# Statistical Racing Techniques for Improved Empirical Evaluation of Evolutionary Algorithms

Bo Yuan and Marcus Gallagher

School of Information Technology and Electrical Engineering,
University of Queensland, QLD 4072, Australia
{boyuan, marcusg}@itee.uq.edu.au

**Abstract.** In empirical studies of Evolutionary Algorithms, it is usually desirable to evaluate and compare algorithms using as many different parameter settings and test problems as possible, in order to have a clear and detailed picture of their performance. Unfortunately, the total number of experiments required may be very large, which often makes such research work computationally prohibitive. In this paper, the application of a statistical method called racing is proposed as a general-purpose tool to reduce the computational requirements of large-scale experimental studies in evolutionary algorithms. Experimental results are presented that show that racing typically requires only a small fraction of the cost of an exhaustive experimental study.

## 1 Introduction

Metaheuristic optimization methods such as Evolutionary Algorithms (EAs) are commonly evaluated and compared using empirical methods, due to the complexity of the dynamics and the problems to which they are applied. Due to many constraints, researchers often perform limited empirical studies where candidate algorithms with hand-tuned parameters are tested on a small set of benchmark problems. The shortcomings of this kind of methodology have been pointed out[1-3]. For example, parameter settings may often have significant influence on the performance of EAs and finding good parameter values can itself be a difficult optimization problem. Also, benchmark problems are often selected arbitrarily, and since there is typically no relationship between these problems, it is dangerous to make general conclusions about performance on the basis of such results.

A more principled way to evaluate EAs empirically is to systematically explore a well-defined experimental space over algorithm parameter values and problems of interest. Unfortunately, an exhaustive or brute force approach quickly becomes computationally prohibitive, typically as the result of an explosion in the size of the space when experiments are scaled up. In this paper, a statistical technique called racing [4, 5] is proposed as one tool that can be applied to allow researchers to expand their empirical studies, by significantly reducing the computational requirements over a large experimental space.

The content of this paper is structured as follows. The next section presents the framework of racing and some details of the statistical tests. Section 3 specifies

algorithms and problems that create the experimental space to be investigated. A set of racing experiments are conducted in Section 4 to justify the usefulness of racing. Section 5 concludes our work and points out some directions of further work.

## 2 Racing

### 2.1 An Overview

Racing algorithms[4, 5] have recently been proposed to solve the model selection problem in Machine Learning: given a set of data points and a number of candidate lazy learning algorithms (which could include multiple versions of some algorithm with different, specified parameter values), which algorithm yields the minimum prediction error based on leave-one-out cross validation? In contrast to a brute force method, which is to sequentially evaluate all algorithms on all available data points and choose the best performing algorithm, the racing method investigates all algorithms in parallel. In each step, all algorithms are tested on a single independently selected data point and their prediction errors on that point are calculated. The mean predication error of each algorithm on data points that have already been seen is also maintained. This error, $E_{est}$, is an estimation of the true prediction error $E_{true}$ over the entire data set. As the algorithms are tested on more and more data points, $E_{est}$ approaches $E_{true}$. The fundamental mechanism of racing attempts to identify and eliminate weak candidates on the basis of $E_{est}$ as early as possible to minimize the number of unnecessary predication queries. Candidates compete with each other for computational resources and only promising candidates survive to undertake further testing. There are several possible ways of deciding if and when a candidate should be eliminated, based on statistical tests.

### 2.2 Statistical Tests in Racing

In Hoeffding races[4], the upper and lower boundaries of $E_{true}$, called the worst possible error and the best possible error respectively, which specify the confidence intervals of $E_{true}$, are estimated at each step based on $E_{est}$, the number of data points that have been seen, the confidence level and the greatest possible error[6]. If at some stage, the best possible error of any algorithm is worse than the worst possible error of the current best algorithm, this algorithm will be eliminated. The advantage of Hoeffding races is that no assumptions are made about the distribution of the data, which makes it applicable to a wide range of situations.

Candidate algorithms can be eliminated more quickly and/or more reliably if the data is approximately normally distributed[5]. In this case, each candidate has a population of errors and statistical methods such as ANOVA (Analysis of Variance) can be utilized to determine if the means of these populations are significantly different. In the meantime, since there is often some kind of correspondence relationship among different groups of data, it is also possible to use methods like the Friedman test[7], which employs a block design[8] to eliminate unwanted sources of variability. The data set in the Friedman test is a $b \times k$ matrix where $b$ is the number

of blocks and $k$ is the number of candidates. Each block contains the experimental results of all candidates on a single test instance and each column contains the experimental results of a single candidate on all test instances. The framework of racing algorithms based on ANOVA or the Friedman test is given in Table 1.

**Table 1.** The framework of racing algorithms based on ANOVA or the Friedman test

| Repeat following steps until only one candidate left or no more unseen instance<br>• Randomly select an unseen instance and test all remaining candidates on it<br>• Store results in corresponding performance populations<br>• If no difference in the means of performance populations is detected by ANOVA or the Friedman test, continue<br>• Conduct multiple comparison of means and delete candidates if they are significantly worse than others at predefined significance level |
|---|

### 2.3 Racing Evolutionary Algorithms

There is a similarity between the model selection problem in Machine Learning and the task of parameter tuning in EAs. In each case, the user is faced with a meta-optimization problem: to find values for all of the adjustable parameters of the model (algorithm) in order to produce the best results when the algorithm is applied to the original problem. Since racing methods only utilize the statistics of the results generated by sets of algorithms, they should also be applicable to parameter tuning in EAs. In fact the set of algorithms to be raced need not to be different instances of the same algorithm. Racing can be used in quite a general sense in an attempt to reduce the experimental effort required, when a comparison is required over a range of experimental configurations.

We are aware of only one application of racing to the experimental evaluation of EAs[9]. Birattari et al employ a racing algorithm to find a parameter configuration of an Ant Colony Optimization (ACO) algorithm that performs as well as possible on a set of instances of the Traveling Salesman Problem (TSP). The TSP instances were randomly generated, which can be regarded as a set of test points drawn from a specific problem space. The parameters of the algorithm were systematically varied to create a set of fully-specified candidate algorithm instances. A racing algorithm called F-race based on the Friedman test was employed to find an as good as possible algorithm instance within a limited amount of time. The experimental results show that F-race outperformed two other racing algorithms based on the paired t-test. Also, there was no significant difference between the results produced by F-race and results produced by a limited brute-force method.

In this paper we aim to clarify the use of racing algorithms in a more general experimental scenario, and to show some different ways in which racing can be applied (e.g., across random restarts of algorithms, or to select for problems instead of algorithms). We also provide a comparison between the racing algorithms and an exhaustive method in each case, and examine the influence of various experimental factors on the effectiveness of racing.

## 3 Algorithms and Problems

### 3.1 Algorithm Framework

The algorithm presented below is within the framework of Estimation of Distribution Algorithms (EDAs)[10]. The basic idea of EDAs is to estimate the probability distribution of a few selected individuals in each generation and all new individuals are generated by sampling from this probability distribution.

**A Gaussian Kernel EDA**

**Step 1**: Initialize population P by randomly generating N individuals
**Step 2**: Evaluate all individuals
**Step 3**: Chose M best individuals as kernels
**Step 4**: Create P' by sampling N individuals from the kernel density estimator
**Step 5**: Evaluate all new individuals in P'
**Step 6**: Combine P and P' to create a new population
**Step 7**: Go to Step 3 until Stop

The general form of the kernel density estimator is given by Eq. 1:

$$p(x) = \frac{1}{M} \sum_{i=1}^{M} K(x, x_i) \tag{1}$$

In this paper, we use a spherical Gaussian kernel function $K$ and the probability distribution is estimated by a Gaussian kernel density estimator, with kernels placed over the selected individuals themselves (i.e., $x_i$). In this model, the value of the standard deviation σ, which is a smoothing parameter, plays an important role in the model's performance [11]. If the value is too small, the model may tend to overfit the data and a very smooth estimation may be generated with a very large value, which may not be able to reveal some structure details. The value of M is another important parameter to determine, which controls the trade off between exploration and exploitation and is usually set based on some kind of rule of thumb. In fact, this algorithm can be regarded as an ES with truncation selection and when M is equal to N, it will work as a standard (μ+λ) ES(See[12] for an analysis of the connection between EDAs and ESs).

### 3.2 Landscape Generator

For some of the experiments in the following section we use a continuous problem/fitness landscape generator as a source of test problems[13]. The landscape generator provides a source from which a large number of (randomized) problem instances (i.e., to be maximized) can be produced. It is based on a sum of Gaussian functions and is parameterized by a small number of parameters such as the number of Gaussians, their locations and covariance structure. These parameters have a direct

influence on the structure of the fitness landscapes generated, and several interesting properties of the landscapes (e.g., the values of local and global optima, the number and distribution of optima) can be controlled and smoothly varied via the parameters.

For results presented below, the parameters of the landscape generator are set to the fixed values specified, to produce a set of multimodal functions.

## 4 Experiments

### 4.1 Racing Multiple Algorithms on a Single Problem

In this section we apply racing to the task of running a set of specified algorithms on a single test problem. The performance of most EAs is stochastic and depends on their initialization. In empirical studies, this is normally accounted for by running an algorithm repeatedly from different initial conditions. Racing can be applied to reduce the total number of algorithms in this scenario. For this purpose, we chose a well-known benchmark problem: Rastrigin's Function, which is a minimization problem.

$$F(x,y) = 20 + x^2 + y^2 - 10 \cdot (\cos 2\pi \cdot x + \cos 2\pi \cdot y) \qquad x, y \in [-5,5] \tag{2}$$

Fifty instances of the Gaussian kernel algorithm were tested with varying values for two of the algorithm parameters: δ from 0.2 to 2.0 with step size 0.2 and the value of M from 10 to 50 with step size 10. To evaluate the performance of the racing algorithms, an exhaustive set of experiments was conducted. Each algorithm was run on the problem for 50 trials (i.e., population size=50, maximum number of generations=100) and the best solution found in each trial was recorded. The performance of all algorithms (in terms of this best solution fitness value) is summarized using boxplots with the horizontal axis nominally representing algorithms and the vertical axis representing performance distributions among restarts (Fig.1). It is possible to inspect these results visually and determine, for example which algorithms were able to reliably find a good solution value.

Using racing, we aim to make a similar observation at a fraction of the computational effort required for the above exhaustive experiment. The procedure operates here by eliminating poor performing algorithms on the basis of a smaller number of random restarts. Two racing algorithms called F-races and A-races based on the Friedman test and ANOVA respectively were applied at the significance level α=0.05. Since the sequence of test instances may have more or less influence on the performance of racing, in this paper, each racing algorithm was run for 10 trials with random sequences. The average number of remaining algorithms during racing is shown in Fig. 2 from which it is clear that both methods eliminated a large number of algorithms after around 15 restarts. The efficiency of racing corresponds to the ratio between the area below the curve and the area of the whole rectangular area in the figure. From a quantitative point of view, the average costs of F-races and A-races were 13.98% and 20.43% of the cost of the brute force method. Note that the single best algorithm in terms of mean performance from the exhaustive experiment (No.2) was always among the remaining algorithms at the end of racing.

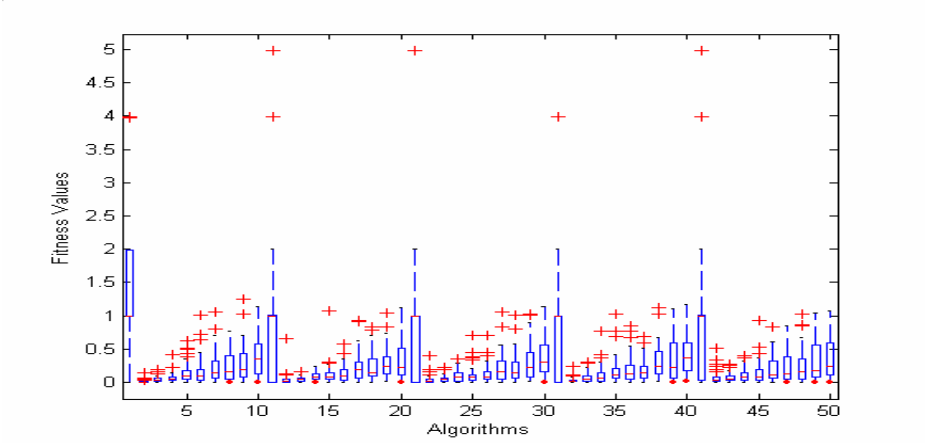


**Fig. 1.** Performance distributions of 50 algorithms on Rastrigin's Function

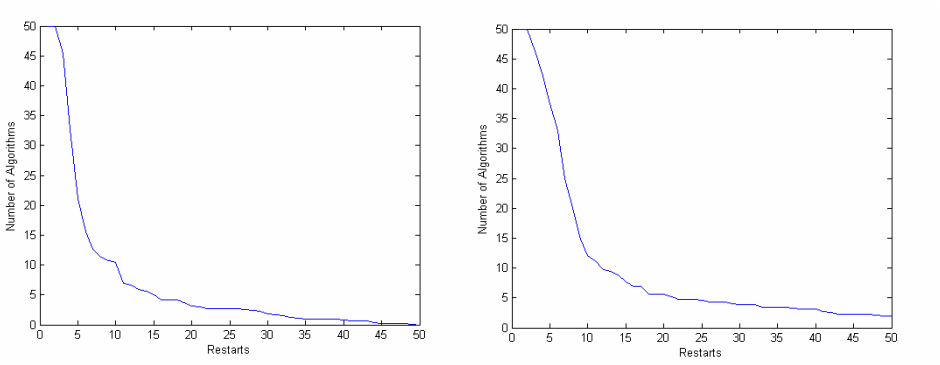


**Fig. 2.** Number of remaining algorithms vs. restarts: F-races (left) and A-races (right)

### 4.2 Racing a Single Algorithm on Multiple Problems

Researchers are often interested in qualifying the performance of a single algorithm on a variety of test problems. In this section we demonstrate how racing algorithms can be used to efficiently identify (for example) which problems from a given set are more difficult for a single algorithm. We selected arbitrarily a Gaussian kernel-based algorithm from above with $\delta$=0.2 and M=20. The landscape generator described in Section 3.2 was used to generate fifty 5-D random landscapes. Each landscape contained 10 Gaussians, which generally corresponds to 10 optima (though this may vary due to overlaps in the random locations of the components of the landscape). The global optimum in each landscape had fitness value 1.0 and the fitness values of other local optima were randomly generated between 0 and 0.8. The algorithm was first run exhaustively on each landscape for 50 trials (population size=50, maximum number of generations=50) and the best solution found in each trial was recorded (Fig. 3).

The racing algorithms (F-races and A-races) were again applied to this task (attempting to remove problems from experimentation iterating over restarts of the algorithm) and the results are shown in Fig. 4. Note that at the end of the experiment, the problems that have not been eliminated during racing correspond to a much smaller set of problems that are seemingly difficult for this algorithm to perform well on. The average costs of F-races and A-races were 30.63% and 36.00% respectively compared to the brute force method (Fig. 4). The most difficult problem(No.1) was among the remaining problems by the end of racing in 9 out of 10 trials.

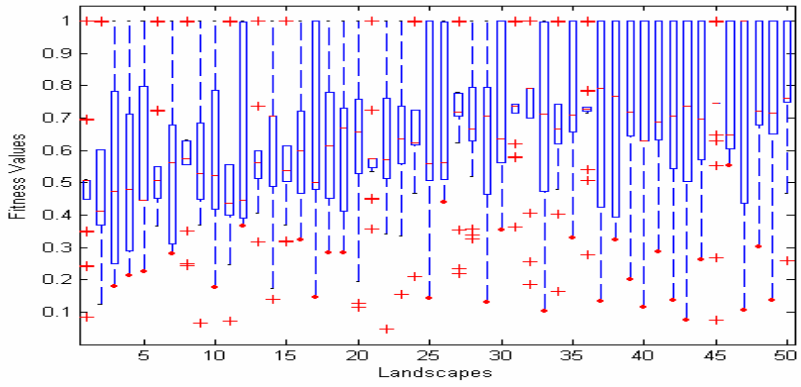


**Fig. 3.** Difficulty distributions of 50 problems on one algorithm (sorted based on mean values)

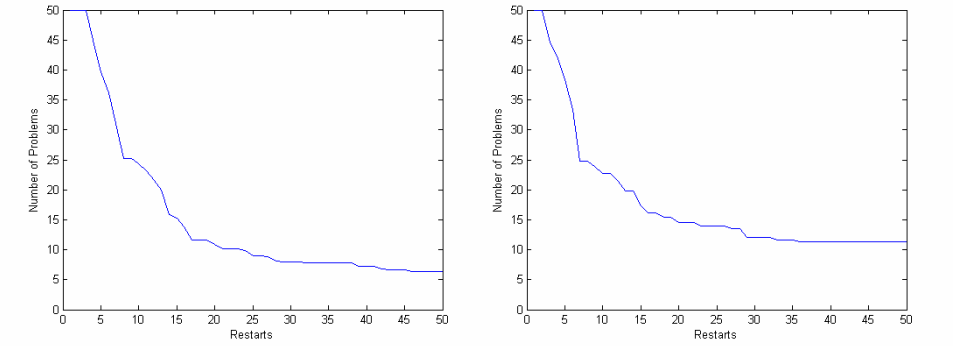


**Fig. 4.** Number of remaining problems vs. restarts: F-races (left) and. A-races (right)

### 4.3 Racing Multiple Algorithms on Multiple Problems

Finally, racing was applied to identifying the best performing algorithms for a set of problems and to identifying the most difficult problems for a set of algorithms. Note that in the experiments in Sections 4.1 and 4.2 racing was carried out with respect to restarts of an algorithm on a problem. In order to carry out racing with respect to different algorithms or problems, the performance of the algorithm across restarts

must be summarized. Birattari et. al appear to conduct racing by running each algorithm only once on a problem – this may lead to sensitive or unrepresentative results in general. In the following experiments, each algorithm was run on each problem for 10 trials and the average value was recorded (population size=200, number of generations=100). One hundred algorithm instances were generated by systematically varying the two parameters of the algorithm (δ from 0.2 to 4.0 with step size 0.2 and M from 20 to 100 with step size 20). Fifty problems were generated in the same way as in Section 4.2.

The results of the exhaustive experiments are shown in Figs. 5&6 over algorithms and problems respectively. The average costs of F-races and A-races in finding the best algorithm were 13.68% and 45.76% respectively (Fig. 7) and the average costs in finding the most difficult problem were 3.58% and 3.05% respectively (Fig. 8), compared to the brute force method. Again, the best algorithm (No. 86) and the most difficult problem (No.1) were never eliminated.

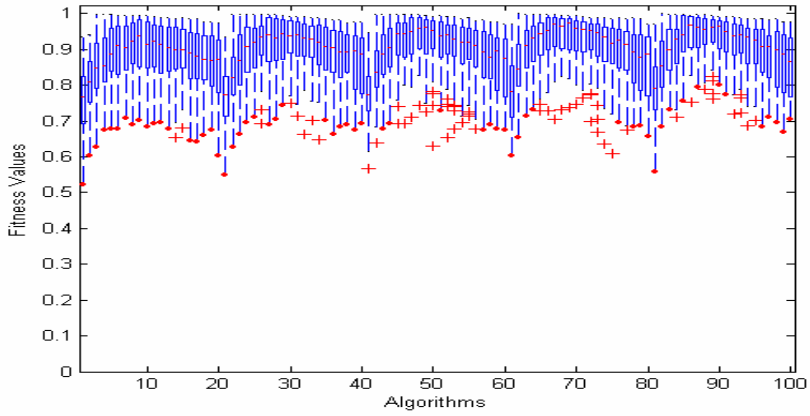


**Fig. 5.** Performance distributions of 100 algorithms on 50 landscapes

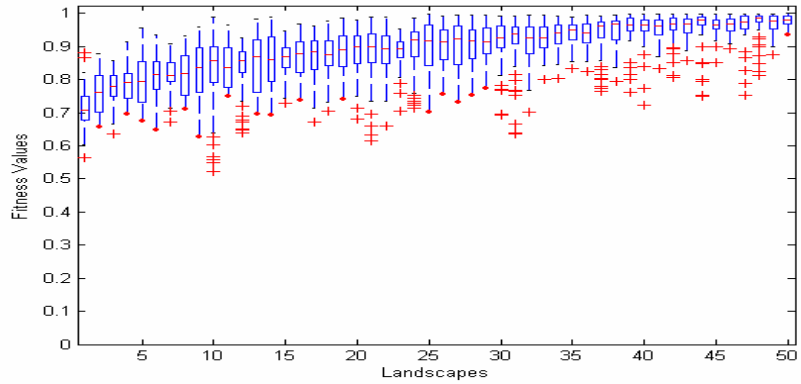


**Fig. 6.** Difficulty distributions of 50 problems on 100 algorithms (sorted based on mean values)

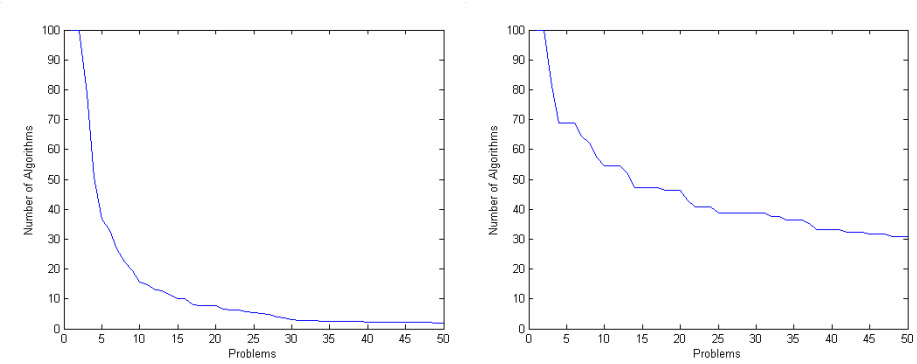


**Fig. 7.** Number of remaining algorithms vs. problems: F-races (left) and A-races (right)

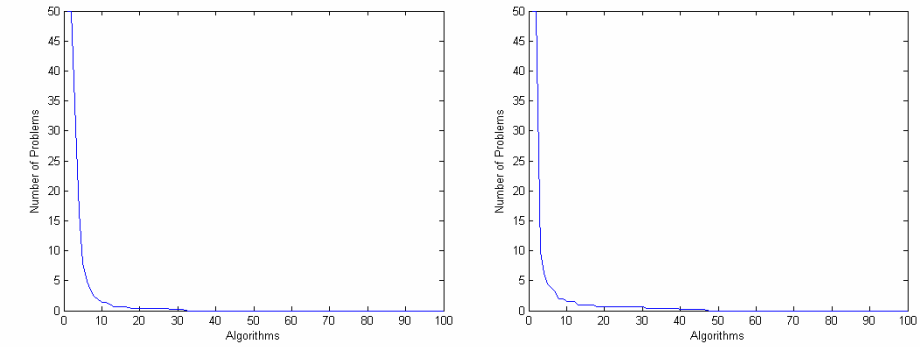


**Fig. 8.** Number of remaining problems vs. algorithms: F-races (left) and A-races (right)

## 5 Conclusion

It is clear that for each of our experiments, racing provides significant computational benefit over an exhaustive set of experiments. The F-races algorithm was observed to be more effective at eliminating candidates from experimentation than A-races. Note however that the F-races method is based on ranks instead of raw fitness values, which means that it actually works on a different performance measure.

The effectiveness of racing in some sense needs to be evaluated according to the aims of the experimenter. In the above, racing almost never incorrectly eliminated the single best candidate (as verified by the exhaustive experiments). The number of candidates remaining at the end of the racing procedure is dependent on the significance level ($\alpha$), the statistics of the experiments and the stopping time. In some of the experiments above, racing could only eliminate a relatively small number of candidates (e.g., Fig. 4) while in other cases most of candidates could be eliminated after a few test instances (e.g., Fig. 8). The worst performance of both racing algorithms occurred where the experimental data had large variance (Fig. 3), which means that the performance of many candidates varied greatly from test instance to test instance. Also, when the performance of the candidates was very similar to each

other (Fig. 5), A-races could not distinguish them because they had very close mean values (Fig. 7 right). However, F-races (based on the block design and ranks) showed much better performance in the same situation (Fig. 7 left). As a contrast, in the final set of experiments (Fig. 8) both racing algorithms worked extremely well. Examination of the experimental data set (Fig. 6) shows that the most difficult problem was significantly more difficult than others and the variance of each group of data was relatively small.

In summary, this paper has shown that racing algorithms represent a promising tool for increasing the capacity and efficiency of empirical research in EAs. We expect that the use of statistical methods will play an important role in the improvement of standard and practice in the empirical evaluation of EAs and other metaheuristics.

## Acknowledgement

This work was supported by the Australian Postgraduate Award granted to Bo Yuan.

## References

1. Whitley, D., Mathias, K., Rana, S. and Dzubera, J.: Evaluating Evolutionary Algorithms. *Artificial Intelligence*, **85**(1-2): pp. 245-276, 1996.
2. De Jong, K.A., Potter, M.A. and Spears, W.M.: Using Problem Generators to Explore the Effects of Epistasis. In *Seventh International Conference on Genetic Algorithms*, T. Bäck Ed., Morgan Kauffman, pp. 338-345, 1997.
3. Eiben, A.E. and Jelasity, M.: A Critical Note on Experimental Research Methodology in EC. In *Congress on Evolutionary Computation*, Hawaii, IEEE, pp. 582-587, 2002.
4. Maron, O. and Moore, A.W.: Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. In *Advances in Neural Information Processing Systems 6*, J.D. Cowan, et al., Editors, pp. 59-66, 1994.
5. Maron, O. and Moore, A.W.: The Racing Algorithm: Model Selection for Lazy Learners. *Artificial Intelligence Review*, **11**: pp. 193-225, 1997.
6. Hoeffding, W.: Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, **58**(301): pp. 13-30, 1963.
7. Conover, W.J.: Practical Nonparametric Statistics. 3rd ed, John Wiley & Sons, Inc., 1999.
8. Box, G.E.P., Hunter, W.G. and Hunter, J.S.: Statistics for Experimenters. Wiley, 1978.
9. Birattari, M., Stutzle, T., Paquete, L. and Varrentrapp, K.: A Racing Algorithm for Configuring Metaheuristics. In *Genetic and Evolutionary Computation Conference (GECCO 2002)*, pp. 11-18, 2002.
10. Mühlenbein, H. and Paaß, G.: From Recombination of Genes to the Estimation of Distributions: I. Binary Parameters. In *Parallel Problem Solving from Nature IV*, H.-M. Voigt, et al. Eds., pp. 178-187, 1996.
11. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, 1995.
12. Gallagher, M.: Multi-Layer Perceptron Error Surfaces: Visualization, Structure and Modelling. PhD Thesis, The University of Queensland, 2000.
13. Yuan, B. and Gallagher, M.: On Building a Principled Framework for Evaluating and Testing Evolutionary Algorithms: A Continuous Landscape Generator. In *the 2003 Congress on Evolutionary Computation*, pp. 451-458, 2003.