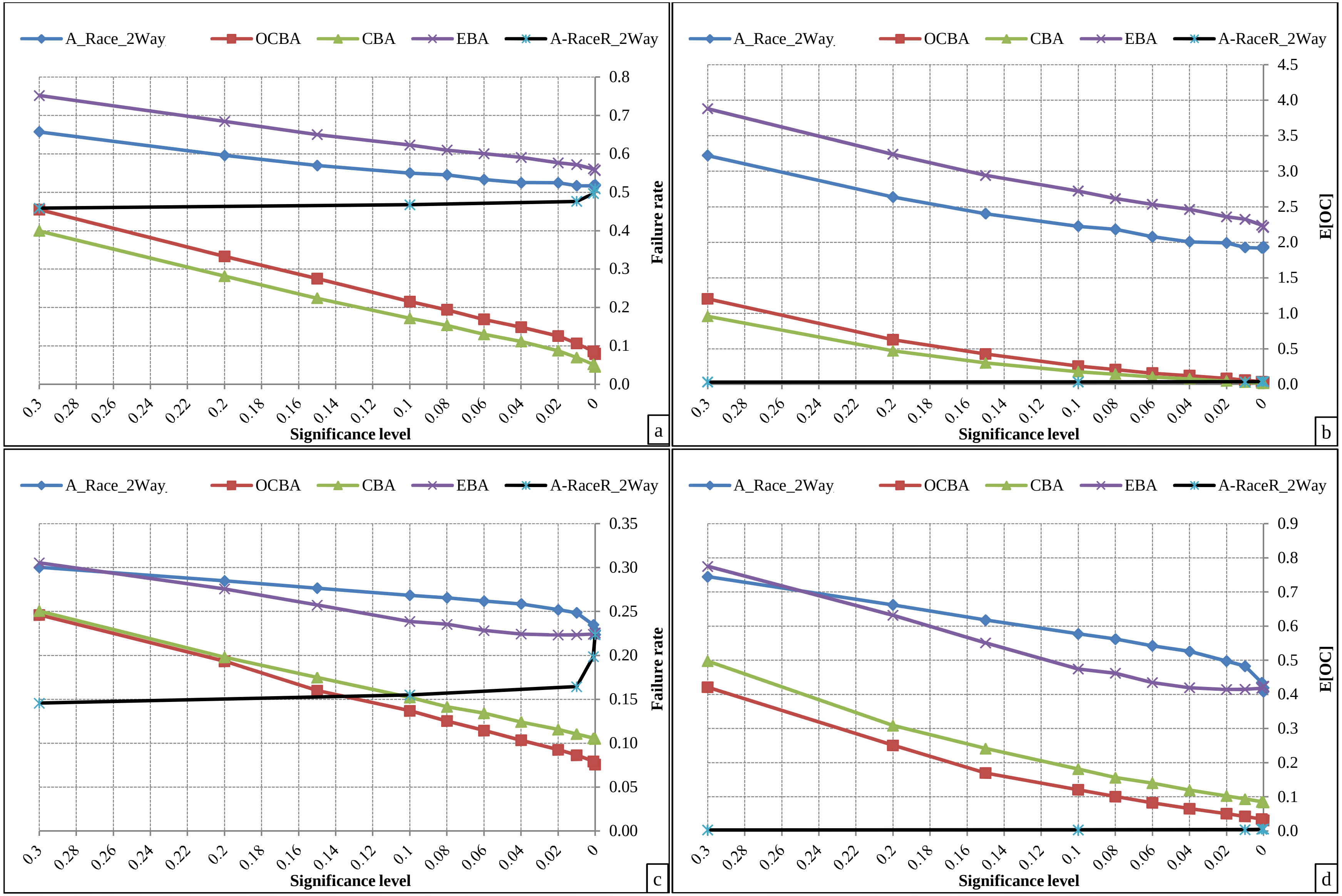


Figure A2.4. ECs based on f_r and *E[OC]* at 0.0 (a and b) and 0.9 (c and d) correlation levels. Data are drawn from $\sim\mathcal{N}(i,(i+1)^4)\forall i=0,\dots,k-1$.

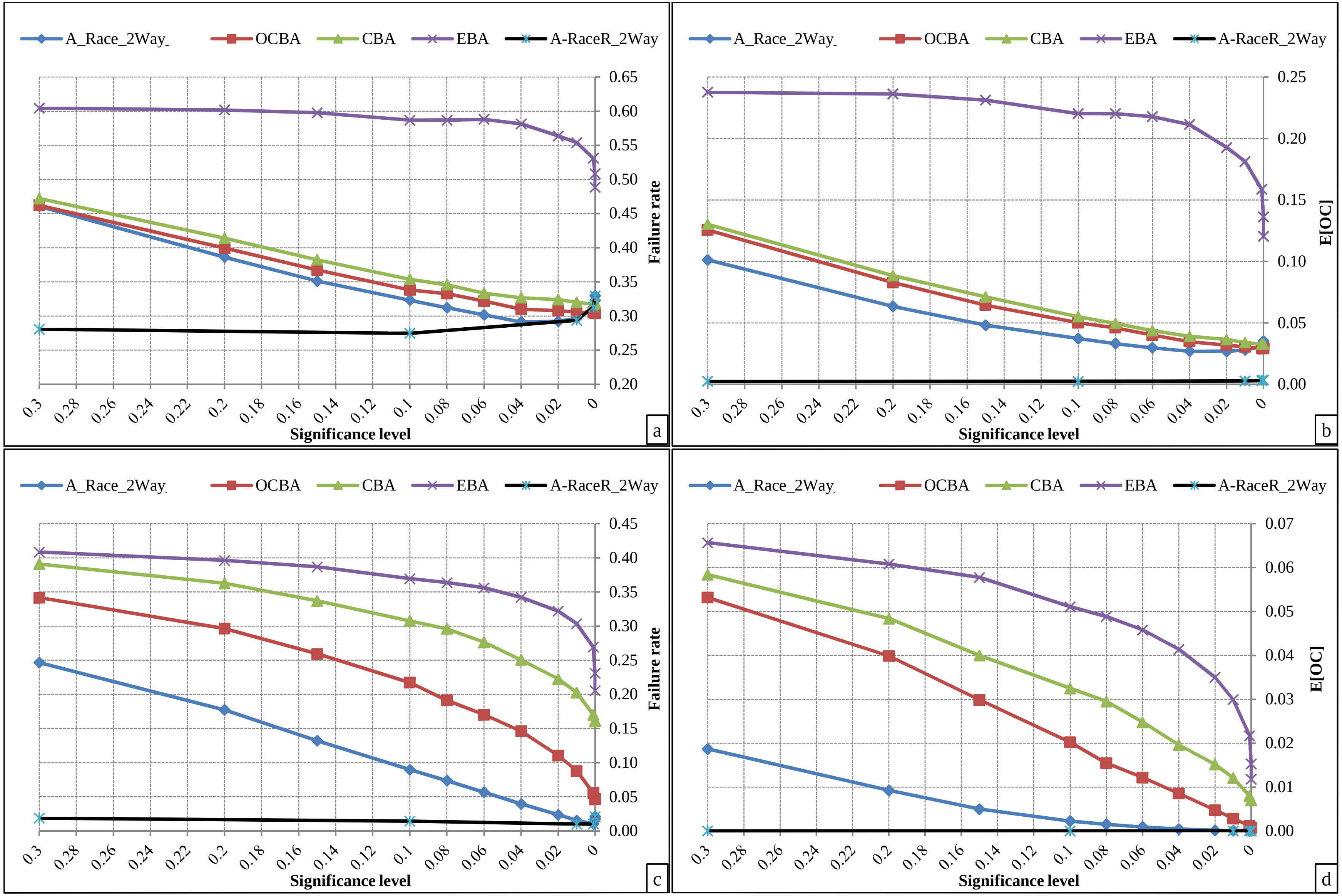


Figure A2.5. ECs based on f_r and *E[OC]* at 0.0 (a and b) and 0.9 (c and d) correlation levels. Data are drawn from $\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$ *Set*-A.

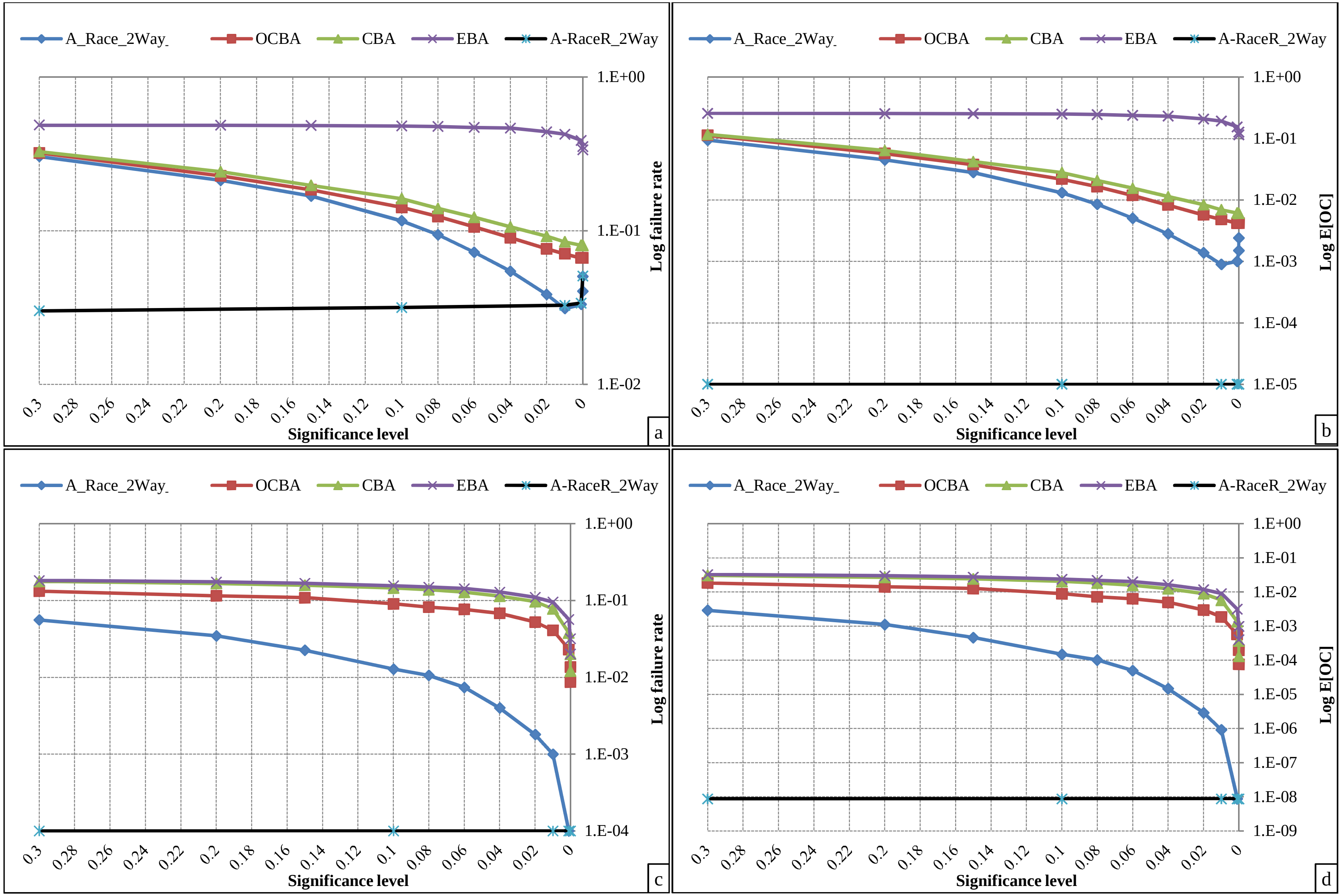


Figure A2.6. ECs based on f_r and *E[OC]* at 0.0 (a and b) and 0.9 (c and d) correlation levels. Data are drawn from $\sim\mathcal{N}\left(U(0,k),U(10,24)\right)$ *Set*-B.

A3. Remaining results of *Set*-1 and *Set*-2 experiments

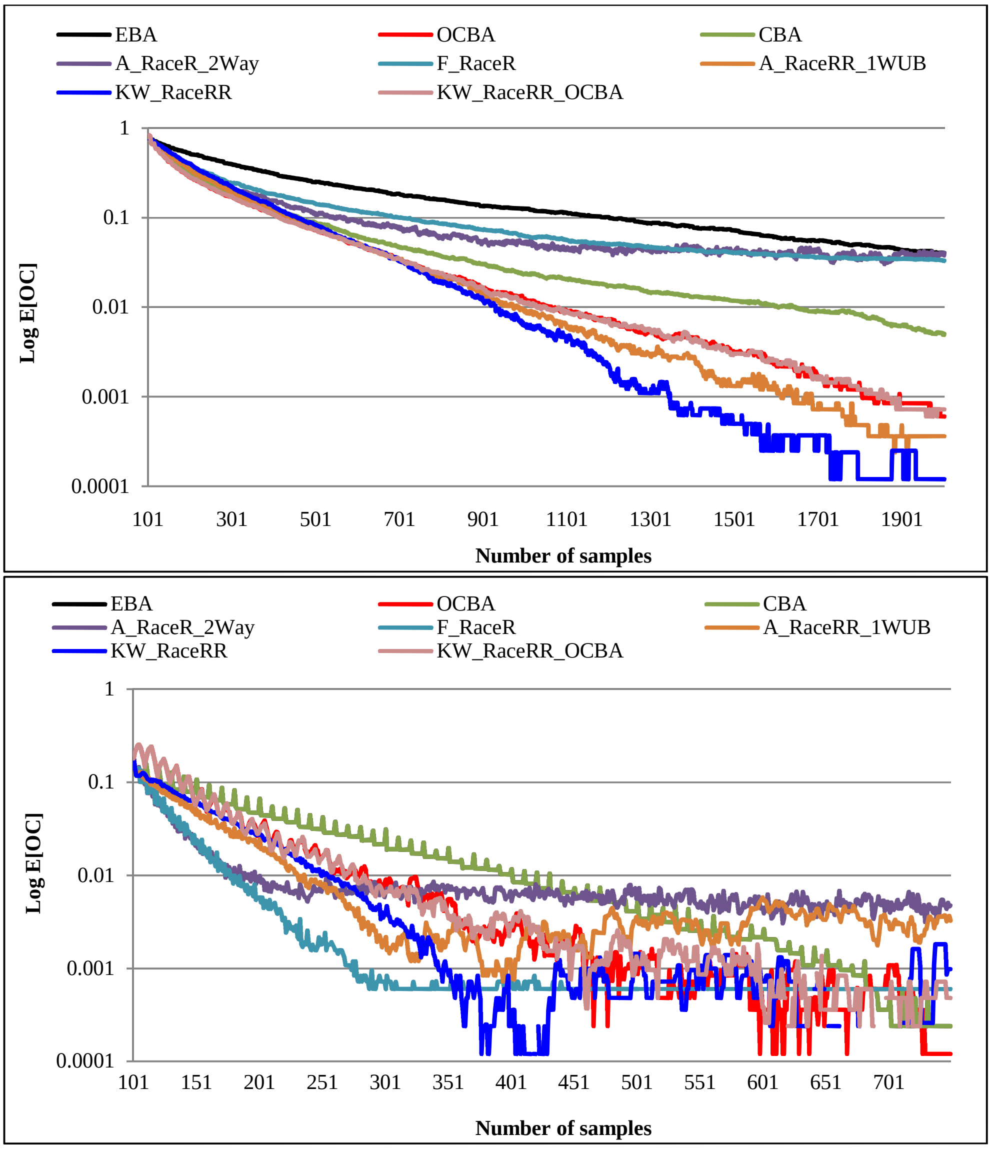
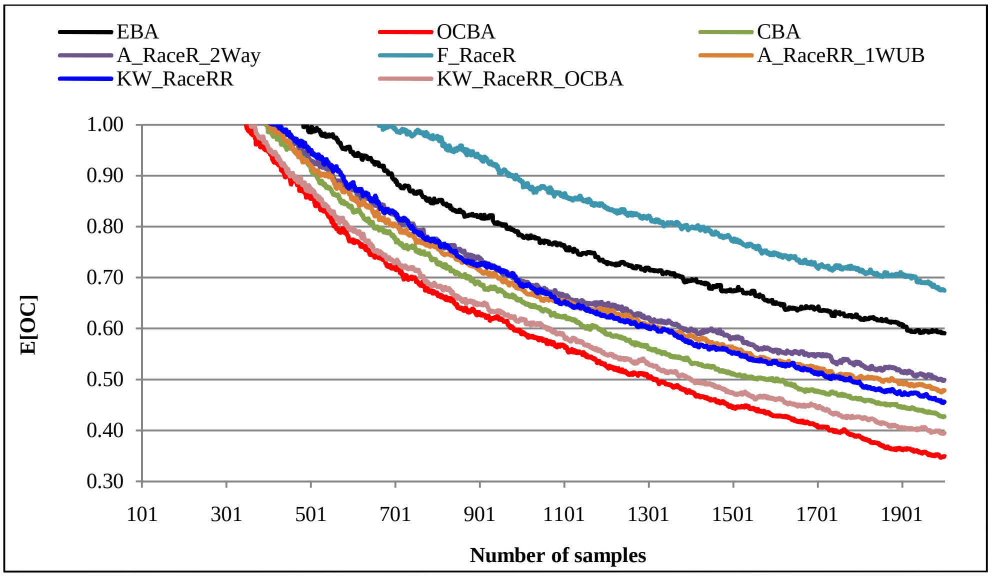


Figure A3.1. *E[OC]* for 0.0 (above) and 0.9 (below) correlation. Case 1.



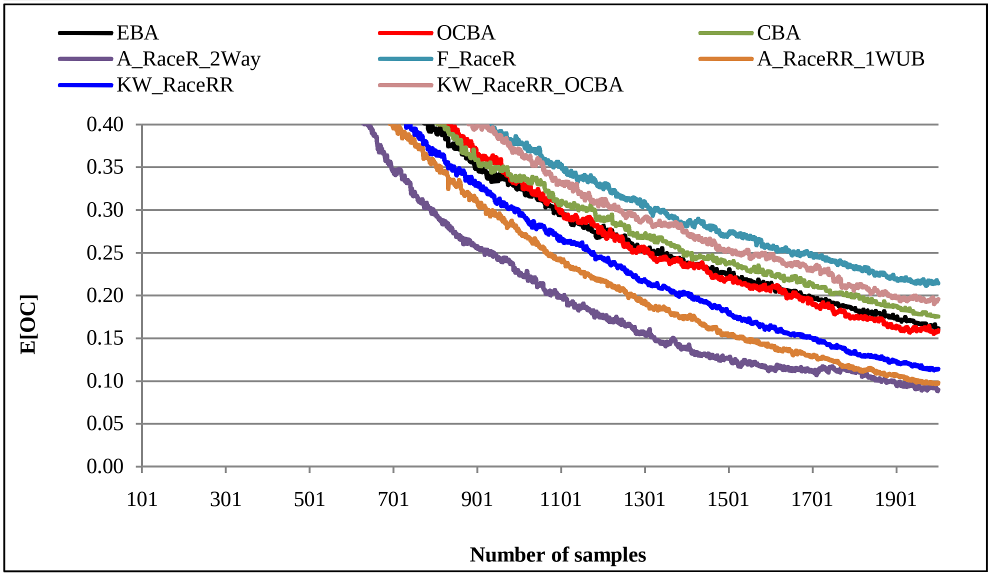


Figure A3.2. *E[OC]* for 0.0 (above) and 0.9 (below) correlation. Case 2.

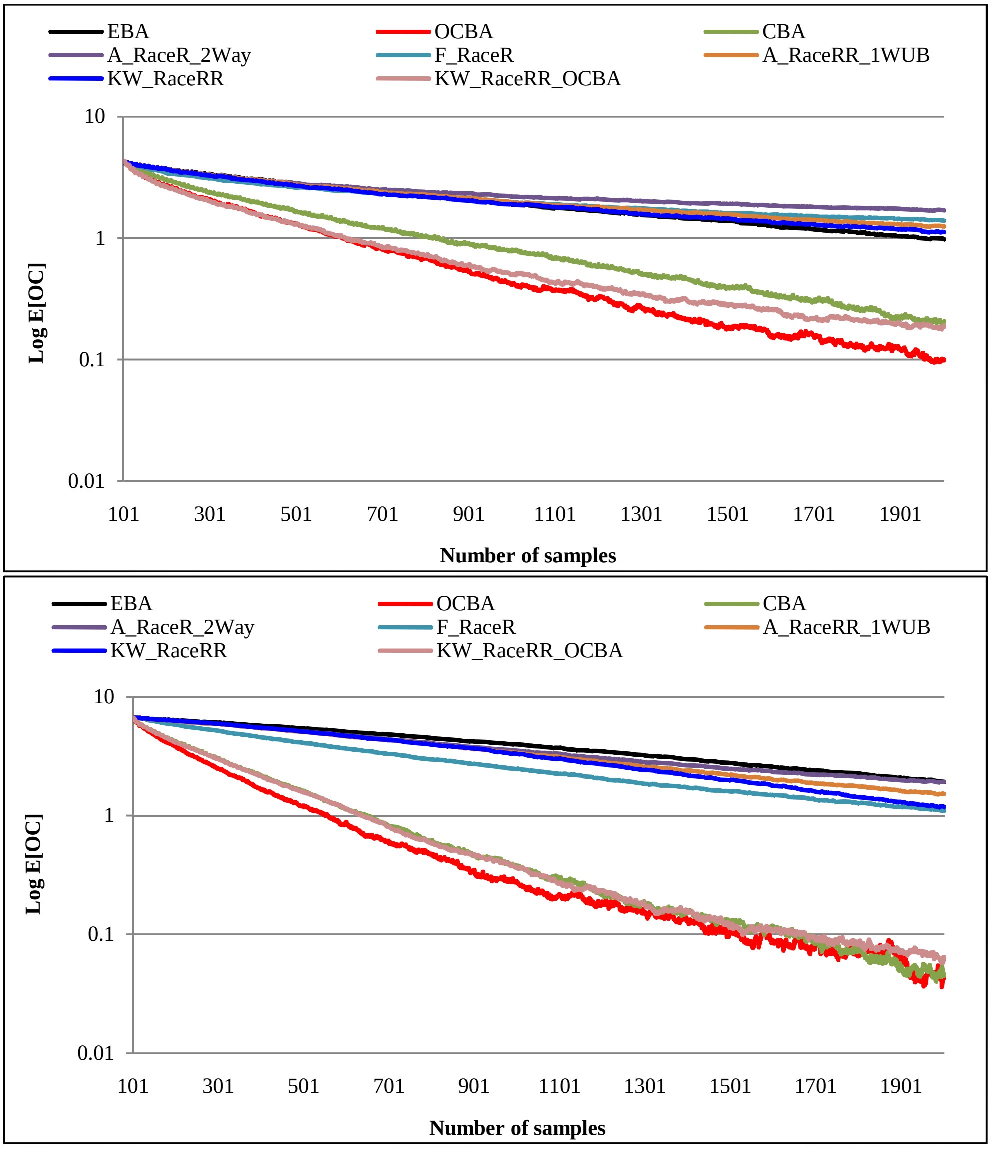


Figure A3.3. *E[OC]* for 0.0 (above) and 0.9 (below) correlation. Case 3.

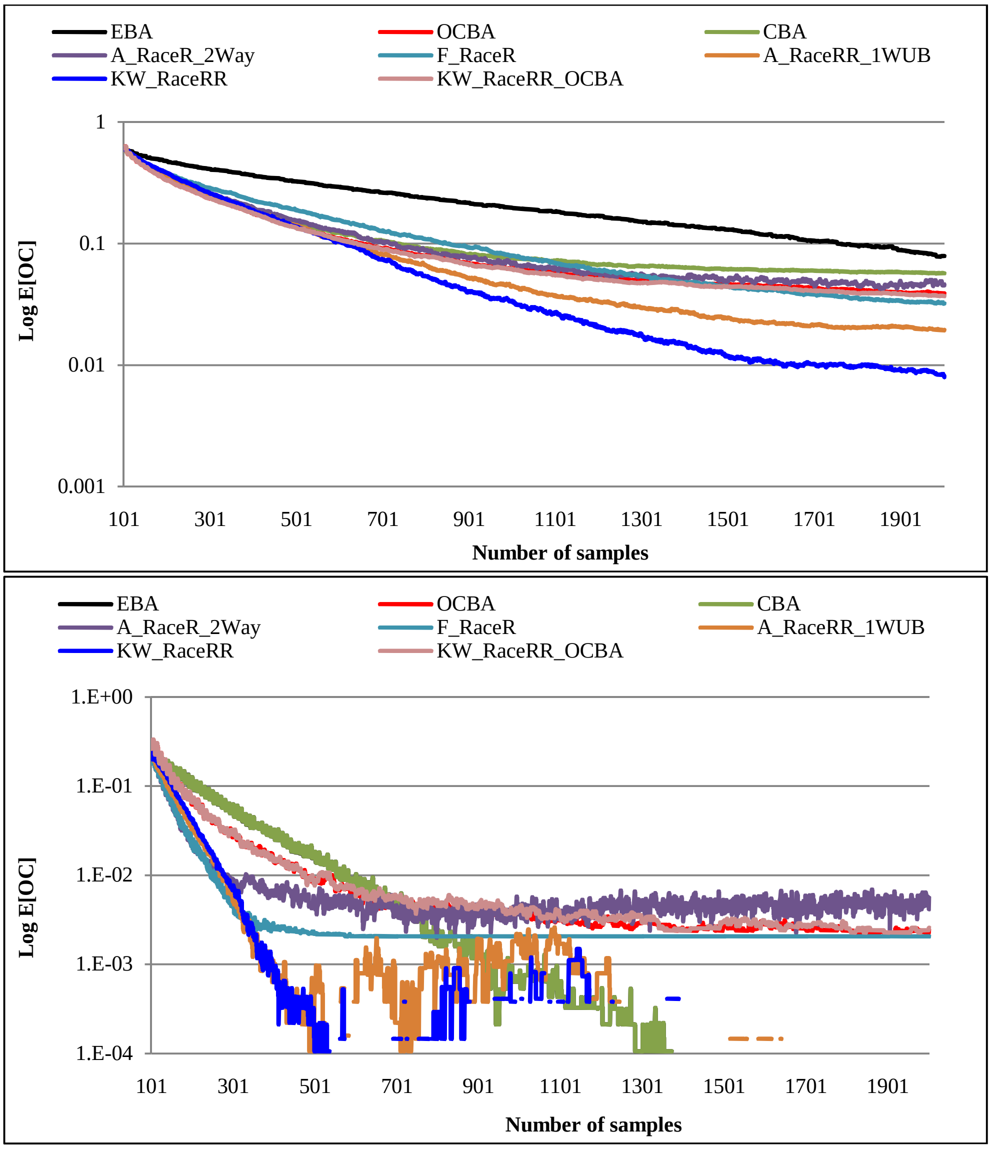


Figure A3.4. *E[OC]* for 0.0 (above) and 0.9 (below) correlation. Case 4, *Set*-A.

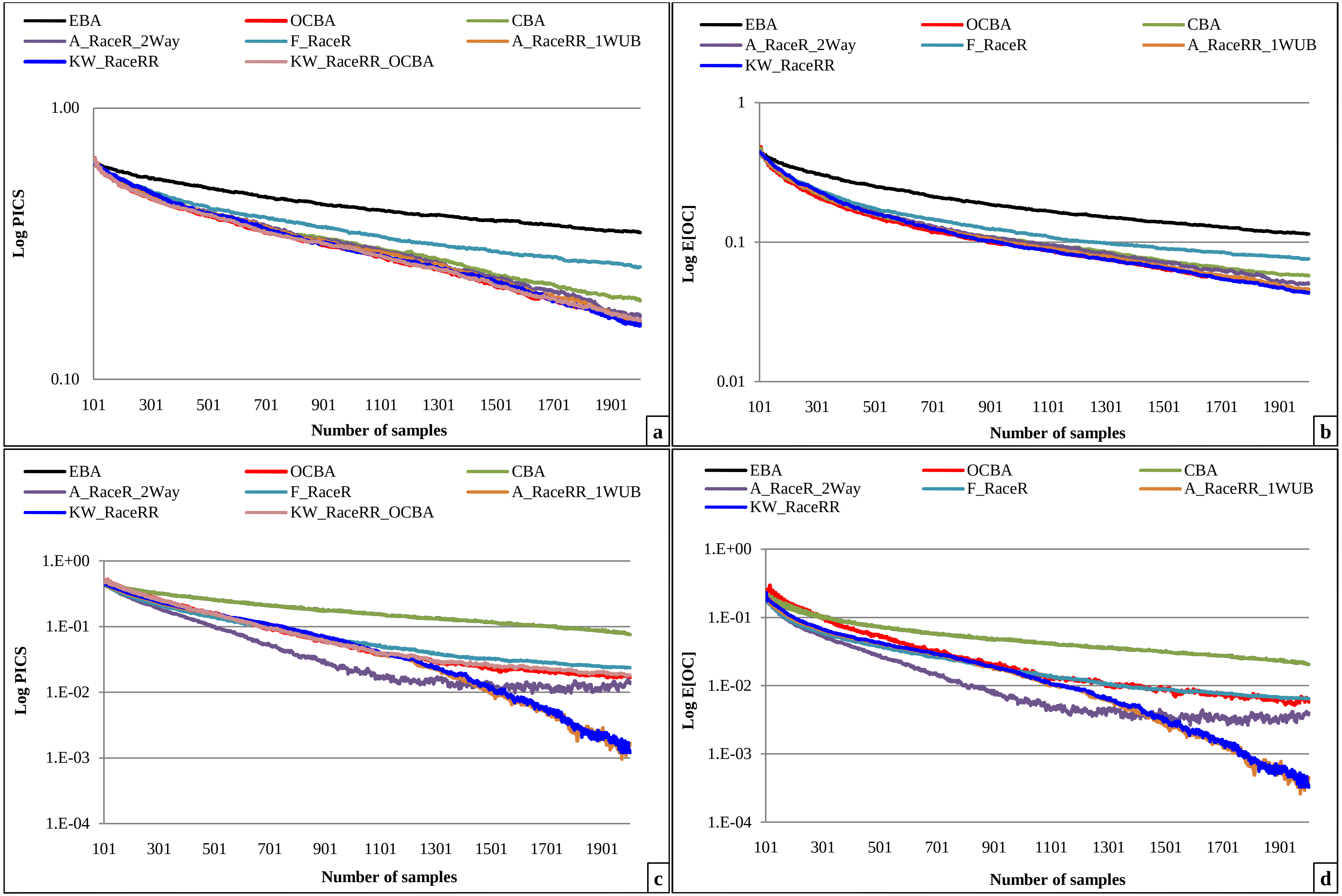


Figure A3.5. Comparison of different algorithms based on *PICS* and *E[OC]* under 0.0 (a and b) and 0.9 (c and d) correlation. $\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$ *Set*-B.

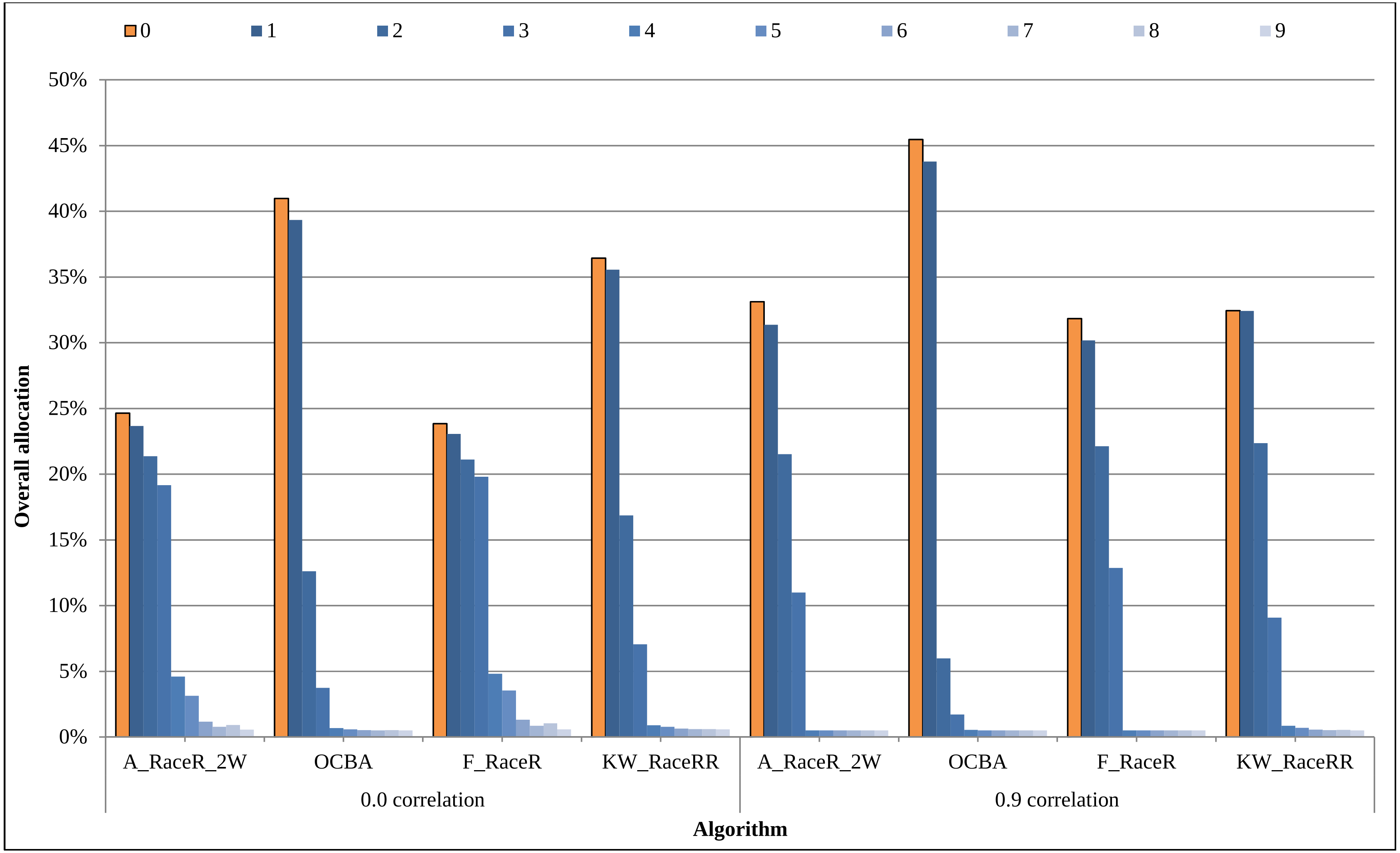


Figure A3.6. Overall allocation of different algorithms at 0.0 and 0.9 correlation. $\sim\mathcal{N}\left(U(0,k),U(10{,}24)\right)$ *Set*-B. The numbers are averaged over all replications.

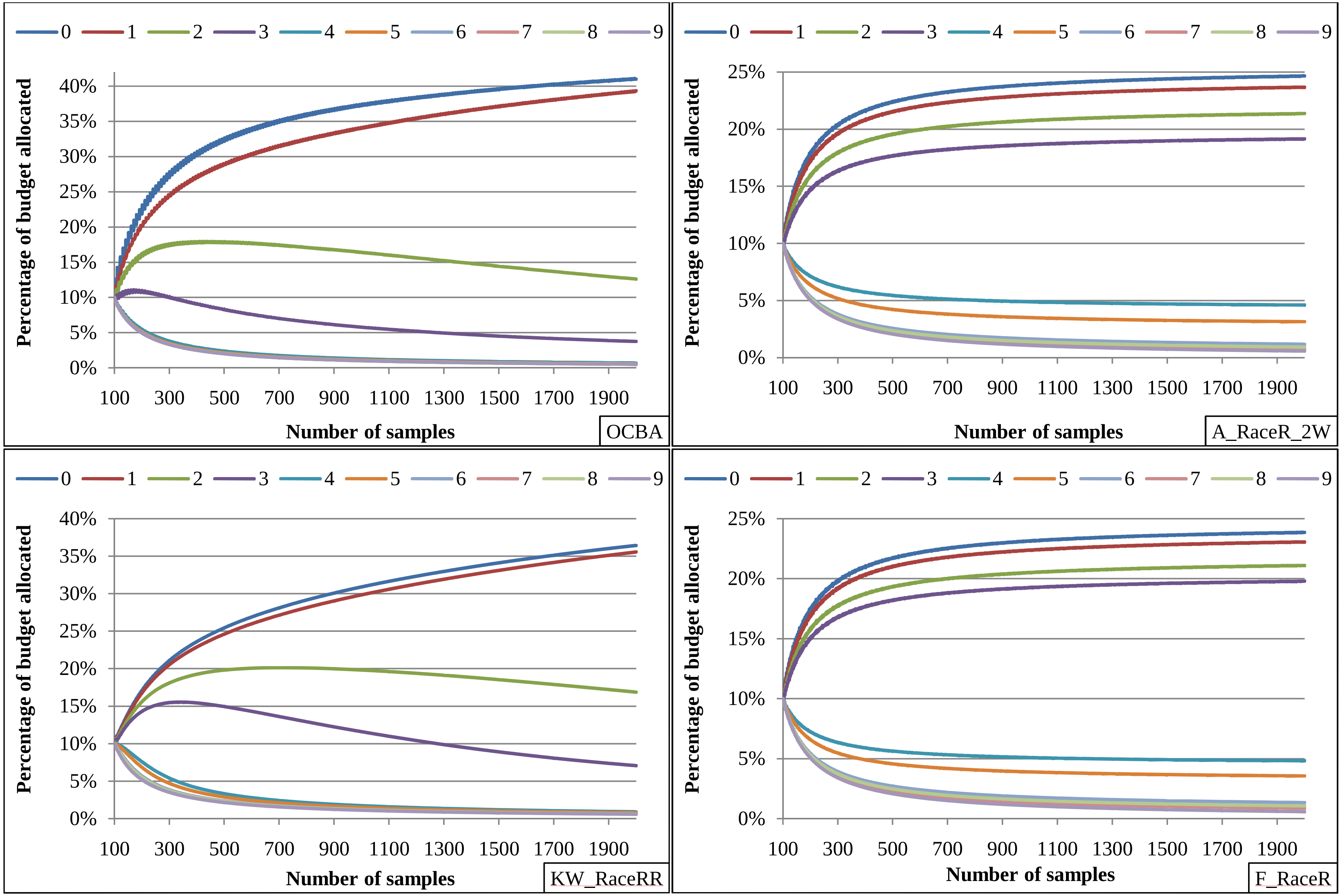


Figure A3.7. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}\big(U(0,k),U(10{,}24)\big)$ *Set*-B. Correlation = 0.0.

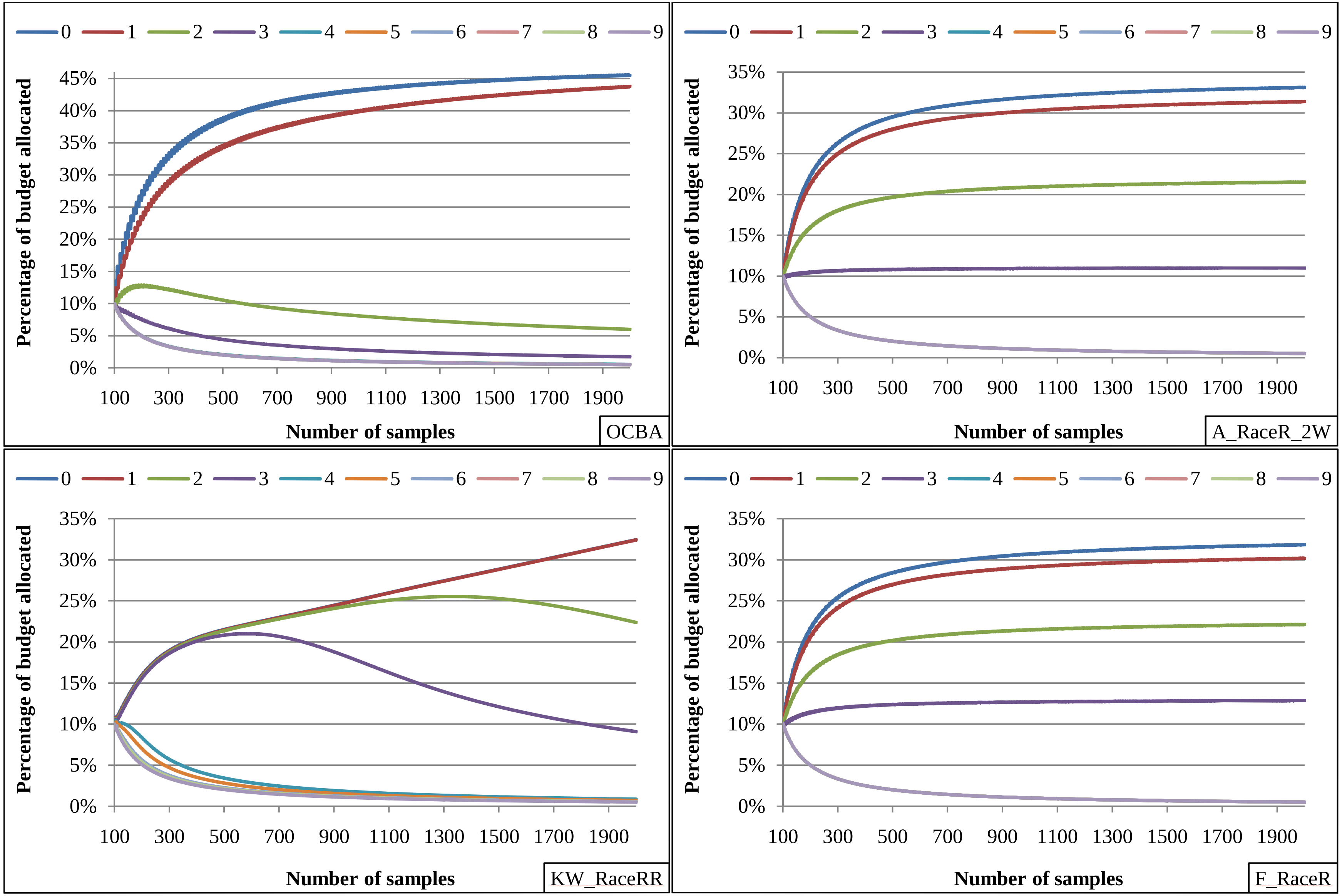


Figure A3.8. Cumulative budget allocated to each system throughout the run, averaged over all replications. $\sim\mathcal{N}\big(U(0,k),U(10,24)\big)$ *Set*-B. Correlation = 0.9.

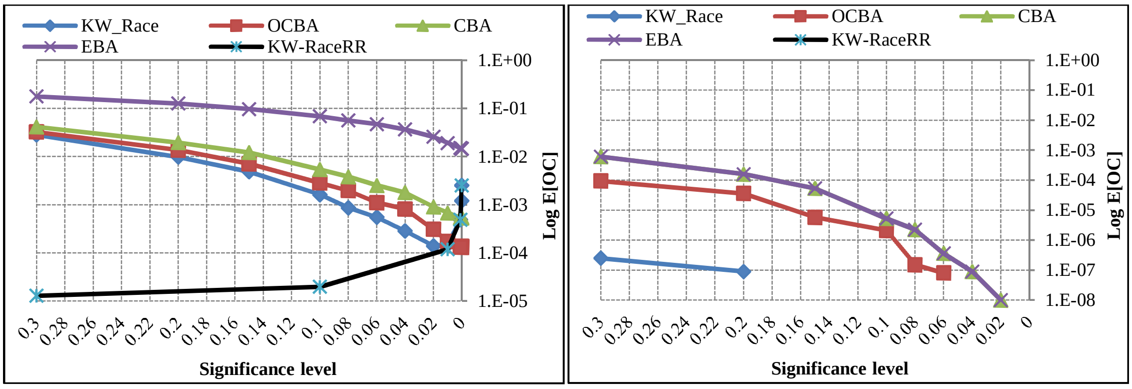


Figure A3.9. *E[OC]* at 0.0 (left) and 0.9 (right) correlation for Case 1.

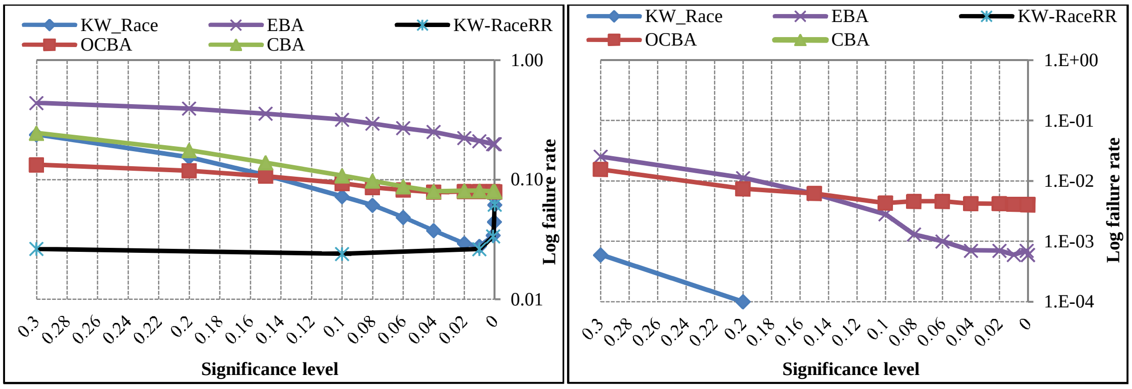


Figure A3.10. *E[OC]* at 0.0 (left) and 0.9 (right) correlation for Case 4 *Set*-A.

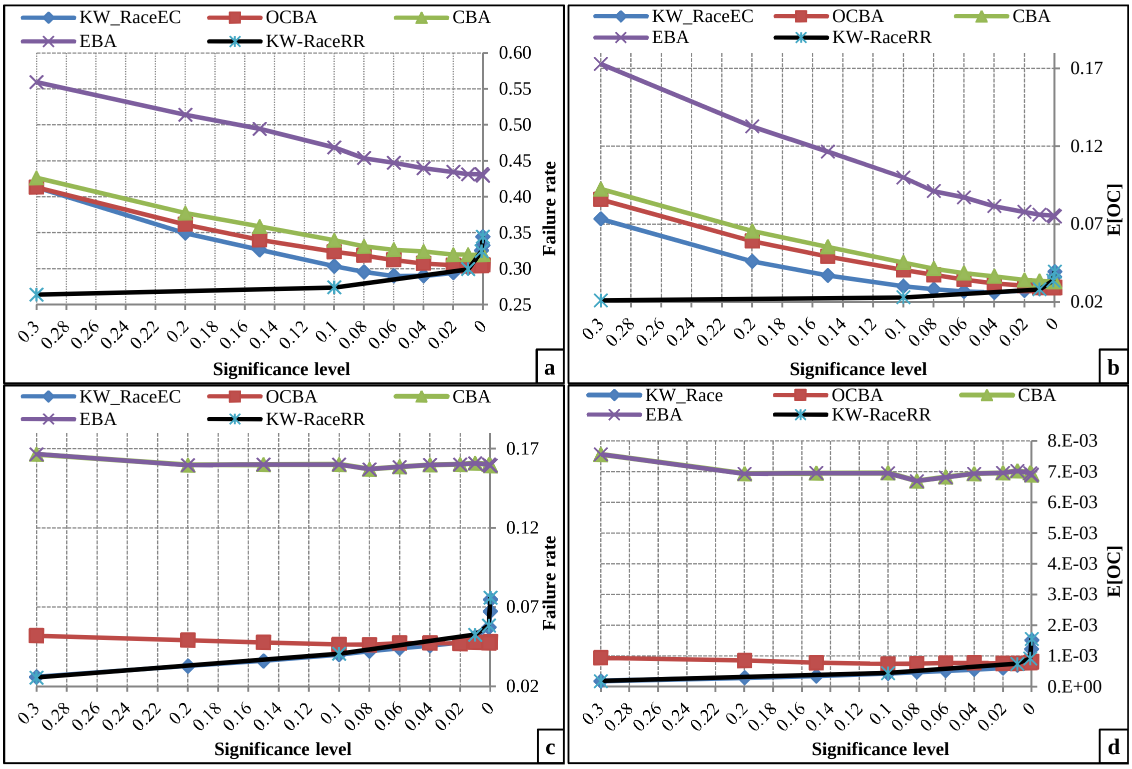


Figure A3.11. ECs based on f_r and *E[OC]* at 0.0 (a and b) and 0.9 (c and d) correlation levels for Case 4 *Set*-B.

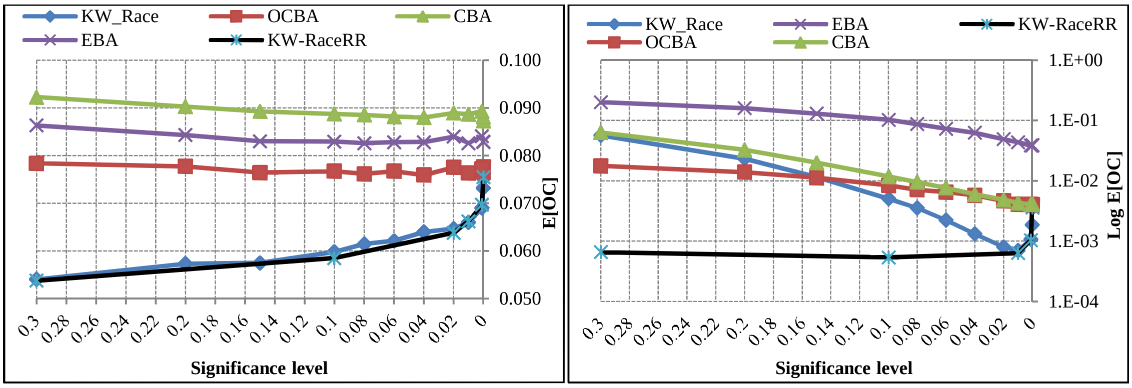


Figure A3.12. *E[OC]* at 0.0 (left) and 0.9 (right) correlation for Case 3.

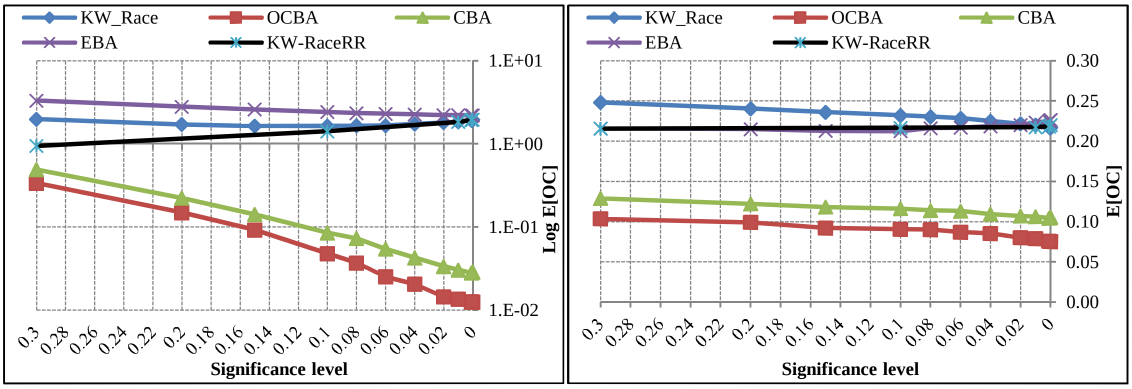


Figure A3.13. *E[OC]* at 0.0 (left) and 0.9 (right) correlation for Case 4.