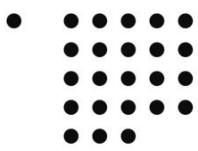
Schriftenreihe CIplus, Band 3/2012

Thomas Bartz-Beielstein, Wolfgang Konen, Horst Stenzel, Boris Naujoks

Beyond Particular Problem Instances: How to Create Meaningful and Generalizable Results

Thomas Bartz-Beielstein



Fachhochschule Köln
Cologne University of Applied Sciences

Beyond Particular Problem Instances: How to Create Meaningful and Generalizable Results

Thomas Bartz-Beielstein

www.spotseven.de

Faculty for Computer and Engineering Sciences
Cologne University of Applied Sciences,
51643 Gummersbach, Germany
Schriftenreihe CIplus
TR 3/2012. ISSN 2194-2870

Abstract. Computational intelligence methods have gained importance in several real-world domains such as process optimization, system identification, data mining, or statistical quality control. Tools are missing, which determine the applicability of computational intelligence methods in these application domains in an objective manner. Statistics provide methods for comparing algorithms on certain data sets. In the past, several test suites were presented and considered as state of the art. However, there are several drawbacks of these test suites, namely: (i) problem instances are somehow artificial and have no direct link to real-world settings; (ii) since there is a fixed number of test instances, algorithms can be fitted or tuned to this specific and very limited set of test functions; (iii) statistical tools for comparisons of several algorithms on several test problem instances are relatively complex and not easily to analyze. We propose a methodology to overcome these difficulties. It is based on standard ideas from statistics: analysis of variance and its extension to mixed models. This paper combines essential ideas from two approaches: problem generation and statistical analysis of computer experiments.

1 Introduction

Computational intelligence (CI) methods have gained importance in several real-world domains such as process optimization, system identification, data mining, or statistical quality control. Tools are missing, which determine the applicability of CI methods in these application domains in an objective manner. Statistics provide methods for comparing algorithms on certain data sets. In the past, several test suites were presented and considered as state of the art. However, there are several drawbacks of these test suites, namely:

- problem instances are somehow artificial and have no direct link to real-world settings;
- since there is a fixed number of test instances, algorithms can be fitted or tuned to this specific and very limited set of test functions. As a consequence, studies (benchmarks) provide insight how these algorithms perform on this specific set of test instances, but no insight on how they perform in general;

– statistical tools for comparisons of several algorithms on several test problem instances are relatively complex and not easily to analyze.

We propose a methodology to overcome these difficulties. It is based on ideas presented in Marco Chiarandini's and Yuri Goegebeur's seminal publication [10]. This methodology, which generates problem classes rather than use one instance, is constructed as follows. First, we pre-process the underlying real-world data. In a second step, features from these data are extracted. This extraction relies on the assumption that mathematical variables can be used to represent real-world features. Since we are using time-series data, standard tools from time-series analysis are applicable. For example, decomposition techniques can be applied to model the underlying data structures. We obtain an analytic model of the data. Then, we parametrize this model. Based on this parametrization and randomization, we can generate infinitely many new problem instances. From this infinite set, we can draw a limited number of problem instances which will be used for the comparison. Since problem instances are selected randomly, we apply random and mixed models for the analysis [14]. Mixed models include fixed and random effects. A fixed effect is an unknown constant. Its estimation from the data is a common practice in analysis of variance (ANOVA) or regression. A random effect is a random variable. We are estimating the parameters that describe its distribution, because—in contrast to fixed effects—it makes no sense to estimate the random effect itself.

We will present data used in case studies from drinking water management, energy production, and finance. These examples cover several application domains and illustrates that our approach is not limited to one specific problem instance only. Further problem domains can be added in an generic manner. This article combines ideas from two approaches: problem generation and statistical analysis of computer experiments. The generation of test problems, which are well-founded and have practical relevance, is an on-going field of research for several decades. [13] present a problem instance (landscape) generator that is parameterized by a small number of parameters, and the values of these parameters have a direct and intuitive interpretation in terms of the geometric features of the landscapes that they produce. The work presented by Chiarandini and Goegebeur [10] provides the basis of our statistical analysis. They present a systematic and well-developed framework for mixed models. We will combine this framework with ideas presented in [5]. Basically, this articler tries to find answers for the following fundamental questions in experimental research.

(Q-1) How to generate problem instances?
(Q-2) How to generalize experimental results?

The article is structured as follows. Section 2 introduces real-world problems and describes a taxonomy of their typical features. Algorithms and typical features are described in Sect. 3. Objective functions and statistical models are introduced in Sect. 4. These models take problem and algorithm features into consideration. Section 5 presents case studies, which illustrate our methodology. This article closes with a summary and an outlook.

2 Features of Real-World Problems

2.1 Problem Classes and Instances

Nowadays, it is a common practice in optimization to choose a *fixed* set of problem instances in advance and to apply classical ANOVA or regression analysis. In many experimental studies a few problem instances π_i $(i = 1, 2, \ldots, q)$ are used and results of some runs of the algorithms α_j $(j = 1, 2, \ldots, h)$ on these instances are collected. The instances can be treated as *blocks* and all algorithms are run on each single instance. Results are grouped per instance π_i. Analyses of these experiments shed some light on the performance of the algorithms on those specific instances. However, the interest of the researcher should not be just the performance of the algorithms on those specific instances chosen, but rather on the generalization of the results to the entire class Π. Generalizations about the algorithm's performance on new problem instances are difficult or impossible in this setting.

Based on ideas from Chiarandini and Goegebeur [10], to overcome this difficulty, we propose the following approach: A small set of problem instances $\{\pi_i \in \Pi | i = 1, 2, \ldots, q\}$ is chosen at random from a large set, or class Π, of possible instances of the problem. Problem instances are considered as factor levels. However, this factor is of a different nature from the fixed algorithmic factors in the classical ANOVA setting. Indeed, the levels are chosen at random and the interest is not in these specific levels but in the problem class Π from which they are sampled. Therefore, the levels and the factor are *random*. Consequently, our results are not based on a limited, fixed number of problem instances. They are randomly drawn from an infinite set, which enables generalization.

2.2 Feature Extraction and Instance Generation

A problem class Π can be generated in different manners. We will consider *artificial* and *natural* problem class generators. Artificially generated problems allow feature generation based on some predefined characteristics. They are basically theory driven, i.e., the researcher defines certain features such as linearity or multi modality. Based on these features, a model (formula) is constructed. By integrating parameters into this formula, many problem instances can be generated by parameter variation. We will exemplify this approach in the following paragraph. The second way, which will generate natural problem classes, uses a two-stage approach. First, features are extracted from the real-world system. Based on this feature set, a model is defined. Adding parameters to this model, new problem instances can be generated. There is also a third way to "generate" test instances: if we are lucky, many data are available. In this case, we can sample a limited number of problem instances from the larger set of real-world data. The statistical analysis is similar for these three cases.

Artificial Test Functions Several problem instance generators have been proposed over the last years. For example, [13] present a landscape test generator,

which can be used to set up problem instances for continuous, bound-constrained optimization problems.

To keep this article focused, we will propose a simple test problem instance generator, which is based on time-series decomposition. Inspired by the harmonic seasonal time series model with s seasons, which can be formulated as

$$Y(t) = m(t) + \sum_{k=1}^{[s/2]} \{s_k \sin(2\pi kt/s) + c_k \cos(2\pi kt/s)\} + Z(t), \tag{1}$$

where $m(t)$ denotes the trend and $Z(t)$ the error, we will define the following function generator $Y(\cdot)$

$$Y(x) = |b_0 + b_1 x + b_2 x^2 + \sin(b_3 \pi x/12) + \cos(b_4 \pi x/12) + \epsilon|, \tag{2}$$

where the b_i's are independent with $b_i \sim \mathcal{U}[0, w_i]$ and $\epsilon \sim \mathcal{N}(0, 1)$ for $i = 0, 1, \ldots, 4$.

The vector $\boldsymbol{w} = (w_0, w_1, w_2, w_3, w_4)'$ is used to define problem classes Π. Problem instances π can be drawn from each instance class. Using different random seeds for a fixed $\boldsymbol{w}$ in (2) results in different problem instances. These instances will be treated as levels of factors in the statistical analysis. Obviously, $\min(y(x)) \geq 0$. Nine typical problem instances are illustrated in Fig. 1. We consider the problem class Π_1, which is based on $\boldsymbol{w} = (-0.1, 0.01, 0.001, 10.0, 10.0)'$. We will use this problem instance generator in Sect. 5 to demonstrate our approach.

Natural Problem Classes This section exemplifies the three fundamental steps for generating real-world problem (RWP) instances, namely

1. Describing the real-world system and its data
2. Feature extraction and model construction
3. Instance generation

We will illustrate this procedure by using the classic Box and Jenkins airline data [9]. These data contain the monthly totals of international airline passengers, 1949 to 1960.

```
> str(AirPassengers)

 Time-Series [1:144] from 1949 to 1961: 112 118 132 129 121 135 148 148 136 119 ...
```

The feature extraction is based on methods from time-series analysis. Because of its simplicity the Holt-Winters method is popular in many application domains. It is able to adapt to changes in trends and seasonal patterns. The multiplicative Holt-Winters prediction function (for time series with period length p) is

$$\hat{Y}_{t+h} = (a_t + hb_t) s_{t-p+1+(h-1) \mod p},$$

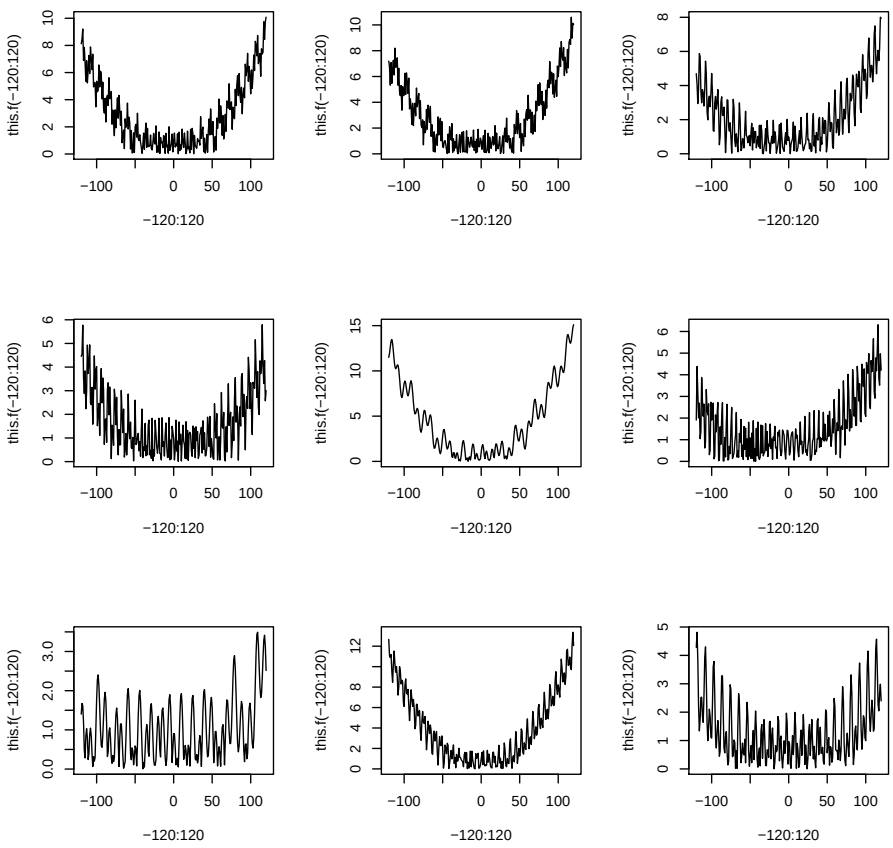


Fig. 1. Nine instances from problem class Π_1. Artificial problem instances are based on a harmonic time series model. The vector $\boldsymbol{w} = (-0.1, 0.01, 0.001, 10.0, 10.0)'$ was used for scaling the parameters in (2). The initial ES population is generated in the interval $[100; 120]$.

where a_t, b_t and s_t are given by

$$a_t = \alpha(Y_t/s_{t-p}) + (1-\alpha)(a_{t-1} + b_{t-1})$$
$$b_t = \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1}$$
$$s_t = \gamma(Y_t/a_t) + (1-\gamma)s_{t-p}$$

The optimal values of α, β and γ are determined by minimizing the squared one-step prediction error. New problem instances can be generated as follows. The parameters α, β, and γ are estimated from original time-series data Y_t. To generate new problem instances, these parameters can be slightly modified. Based on these modified values, the model is re-fitted. Finally, we can extract the new time series. Here, we plot the original data, the Holt-Winters predictions and the modified time series.

```
> generateHW <- function(a,b,c){
+ ## Estimation
+   m <- HoltWinters(AirPassengers, seasonal = "mult")
+ ## Extraction
+   alpha0<-m$alpha
+   beta0<-m$beta
+   gamma0<-m$gamma
+ ## Modification
+   alpha1 <- alpha0*a
+   beta1 <- beta0*b
+   gamma1 <- gamma0*c
+ ## Re-estimation
+   m1 <- HoltWinters(AirPassengers, alpha=alpha1
+   , beta = beta1, gamma = gamma1)
+ ## Instance generation
+   plot(AirPassengers)
+   lines(fitted(m)[,1], col = 1, lty=2, lw=2)
+   lines(fitted(m1)[,1], col = 1, lty = 3, lw =2)
+ }
> generateHW(a=.05,b=.025,c=.5)
```

One typical result from this instance generation is shown in Fig. 2.

To illustrate the wide applicability of this approach, we will list further real-work problem domains, which are subject of our current research.

Smart Metering. The development of accurate forecasting methods for electrical energy consumption profiles is an important task. Accurate consumption profile forecasting enables intelligent control of renewable energy source infrastructure, such as storage power plants, and therefore contributes to a smaller carbon footprint. Accurate consumption profile forecasting also enables energy consumers to accurately assess the return on investment of measures to increase energy efficiency. We consider time series collected from a manufacturing process. Each time series contains quarter-hourly samples

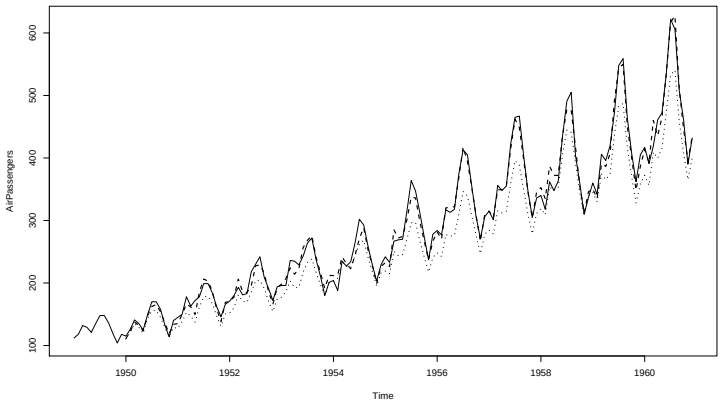


Fig. 2. Holt-Winters problem instance generator. The *solid* line represents the real data, the *dotted* line predictions from the Holt-Winters model and the *fine dotted* line modified predictions, respectively.

of the energy consumption of a bakery. A detailed data description can be found in [2].

Water Industry. CANARY is a software developed by the *United States Environmental Protection Agency* (US EPA) and Sandia National Laboratories. Its purpose is to detect events in the context of water contamination. An event is in this context defined as a certain time period where a contaminant deteriorates the water quality significantly. Distinguishing events from (i) background changes, (ii) maintenance and modification due to operation, and (iii) outliers is an essential task, which was implemented in the CANARY software. Therefore, deviations are compared to regular patterns and short term changes. The corresponding data contains multi-variate time-series data. It is a selection of a larger dataset shipped with the open source Event Detection Software CANARY developed by US EPA and Sandia National Laboratories [16].

Finance. The data are real-world data from intraday *foreign exchange* (FX) trading. The FX market is a financial market for trading currencies to enable international trade and investment. It is the largest and most liquid financial market in the world. Currencies can be traded via a wide variety of different financial instruments, ranging from simple spot trades over to highly complex derivatives. We are using three foreign exchange (currency rate) time series collected from Bloomberg. Each time series contains hourly samples of the change in currency exchange rate [11].

Now that we have demonstrated the applicability of our approach to a well known time series and listed time series, which are subject of our current research, we will introduce the optimization algorithm.

3 Algorithm Features

3.1 Factors and Levels

Evolutionary algorithms (EA) belong to the large class of bio-inspired search heuristics. They combine specific components, which may be *qualitative*, like the recombination operator or *quantitative*, like the population size. Our interest is in understanding the contribution of these components. In statistical terms, these components are called *factors*. The interest is in the effects of the specific *levels* chosen for these factors. Hence, we say that the levels and consequently the factors are *fixed*. Although modern search techniques like sequential parameter optimization or Pareto genetic programming allow multi-objective performance measures (solution quality versus variability or description length), we restrict ourselves to analyze the effect of these factors on a univariate measure of performance. We will use the quality of the solutions returned by the algorithm at termination as the performance measure.

3.2 Example: Evolution Strategy

Evolution strategies (ES) are prominent representatives of evolutionary algorithms, which includes genetic algorithms and genetic programming as well [15]. Evolution strategies are applied to hard real-valued optimization problems. Mutation is performed by adding a normally distributed random value to each vector component. The standard deviation of these random values is modified by self-adaptation. Evolution strategies can use a population of several solutions. Each solution is considered as as individual and consists of object and strategy variables. Object variables represent the position in the search space, whereas strategy variables store the step sizes, i.e., the standard deviations for the mutation. We are analyzing the ES basic variant, which has been proposed in [8]. It is understood as population based stochastic direct search algorithm—not excluding population sizes of one as e.g. featured in simple evolution strategies—that in some sense mimics the natural evolution.

Besides initialization and termination as necessary constituents of every algorithm, ES consist of three important factors: A number of search operators, an imposed control flow (Figure 3), and a representation that maps adequate variables to implementable solution candidates.

Although different ES may put different emphasis on the search operators mutation and recombination, their general effects are not in question. Mutation means neighborhood based movement in search space that includes the exploration of the "outer space" currently not covered by a population, whereas recombination rearranges existing information and so focuses on the "inner space". Selection is meant to introduce a bias towards better fitness values; GAs do so by regulating the crossover via mating selection, ESs utilize the environmental selection.

A concrete ES may contain specific mutation, recombination, or selection operators, or call them only with a certain probability, but the control flow is

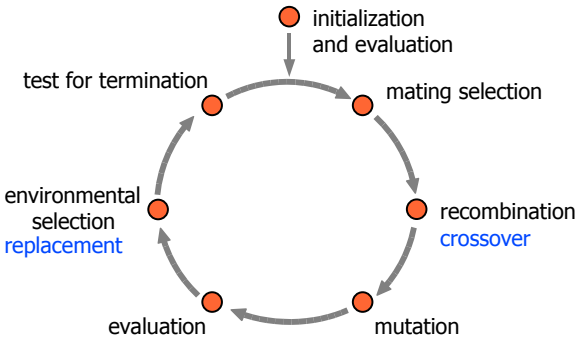


Fig. 3. The evolutionary cycle, basic working scheme of all ES and EA. Terms common for describing evolution strategies are used, alternative terms are added below in blue.

usually left unchanged. Each of the consecutive cycles is termed a *generation*. Concerning the representation, it should be noted that most empiric studies are based on canonical forms as binary strings or real-valued vectors, whereas many real-world applications require specialized, problem dependent ones. Table 1 summarizes important ES parameters. The reader is referred to [1] for a detailed description of these parameters.

Factors, which were modified during our case studies are listed in Table 1. We will consider effects of the mutation operator and the recombination operator for strategy and object variables on the performance.

4 Objective Functions and Statistical Models

4.1 Objective Function

We are convinced that the applicability of the methods presented in this article go far beyond the simplified case studies. Our main contribution is a framework, which allows conclusions that are no limited to a small number of problem instances but to problem classes. The focus our presentation lies on the real-world function generator and the related statistical framework. Under this perspective, it is legitimate to use the following simplified optimization framework: An ES is applied directly as a minimizer on the test function $f(x)$. Formally speaking, let S denote some set, e.g., $S \subseteq \mathbb{R}^n$. We are seeking for values f^* and x^*, such that $\min_{x \in S} f(x)$ with $f^* = \min_{x \in S} f(x)$ and $x^* = \arg\min f(x) : f(x^*) \leq f(x) \quad \forall x \in S$. This approach can be extended in many ways, e.g., by applying an optimization algorithm to minimize the empirical mean squared prediction error $\frac{1}{|S|} \sum_{x \in S} (\hat{Y}(x) - Y(x))^2$.

Table 1. Settings of exogenous parameters of an ES. Recombination operators are labeled as follows: 1=no, 2=dominant, 3=intermediate, 4=intermediate as in [8]. Mutation uses the following encoding: 1 = no mutation, 2 = self adaptive mutation.

Parameter	Symbol	Name	Range	Value
mue	μ	Number of parent individuals	$\mathbb{N}$	5
nu	$\nu = \lambda/\mu$	Offspring-parent ratio	$\mathtt{R}_+$	2
sigmaInit	$\sigma_i^{(0)}$	Initial standard deviations	$\mathtt{R}_+$	1
nSigma	n_σ	Number of standard deviations. d denotes the problem dimension	$\{1, d\}$	1
	c_τ	Multiplier for individual and global mutation parameters	$\mathtt{R}_+$	1
tau0			$\mathtt{R}_+$	0
tau			$\mathtt{R}_+$	1
rho	ρ	Mixing number	$\{1, \mu\}$	2
sel	κ	Maximum age	$\mathtt{R}_+$	1
sreco	r_σ	Recombination operator for strategy variables	$\{1, 2, 3, 4\}$	3
oreco	r_x	Recombination operator for object variables	$\{1, 2, 3, 4\}$	2
mutation		Mutation	$\{1, 2\}$	2

4.2 Mixed Models

Standard models in statistics are based on linear combinations of the factor effects. The standard analysis assumes that the factors remain fixed, i.e., the levels of the factors were the levels of interest. Conclusions from the statistical analysis are valid for these levels. In our setting, we are interested in drawing conclusions about a larger population of levels, not only those that were used in the experimental design. Therefore, factor levels are chosen at random from a larger population or class. The simplest situation occurs when an in a single-factor experiment one factor is random. This leads to the *random factor model*, which will be introduced in Sec. 5.2. Combining the fixed and random effects factors results in so-called *linear mixed models*.

5 Case Studies

As stated in Sec. 4, we are discussing simple optimization problems. The underlying algorithm and problem designs were chosen for didactical purpose only. However, this approach can be extended to time-series problems, e.g., in order to minimize prediction accuracy, mean-squared errors, or mean absolute errors.

5.1 SASP: Single Algorithm, Single Problem

In the most basic design, the researcher wishes to assess the performance of an *optimization algorithm* on a single problem instance π. An optimization problem has a set of input data which instantiate the problem. This might be a function in continuous optimization or the location and distances between cities

in a traveling salesman problem. Because many optimization algorithms such as evolutionary algorithms are randomized, their performance Y on one instance is a random variable. It might be described by a probability density/mass function $p(y|\pi)$. In experiments we collect sample data $y_1, \ldots, y_r$, which are independent, identically distributed. There are situations, in which SASP is the method of first choice. Real-world problems, which have to be solved only once in a very limited time, are good examples for using SASP optimizations. Because of its limited capacity for generalization, SASP will not be further investigated in this study.

5.2 SAMP: Single Algorithm, Multiple Problems

Fixed-effects Models This setup is commonly used for testing an algorithm on a given (fixed) set of problem instances. It is subject to many criticism, e.g., that algorithms are trained for this specific set up test instances (over fitting).

Standard assumptions from *analysis of variance* (ANOVA) lead us to propose the following *fixed-effects model* [14]:

$$Y_{ij} = \mu + \tau_i + \varepsilon_{ij}, \tag{3}$$

where μ is an overall mean, τ_i is a parameter unique to the ith treatment (problem instance factor), and ε_{ij} is a random error term for replication j on problem instance i. Usually, the model errors ε_{ij} are assumed to be normally and independently distributed with mean zero and variance σ^2. Since problem instance factors are fixed, i.e., non random, the stochastic behavior of the response variable originates from the algorithm. This implies the experimental results

$$Y_{ij} \sim N(\mu + \tau_i, \sigma^2), \quad i = 1, \ldots, q, \; j = 1, \ldots, r, \tag{4}$$

and that the Y_{ij} are mutually independent. Results from statistical analyses remains valid only on the specific instances. In order to take into account dependencies arising from applying an algorithm repeatedly to the same instances, [10] propose randomized and mixed models as an extension of (3). In contrast to model (3), these models allow generalizations of results to the whole class of possible instances.

Randomized Models In the following, we consider a population or *class* of instances Π. The class Π consists of a large, possibly an infinite, number of problem instances $\pi_i, i = 1, 2, 3, \ldots$ The performance Y of the algorithm α on the class Π is described by the probability function

$$p(y) = \sum_{\pi \in \Pi} p(y|\pi) p(\pi), \tag{5}$$

with $p(\pi)$ being the probability of sampling instance π. If we ran an algorithm α r times on instance π, then we receive r replicates of α's performance, denoted

by $Y_1, \ldots, Y_r$. These r observations are independent and identically distributed (i.i.d.), i.e.,

$$p(y_1, \ldots, y_r|\pi) = \prod_{j=1}^{r} p(y_j|\pi). \tag{6}$$

So far, we have considered r replicates of the performance measure Y on *one* problem instance π. Now consider *several*, randomly sampled problem instances. Over all the instances the observed performance measures may show dependence:

$$p(y_1, \ldots, y_r) = \sum_{\pi \in \Pi} p(y_1, \ldots, y_r|\pi)p(\pi). \tag{7}$$

Equation (6) will be referred to as the *conditional model*, whereas (7) will be referred to as the *marginal model*.

Example SAMP: ES on Π_1 (Random-Effects Design) The simplest random-effects experiment is performed as follows. For $i = 1, \ldots, q$ a problem instance π_i is drawn randomly from the class of problem instances Π. On each of the sampled π_i, the algorithm α is run r times using different seeds for α. Due to α's stochastic nature, we obtain, *conditionally on the sampled instance*, r replications of the performance measure that are i.i.d.

Let Y_{ij} $(i = 1, \ldots, q;\ j = 1, \ldots, r)$ denote the random performance measure obtained in the jth replication of α on π_i. We are interested in drawing conclusions about α's performance on a larger set of problem instances from Π, and not just on those q problem instances included in the experiment. A systematic approach to accomplish this task comprehends the following steps.

SAMP-1 Algorithm and Problem Instances
SAMP-2 Validation of the Model Assumptions
SAMP-3 Building the Model and ANOVA
SAMP-4 Hypothesis Testing
SAMP-5 Confidence Intervals and Prediction

SAMP-1 Algorithm and Problem Instances The goal of this case study is to analyze if one algorithm shows a similar performance on a class of problem instances, say Π_1. A random-effects design will be used to model the results. We illustrate the decomposition of the variance of the response values in (i) the variance due to problem instance and (ii) the variance due to the algorithm and derive results, which are based on hypotheses testing as introduced in (12).

We consider one algorithm, an ES, which is run $r = 5$ times on a set of randomly generated problem instances. The ES is parametrized with setting from Table 1. These parameters are kept constant during the experiment. Nine instances were drawn for the set of problem instances, which is generated with the time-series decomposition function generator from (2). The corresponding problem instances are shown in Fig. 1. The null hypothesis reads "There is no instance effect". Since we are considering the SAMP case, our experiments is based on one ES instance only, i.e., we are selecting self-adaptive mutation. We load the data frame and display its content.

```
'data.frame':        45 obs. of  4 variables:
 $ y      : num  0.2036 0.0557 0.0979 0.7142 4.3018 ...
 $ fSeed  : Factor w/ 9 levels "1","2","3","4",..: 1 1 1 1 1 2 2 2 2 2 ...
 $ algSeed: Factor w/ 5 levels "1","2","3","4",..: 1 2 3 4 5 1 2 3 4 5 ...
 $ yLog   : num  -1.592 -2.887 -2.324 -0.337 1.459 ...
```

Here, y denotes the function value. Since self adaptation was used, `mut` is set to "2". The variable `fSeed` represents the problem instance, `algSeed` is the repeat of the ES run with different seeds, while `yLog` denotes the log function values. There are 45 observations of 5 variables, because 5 repeats were performed on 9 problem instances.

SAMP-2 Validation of the Model Assumptions In the second step we should test the validity of the model assumptions by generating normal quantile plots (QQ plots) as shown in Fig. 4.

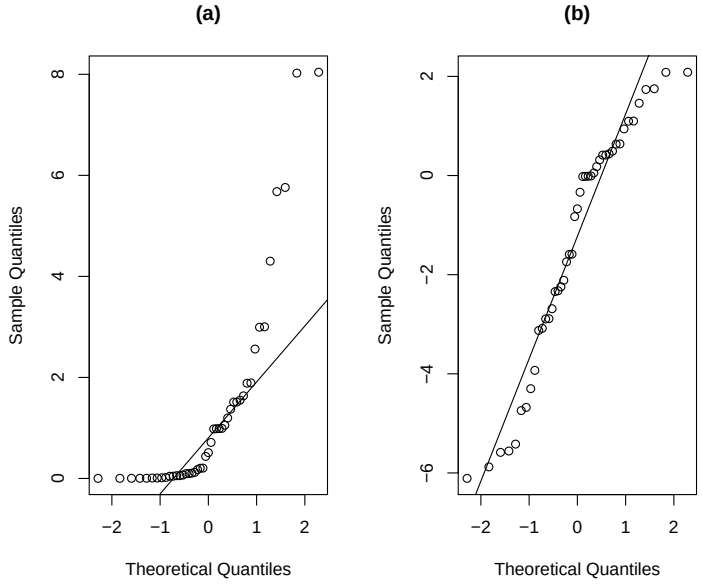


Fig. 4. Quantile-Quantile (Q-Q) plots provide a good way to compare the the distribution of a sample with a distribution. Here, the samples are compared to a normal distribution. Large deviations from the line indicate non-normality of the sample data. These two Q-Q plots show that a log transformation might be useful: (a) before the log transformation, (b) after the log transformation is applied to the data.

Table 2. ANOVA table for a one-factor fixed and random effects models

Source of Variation	Sum of Squares	Degrees of freedom	Mean Square	EMS Fixed	EMS Random
Treatment	$\mathrm{SS}_{\mathrm{treat}}$	$q-1$	$\mathrm{MS}_{\mathrm{treat}}$	$\sigma^2 + r\frac{\sum_{i=1}^{q}\tau_i^2}{q-1}$	$\sigma^2 + r\sigma_\tau^2$
Error	$\mathrm{SS}_{\mathrm{err}}$	$q(r-1)$	$\mathrm{MS}_{\mathrm{err}}$	σ^2	σ^2
Total	$\mathrm{SS}_{\mathrm{total}}$	$qr-1$			

SAMP-3 Building the Model and ANOVA The following analysis is based on the linear statistical model

$$Y_{ij} = \mu + \tau_i + \varepsilon_{ij} \begin{cases} i = 1, \ldots, q \\ j = 1, \ldots, r \end{cases} \tag{8}$$

where μ is an overall mean and ε_{ij} is a random error term for replication j on instance i. Note, in contrast to the fixed-effects model from (3), τ_i is a random variable representing the effect of instance i. The stochastic behavior of the response variable originates from both the instance and the algorithm. This is reflected in (8), where both τ_i and ϵ_{ij} are random variables. The model (8) is the so-called *random-effects model*, cf. [14, p. 512] or [10, p. 229].

If τ_i is independent of ϵ_{ij} and has variance $\mathrm{Var}(\tau_i) = \sigma_\tau^2$, the variance of any observation is $\mathrm{Var}(y_{ij}) = \sigma^2 + \sigma_\tau^2$. Furthermore, we assume that $\tau_1, \ldots, \tau_q$ are i.i.d. $\mathcal{N}(0, \sigma_\tau^2)$ and ε_{ij}, $i = 1, \ldots, q$, $j = 1, \ldots, r$, are i.i.d. $\mathcal{N}(0, \sigma^2)$. Similar to the partition in classical ANOVA, the variability in the observations can be partitioned into a component that measures the variation between treatments and a component that measures the variation within treatments. Based on the fundamental ANOVA identity $\mathrm{SS}_{\mathrm{total}} = \mathrm{SS}_{\mathrm{treat}} + \mathrm{SS}_{\mathrm{err}}$, we define

$$\mathrm{MS}_{\mathrm{treat}} = \frac{\mathrm{SS}_{\mathrm{treat}}}{q-1} = \frac{r\sum_{i=1}^{q}(\bar{Y}_{i.} - \bar{Y}_{..})^2}{q-1},$$

and

$$\mathrm{MS}_{\mathrm{err}} = \frac{\mathrm{SS}_{\mathrm{err}}}{q(r-1)} = \frac{\sum_{i=1}^{q}\sum_{j=1}^{r}(Y_{ij} - \bar{Y}_{i.})^2}{q(r-1)}.$$

It can be shown that

$$E(\mathrm{MS}_{\mathrm{treat}}) = \sigma^2 + r\sigma_\tau^2 \quad \text{and} \quad E(\mathrm{MS}_{\mathrm{err}}) = \sigma^2, \tag{9}$$

cf. [14]. Therefore, the estimators of the variance components are

$$\hat{\sigma}^2 = \mathrm{MS}_{\mathrm{err}} \tag{10}$$

$$\hat{\sigma_\tau^2} = \frac{\mathrm{MS}_{\mathrm{treat}} - \mathrm{MS}_{\mathrm{err}}}{r}. \tag{11}$$

The corresponding ANOVA table is shown in Table 2.

We will demonstrate, how these estimators can be calculated in R. First, the ANOVA model is build. Then, we extract the mean squared values, i.e., MSA (treatment) and MSE (error). The estimators of the variance components can be calculated as follows. From (10) we obtain an estimator of the first variance component $\hat{\sigma}^2$ as the mean squared error and from (11), we obtain the second component $\hat{\sigma}_\tau^2$. The model variance can be determined as `var.A +var.B`. Finally, the mean μ from (8) can extracted.

```
> samp.aov <- aov(yLog ~fSeed, data=samp.df)
> (M1 <- anova(samp.aov))

Analysis of Variance Table

Response: yLog
          Df  Sum Sq Mean Sq F value Pr(>F)
fSeed      8  48.832  6.1040  1.0707 0.4048
Residuals 36 205.230  5.7008

> (MSA <- M1[1,3])

[1] 6.10401

> (MSE <- M1[2,3])

[1] 5.700838

> r <-length(unique(samp.df$algSeed))
> q <- nlevels(samp.df$fSeed)
> (var.A <- (MSA - MSE)/(r))

[1] 0.0806345

> (var.E <- MSE)

[1] 5.700838

> var.A + var.E

[1] 5.781472

> coef(samp.aov)[1]

(Intercept)
  -1.136131
```

The p value in the ANOVA table is calculated as

```
> 1-pf(MSA/MSE,q-1,q*(r-1))

[1] 0.4047883
```

The MSA value will be stored for the calculation of confidence intervals.

```
> MSA.anova <- MSA
```

In some cases, the standard ANOVA, which was used in our example, produces a negative estimate of a variance component. This can be seen in (11): If $\mathrm{MS}_{\mathrm{err}} > \mathrm{MS}_{\mathrm{treat}}$, negative values occur. By definition, variance components are positive. Methods, which always yield positive variance components have been developed. Here, we will use restricted maximum likelihood estimators (REML). The ANOVA method of variance component estimation, which is a method of moments procedure, and REML estimation may lead to different results.

Restricted maximum likelihood. Based on the same data, we fit the random-effects model (8) using the function `lmer()` from the `R` package `lme4` [7]:

```
> library(lme4)
> samp.lmer <- lmer(yLog~ 1 +(1|fSeed),data=samp.df)
> print(samp.lmer, digits = 4, corr = FALSE)

Linear mixed model fit by REML
Formula: yLog ~ 1 + (1 | fSeed)
   Data: samp.df
   AIC   BIC logLik deviance REMLdev
 211.8 217.2 -102.9    205.6   205.8
Random effects:
 Groups   Name        Variance   Std.Dev.
 fSeed    (Intercept) 2.6192e-11 5.1179e-06
 Residual             5.7741e+00 2.4029e+00
Number of obs: 45, groups: fSeed, 9

Fixed effects:
            Estimate Std. Error t value
(Intercept)  -1.3528     0.3582  -3.776
```

First, the model formula `( yLog` $\sim$ `1 + (1| fSeed) )` is shown. The data is grouped by `fSeed`, because problem instances π_i are generated using (2) with nine different seeds. The fixed effect is the intercept, which is represented by the symbol `1` in the formula. The term `(1| fSeed)` indicates that the data is grouped by `fSeed`. The `1` is indicating that the random effect is constant within each group. Information about measures of the fitting (`AIC, BIC,` etc.) are displayed next. Our mains interest lies on the next lines of the output, which are labeled `Random effects`. Here we find the estimates of parameters related to the random effects and the error distributions, i.e., the variances for the problem instances, i.e., τ or `fSeed` and the algorithm, i.e., ϵ or `Residual`. This shows that the variability in the response observations can be attributed to the variability of the algorithm.

SAMP-4 Hypothesis Testing Testing hypotheses about individual treatments (instances) is useless, because the problem instances π_i are here considered as samples from some larger population of instances Π. We test hypotheses about the variance component σ_τ^2, i.e., the null hypothesis

$$H_0 : \sigma_\tau^2 = 0 \qquad \text{is tested versus the alternative} \qquad H_1 : \sigma_\tau^2 > 0. \tag{12}$$

Under H_0, all treatments are identical, i.e., $r\sigma_\tau^2$ is very small. Based on (9), we conclude that $E(\mathrm{MS}_{\mathrm{treat}}) = \sigma^2 + r\sigma_\tau^2$ and $E(\mathrm{MS}_{\mathrm{err}}) = \sigma^2$ are similar. Under the alternative, variability exists between treatments. Standard analysis shows that $\mathrm{SS}_{\mathrm{err}}/\sigma^2$ is distributed as chi-square with $q(r-1)$ degrees of freedom. Under H_0, the ratio

$$F_0 = \frac{\frac{\mathrm{SS}_{\mathrm{treat}}}{q-1}}{\frac{\mathrm{SS}_{\mathrm{err}}}{q(r-1)}} = \frac{\mathrm{MS}_{\mathrm{treat}}}{\mathrm{MS}_{\mathrm{err}}}$$

is distributed as $F_{q-1,q(r-1)}$. To test hypotheses in (8), we require that $\tau_1, \ldots, \tau_q$ are i.i.d. $\mathcal{N}(0, \sigma_\tau^2)$, ε_{ij}, $i = 1, \ldots, q$, $j = 1, \ldots, r$, are i.i.d. $\mathcal{N}(0, \sigma^2)$, and all τ_i and ε_{ij} are independent of each other.

These considerations lead to the decision rule to reject H_0 at the significance level α if

$$f_0 > F(1-\alpha; q-1, q(r-1)), \tag{13}$$

where f_0 is the realization of F_0 from the observed data. An intuitive motivation for the form of statistic F_0 can be obtained from the expected mean squares. Under H_0 both $\mathrm{MS}_{\mathrm{treat}}$ and $\mathrm{MS}_{\mathrm{err}}$ estimate σ^2 in an unbiased way, and F_0 can be expected to be close to one. On the other hand, large values of F_0 give evidence against H_0.

Based on (9), we can determine the F statistic and the p values:

```
> VC <- VarCorr(samp.lmer)
> (sigma.tau <- as.numeric(attr(VC$fSeed,"stddev")))

[1] 5.117856e-06

> (sigma <- as.numeric(attr(VC,"sc")))

[1] 2.402944

> q <- nlevels(samp.df$fSeed)
> r <- length(unique(samp.df$algSeed))
> (MSA <- sigma^2+r*sigma.tau^2)

[1] 5.774142

> (MSE <- sigma^2)

[1] 5.774142
```

Now we can determine the p value based on (13):

```
> 1-pf(MSA/MSE,q-1,q*(r-1))

[1] 0.4529257
```

Since the p value is large, the null hypothesis $H_0 : \sigma_\tau^2 = 0$ from (12) can not be rejected, i.e., we conclude that there is no instance effect. A similar conclusion was obtained from the ANOVA method of variance component estimation.

SAMP-5 Confidence Intervals and Prediction An unbiased estimator of the overall mean μ is $\sum_{i=1}^{q}\sum_{j=1}^{r} y_{ij}/(qr)$. It can be shown that its estimated standard error is given by $\mathrm{se}(\hat{\mu}) = \sqrt{\mathrm{MStreat}/qr}$ and that

$$\frac{\bar{Y}_{\cdot\cdot} - \mu}{\sqrt{\mathrm{MStreat}/qr}} \sim t(q-1).$$

Hence, [10, p. 232] show that confidence limits for μ can be derived as

$$\bar{y}_{\cdot\cdot} \pm t(1-\alpha/2; q-1)\sqrt{\mathrm{MStreat}/qr}. \tag{14}$$

We conclude this case study with prediction of the algorithm's performance on a new instance. Based on (14), the 95% confidence interval can be calculated as follows.

```
> s <- sqrt(MSA/(q*r))
> Y.. <- mean(samp.df$yLog)
> qsr <- qt(1-0.025,r)
> c( exp(Y.. - qsr * s), exp(Y.. + qsr * s))

[1] 0.1029441 0.6492394
```

Since we performed the analysis on log data, the exp() function was applied to the final result. Hence, 95% confidence interval for μ is $[0.10; 0.65]$.

Using the ANOVA results from above, we obtain the following confidence interval for the performance of the ES:

```
> s <- sqrt(MSA.anova/(q*r))
> Y.. <- mean(samp.df$yLog)
> qsr <- qt(1-0.025,5)
> c( exp(Y.. - qsr * s), exp(Y.. + qsr * s))

[1] 0.1003084 0.6662989
```

Second SAMP Example The ES parametrization remains unchanged, but the parametrization of the problem instances was modified. Nine problems instances π_i $(i = 1, 2, \ldots, 9)$ were generated, using the problem parameter vector $(-0.1, -0.1, -0.001, .10, 2)\times i^2$. The resulting realizations are illustrated in Fig. 5.

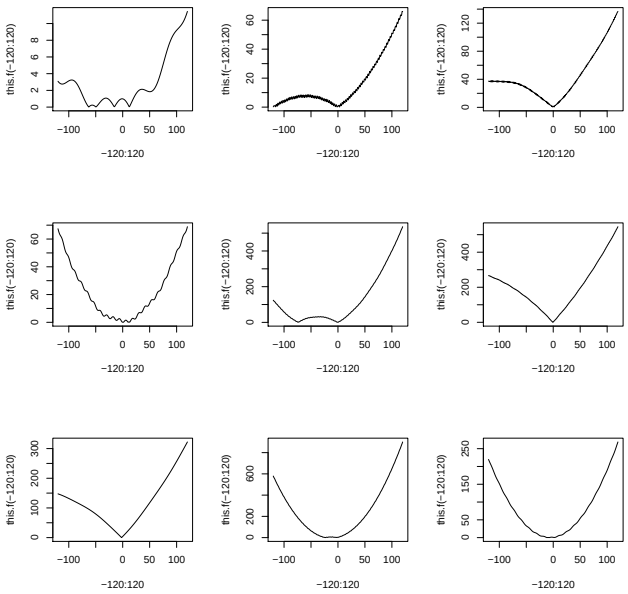


Fig. 5. Second set of problem instances Π_2. The ES shows different performances on this set of problem instances.

```
> str(samp2.df)

'data.frame':        45 obs. of  4 variables:
 $ y      : num  0.0315 0.1171 0.0136 1.8438 0.5961 ...
 $ fSeed  : Factor w/ 9 levels "1","2","3","4",..: 1 1 1 1 1 2 2 2 2 2 ...
 $ algSeed: Factor w/ 5 levels "1","2","3","4",..: 1 2 3 4 5 1 2 3 4 5 ...
 $ yLog   : num  -3.456 -2.145 -4.299 0.612 -0.517 ...
```

Again, we test the validity of the model assumptions by generating normal quantile plots (QQ plots) as shown in Fig. 6.

We consider the classical ANOVA first.

```
> samp2.aov <- aov(yLog ~fSeed, data=samp2.df)
> (M2 <- anova(samp2.aov))

Analysis of Variance Table

Response: yLog
          Df Sum Sq Mean Sq F value    Pr(>F)
fSeed      8 82.830 10.3538  5.9856 6.805e-05 ***
Residuals 36 62.272  1.7298
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

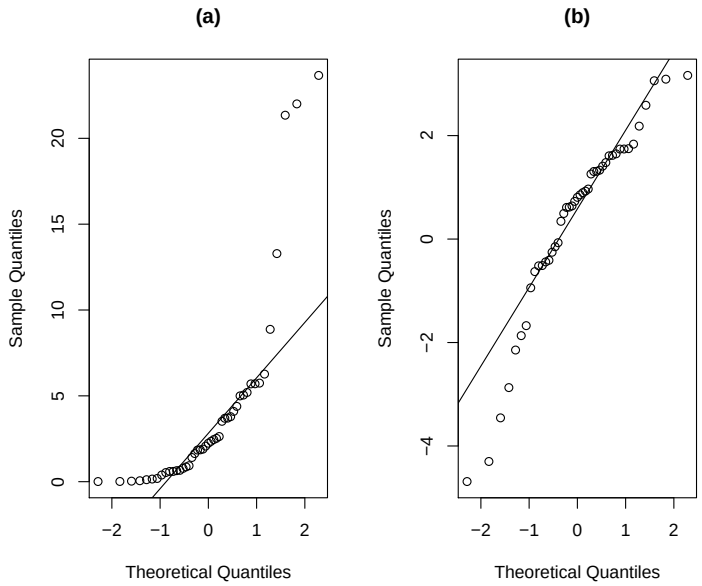


Fig. 6. Quantile-Quantile (Q-Q) plots for the second SAMP example: (a) before the log transformation, (b) after the log transformation is applied to the data.

```
> (MSA <- M2[1,3])

[1] 10.35378

> (MSE <- M2[2,3])

[1] 1.729791

> r <-length(unique(samp2.df$algSeed))
> q <- nlevels(samp2.df$fSeed)
```

Following (11), the variance components $\hat{\sigma_\tau^2}$ (`var.A`) and $\hat{\sigma}^2$ (`var.E`) can be determined as follows.

```
> (var.A <- (MSA - MSE)/(r))

[1] 1.724798

> (var.E <- MSE)

[1] 1.729791
```

That is, we have $\hat{\sigma}_\tau^2 = 0.08$ and $\sigma^2 = 5.7$. The p value is

```
> 1-pf(MSA/MSE,q-1,q*(r-1))

[1] 6.805386e-05
```

We obtain the following confidence interval.

```
> s <- sqrt(MSA/(q*r))
> Y.. <- mean(samp2.df$yLog)
> qsr <- qt(1-0.025,5)
> c( exp(Y.. - qsr * s), exp(Y.. + qsr * s))

[1] 0.4260439 5.0171031
```

REML Next, we consider the restricted maximum likelihood approach.

```
Linear mixed model fit by REML
Formula: yLog ~ 1 + (1 | fSeed)
   Data: samp2.df
   AIC   BIC logLik deviance REMLdev
 173.1 178.5 -83.55    167.5   167.1
Random effects:
 Groups   Name        Variance Std.Dev.
 fSeed    (Intercept) 1.7248   1.3133
 Residual             1.7298   1.3152
Number of obs: 45, groups: fSeed, 9

Fixed effects:
            Estimate Std. Error t value
(Intercept)   0.3798     0.4797   0.792
```

The statistical analysis reveals that the variability in the response observations can be attributed to the variability in the problem instances. We continue by computing the F statistic and the p value.

```
> VC <- VarCorr(samp2.lmer)
> (var.A  <- (as.numeric(attr(VC$fSeed,"stddev")))^2)

[1] 1.724797

> (var.E <- (as.numeric(attr(VC,"sc")))^2)

[1] 1.729791

> q <- nlevels(samp2.df$fSeed)
> r <- length(unique(samp2.df$algSeed))
> (MSA <- var.E+r*var.A)

[1] 10.35378
```

```
> (MSE <- var.E)

[1] 1.729791

> 1-pf(MSA/MSE,q-1,q*(r-1))

[1] 6.805392e-05
```

The resulting p value gives reason for rejecting the null hypotheses $H_0 : \sigma_\tau^2 = 0$ as shown in (12), i.e., we conclude that there might be instance effects. The corresponding 95% confidence interval for new problem instances is larger, which also indicates that there are performance differences. Based on (14), we obtain the following confidence interval for the performance of the ES:

```
[1] 0.4260439 5.0171029
```

Confidence intervals from the REML and ANOVA methods are very similar.

5.3 MAMP: Multiple Algorithms, Multiple Problems:

In this case study, we demonstrate how the marginal model (7) can be extended to the case where several algorithms are applied to the same instance. We add fixed effects in the conditional structure of (6). Next, we illustrate how this leads naturally to a mixed model.

Instead of one fixed algorithm, we consider several algorithms or algorithms with several parameters. Both situations can be treated while considering algorithms as levels of a fixed factor, whereas problem instances are drawn randomly from some population of instances Π.

MAMP-1 Algorithm and Problem Instances
MAMP-2 Validation of the Model Assumptions
MAMP-3 Building the Model and ANOVA
MAMP-4 Hypothesis Testing
  a) Random effects
  b) Fixed effects
  c) Back-fitting (for multiple fixed factors)
MAMP-5 Confidence Intervals and Prediction

MAMP-1 Algorithm and Problem Instances In the first design we aim at comparing the performance of the ES with different recombination operators over an instance class. More precisely, we have the following factors:

- `algorithm`: four ES instances using recombination operators $\{1, 2, 3, 4\}$
- `instances`: nine instances randomly sampled from the class Π_1 as illustrated in Fig. 1 with problem parameters `(-0.1, 0.01, 0.001, 10.0, 10.0)`
- `replicates`: five

```
> str(mamp.df)
```

```
'data.frame':        180 obs. of  5 variables:
 $ y      : num  0.001725 0.008679 0.001094 0.010323 0.000853 ...
 $ sreco  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 2 2 2 2 2 ...
 $ fSeed  : Factor w/ 9 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ algSeed: Factor w/ 5 levels "1","2","3","4",..: 1 2 3 4 5 1 2 3 4 5 ...
 $ yLog   : num  -6.36 -4.75 -6.82 -4.57 -7.07 ...
```

As can be seen from the `str` output, $4 \times 9 \times 5 = 180$ data were used in this study.

MAMP-2 Validation of the Model Assumptions Again, we test the validity of the model assumptions by generating normal quantile plots (QQ plots) as shown in Fig. 6.

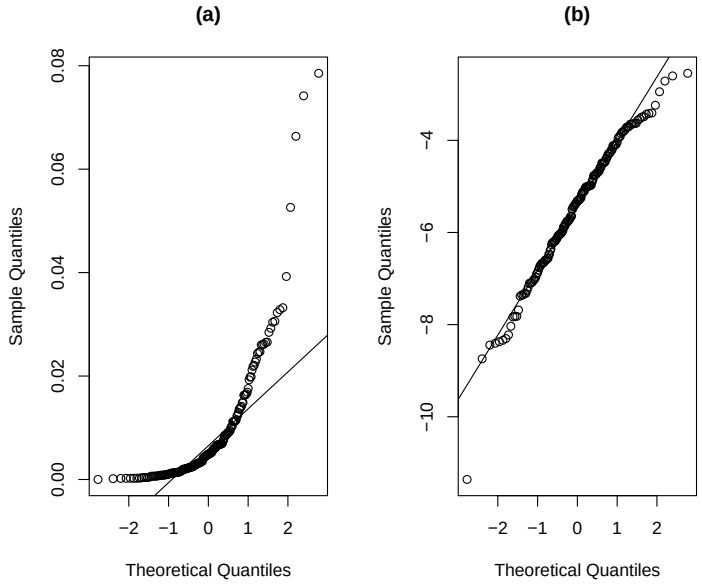


Fig. 7. Quantile-Quantile (Q-Q) plots for the MAMP example: (a) before the log transformation, (b) after the log transformation is applied to the data. Although there is still an outlier in the log transformed data, we will use the transformed data.

Next, we plot the results for each group. A first visual inspection, which plots the performance of the algorithm within each problem instance, is shown in Fig. 8.

```
> library(lattice)
> print(xyplot(yLog ~ sreco | fSeed, data=mamp.df,
```

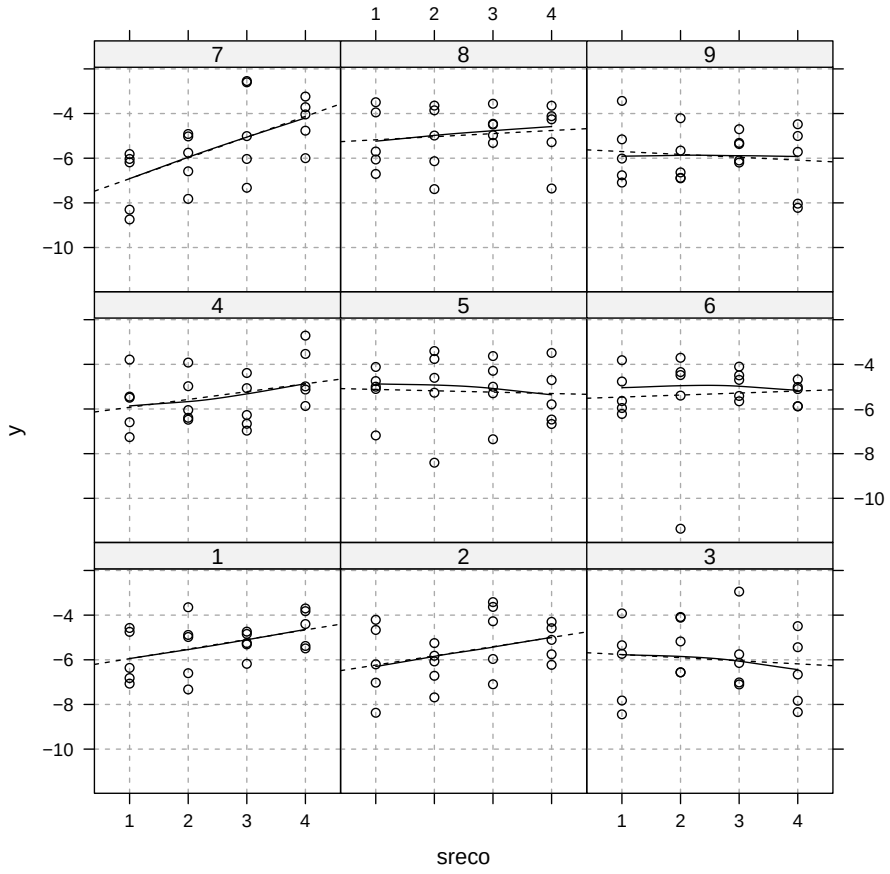


Fig. 8. Four algorithms (ES with modified recombination operators) on nine test problem instances. Each panel represents one problem instance. Performance is plotted against the level of the recombination operator.

```
+               main="",ylab="y",xlab="sreco",
+               panel=function(x, y){
+                 m <- sort.list(x)
+                 panel.grid(h=-1,v=-1,lty=2)
+                 panel.xyplot(x[m], y[m])
+                 panel.loess(x[m], y[m], span=2, lty=1)
+                 panel.lmline(x[m], y[m], lty=2)
+               }
+         )
+ )
```

MAMP-3 Building the Model and ANOVA Variance decompositions. The variability in the performance measure can be decomposed according to the following *mixed-effects ANOVA model*:

$$Y_{ijk} = \mu + \alpha_j + \tau_i + \gamma_{ij} + \varepsilon_{ijk}, \tag{15}$$

where μ is an overall performance level common to all observations, α_j is a fixed effect due to the algorithm j, τ_i is a random effect associated with instance i, γ_{ij} is a random interaction between instance i and algorithm j, and ε_{ijk} is a random error for replication k of algorithm j on instance i. For identification purposes we impose the usual sum constraint on the factor level effects, i.e., $\sum_{j=1}^{h} \alpha_j = 0$. This can be accomplished in R using the following command

```
> options(contrasts=c("contr.sum","contr.poly"))
```

The assumptions imposed on the random elements are τ_i are i.i.d. $\mathcal{N}(0, \sigma_\tau^2)$, γ_{ij} are i.i.d. $\mathcal{N}(0, \sigma_\gamma^2)$, ε_{ijk} are i.i.d. $N(0, \sigma^2)$, and τ_i, γ_{ij} and ε_{ijk} are mutually independent random variables. Similar to (6) the conditional distribution of the performance measure given the instance and the instance–algorithm interaction is given by

$$Y_{ijk} | \tau_i, \gamma_{ij} \sim N(\mu + \alpha_j + \tau_i + \gamma_{ij}, \sigma^2), \tag{16}$$

with $i = 1, \ldots, q$, $j = 1, \ldots, h$, $k = 1, \ldots, r$. The marginal model reads (after integrating out the random effects τ_i and γ_{ij}):

$$Y_{ijk} \sim \mathcal{N}(\mu + \alpha_j, \sigma^2 + \sigma_\tau^2 + \sigma_\gamma^2), \quad i = 1, \ldots, q,\ j = 1, \ldots, h,\ k = 1, \ldots, r. \tag{17}$$

Based on these statistical assumptions, hypotheses tests can be performed about fixed and random factor effects.

If all treatment (problem instances) combinations have the same number of observations, i.e., if the design is balanced, the test statistics for these hypotheses are ratios of mean squares that are chosen such that the expected mean squares of the numerator differs from the expected mean squares of the denominator only by the variance components of the random factor in which we are interested. Chiarandini and Goegebeur [10] present the resulting analysis of variance, which is shown in Table 3.

Table 3. Expected mean squares and consequent appropriate test statistics for a mixed two-factor model with h fixed factors, q random factors, and r repeats. From [10, p. 235].

Effects	Mean squares	df	Expected mean squares	Test statistics
Fixed factor	MSA	$h-1$	$\sigma^2 + r\sigma_\gamma^2 + rp\frac{\sum_{j=1}^{h} \alpha_j^2}{h-1}$	MSA/MSAB
Random factor	MSB	$q-1$	$\sigma^2 + r\sigma_\gamma^2 + rh\sigma_\tau^2$	MSB/MSAB
Interaction	MSAB	$(h-1)(q-1)$	$\sigma^2 + r\sigma_\gamma^2$	MSAB/MSE
Error	MSE	$hq(r-1)$	σ^2	

ANOVA

```
> mamp.aov <- aov(yLog ~ sreco*fSeed, data=mamp.df)
> h <- nlevels(mamp.df$sreco)
> q <- nlevels(mamp.df$fSeed)
> r <- nlevels(mamp.df$algSeed)
> (M1 <- anova(mamp.aov))

Analysis of Variance Table

Response: yLog
             Df Sum Sq Mean Sq F value Pr(>F)
sreco         3   13.5    4.51    2.28  0.082
fSeed         8   16.5    2.07    1.05  0.405
sreco:fSeed  24   37.8    1.57    0.80  0.738
Residuals   144  284.9    1.98

> (MSA <- M1[1,3])

[1] 4.51

> (MSB <- M1[2,3])

[1] 2.07

> (MSAB <- M1[3,3])

[1] 1.57

> (MSE <- M1[4,3])

[1] 1.98
```

Now we can extract the variance components.

```
> (var.A <- (MSA - MSAB)/(q*r))

[1] 0.0652

> (var.B <- (MSB - MSAB)/(h*r))

[1] 0.0247

> (var.AB <- (MSAB - MSE)/r)

[1] -0.0809

> (var.E <- MSE)

[1] 1.98

> (var.A + var.B + var.AB + var.E)

[1] 1.99
```

The estimate of the variance component σ^2_γ, i.e., var.AB, is negative. There are several ways to deal with this result [14]. Here, we are assuming that var.AB vanishes and generate the following additive model without interactions:

```
> mamp.aov2 <- aov(yLog ~ sreco + fSeed, data=mamp.df)
> summary(mamp.aov2)

             Df Sum Sq Mean Sq F value Pr(>F)
sreco         3     14    4.51    2.35  0.075
fSeed         8     17    2.07    1.08  0.382
Residuals   168    323    1.92
```

By inspection of the p values, we can even assume that one of the main effects, `fSeed`, vanishes.

```
> mamp.aov3 <- aov(yLog ~ sreco, data=mamp.df)
> (M3 <- anova(mamp.aov3))

Analysis of Variance Table

Response: yLog
           Df Sum Sq Mean Sq F value Pr(>F)
sreco       3     14    4.51    2.34  0.075
Residuals 176    339    1.93
```

We are back in the one-way ANOVA.

```
> (MSA <- M3[1,3])

[1] 4.51
```

```
> (MSE <- M3[2,3])

[1] 1.93

> (var.A <- (MSA - MSE)/(h*r))

[1] 0.129

> (var.E <- MSE)

[1] 1.93

> var.A + var.E

[1] 2.06

> (Yj. <- with(mamp.df,aggregate(yLog,list(alg=sreco),mean)))

  alg     x
1   1 -5.82
2   2 -5.65
3   3 -5.18
4   4 -5.23
```

A *Tukey's Honest Significant Difference* (Tukey HSD) test, which generates a set of confidence intervals on the differences between the means of the levels of a factor with the specified family-wise probability of coverage, is performed next.

```
> (mamp.aov3.Tukey=TukeyHSD(mamp.aov3,"sreco"))

  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = yLog ~ sreco, data = mamp.df)

$sreco
       diff    lwr   upr p adj
2-1  0.1659 -0.593 0.925 0.942
3-1  0.6435 -0.116 1.403 0.128
4-1  0.5906 -0.169 1.350 0.185
3-2  0.4776 -0.282 1.237 0.363
4-2  0.4247 -0.334 1.184 0.470
4-3 -0.0529 -0.812 0.706 0.998
```

The largest difference occurs between *no* (1) and *intermediate* (3) recombination of the strategy variables. However, this difference is not statistically significant. Figure 9 illustrates this result.

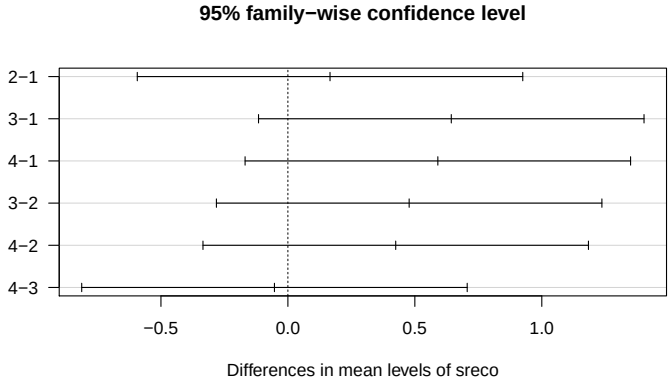


Fig. 9. Tukey HSD test comparison plots. Results from four ES instances with different recombination operators are shown in this plot.

REML We will consider the results of the analysis of the mixed model based on `lmer()`. As stated above, we have specified sum contrasts instead of the default treatment contrasts used in `lmer()`.

```
> op <- options(contrasts=c("contr.sum","contr.poly"))
> mamp.lmer <- lmer(yLog ~ sreco + (1|fSeed) + (1|fSeed:sreco), data=mamp.df)
> print(mamp.lmer, digits = 4)

Linear mixed model fit by REML
Formula: yLog ~ sreco + (1 | fSeed) + (1 | fSeed:sreco)
   Data: mamp.df
   AIC   BIC logLik deviance REMLdev
 646.9 669.3 -316.5    624.9   632.9
Random effects:
 Groups      Name        Variance Std.Dev.
 fSeed:sreco (Intercept) 3.46e-18 1.86e-09
 fSeed       (Intercept) 7.37e-03 8.58e-02
 Residual                1.92e+00 1.39e+00
Number of obs: 180, groups: fSeed:sreco, 36; fSeed, 9

Fixed effects:
            Estimate Std. Error t value
(Intercept)  -5.4694     0.1072  -51.03
sreco1       -0.3500     0.1789   -1.96
sreco2       -0.1841     0.1789   -1.03
sreco3        0.2935     0.1789    1.64

Correlation of Fixed Effects:
```

```
       (Intr) sreco1 sreco2
sreco1  0.000
sreco2  0.000 -0.333
sreco3  0.000 -0.333 -0.333
```

We will consider random effects first.

MAMP-4a) Hypothesis Testing: Random Effects The estimated variances for the problem instance (`fSeed`) is small ($7.37e-03$), whereas the variance due to the algorithm (`Residual`) is relatively large (1.92).

Regarding problem instances, test about levels are meaningless. Hence, we perform tests about the variance components σ_τ^2 and σ_γ^2, which can be formulated as follows:

$$\begin{array}{lll} H_0 : \sigma_\tau^2 = 0, & \text{and} & H_0 : \sigma_\gamma^2 = 0, \\ H_1 : \sigma_\tau^2 > 0, & & H_1 : \sigma_\gamma^2 > 0, \end{array}$$

respectively. In contrast to the ANOVA method described before, we do not look up the values in the ANOVA table (e.g., Table 3), but compute the likelihood ratio as follows.

```
> mamp.lmer2 <- lmer(yLog ~ sreco + (1|fSeed:sreco)
         , data=mamp.df, REML=FALSE)
> mamp.lmer3 <- lmer(yLog ~ sreco + (1|fSeed) + (1|fSeed:sreco)
         , data=mamp.df, REML=FALSE)
> anova(mamp.lmer3,mamp.lmer2)

Data: mamp.df
Models:
mamp.lmer2: yLog ~ sreco + (1 | fSeed:sreco)
mamp.lmer3: yLog ~ sreco + (1 | fSeed) + (1 | fSeed:sreco)
           Df AIC BIC logLik Chisq Chi Df Pr(>Chisq)
mamp.lmer2  6 637 656   -312
mamp.lmer3  7 639 661   -312     0      1          1
```

This test clearly indicates that the problem instances, which are coded in the model as `(1|fSeed)`, have no significant effect. Performing the following test shows that there are also no instance-algorithm interactions (`(1|fSeed:sreco)`:

```
> mamp.lmer1 <- lmer(yLog ~ sreco + (1|fSeed)
         , data=mamp.df, REML=FALSE)
> anova(mamp.lmer3, mamp.lmer1)

Data: mamp.df
Models:
mamp.lmer1: yLog ~ sreco + (1 | fSeed)
mamp.lmer3: yLog ~ sreco + (1 | fSeed) + (1 | fSeed:sreco)
           Df AIC BIC logLik Chisq Chi Df Pr(>Chisq)
mamp.lmer1  6 637 656   -312
mamp.lmer3  7 639 661   -312     0      1          1
```

MAMP-4b) Hypothesis Testing: Fixed Factor Effects Regarding fixed factors, we are interested in testing for differences in the factor level means $\mu + \alpha_i$. These tests can be formulated in the hypothesis testing framework as:

$$H_0 : \alpha_i = 0 \, \forall i \quad \text{against} \quad H_1 : \exists \, \alpha_j \neq 0 \tag{18}$$

Here, we are using the test statistic from [14, p. 523] for testing that the means of the fixed factor effects are equal:

```
> anova(mamp.lmer)

Analysis of Variance Table
      Df Sum Sq Mean Sq F value
sreco  3   13.5    4.51    2.35
```

Based on the F_0 value, we calculate the p value for the test on the fixed-effect term.

```
> h <- nlevels(mamp.df$sreco)
> q <- nlevels(mamp.df$fSeed)
> anova(mamp.lmer)$"F value"

[1] 2.35

> 1 - pf(anova(mamp.lmer)$"F value", h-1, (h-1)*(q-1))

[1] 0.0981
```

The obtained p value 0.1 is only of minor significance. It does not give clear evidence that `sreco` should be included in the model. However, the impact of the problem instances is negligible, because the corresponding p values are significantly larger than zero.

We can estimate the fixed factor effects $\hat{\alpha}_j$ in the mixed model as

$$\hat{\alpha}_j = \bar{Y}_{\cdot j \cdot} - \bar{Y}_{\ldots}$$

Using sum of contrasts implies that $\sum \alpha_j = 0$. The point estimates for the mean algorithm performance with the jth fixed factor setting can be obtained by $\mu_{\cdot j} = \mu + \alpha_j$. The corresponding fixed effects are shown in the `Fixed effects` section of the output from `fm2a <- lmer(yLog ~ sreco + (1|fSeed) + (1|fSeed:sreco), data=df)` on page 30. For example, we obtain the following value: `sreco1` = -0.35. Usually, we are interested in the marginal mean $\mu_{\cdot j} = \mu + \alpha_j$, whose best estimator is

$$\hat{\mu}_{\cdot j} = \overline{Y}_{\cdot j \cdot}$$

```
> (Y.j. <- with(mamp.df,aggregate(yLog,list(sreco=sreco),mean)))

  sreco     x
1     1 -5.82
2     2 -5.65
3     3 -5.18
4     4 -5.23
```

MAMP-5 Confidence Intervals and Prediction Finally, we generate paired comparisons plots, which are based on confidence intervals. The confidence interval are determined with the `VarCorr()` function, which extracts estimated variances, standard deviations, and correlations of the random-effects terms.

```
> VC<-VarCorr(mamp.lmer)
> sigma.gamma<-as.numeric(attr(VC$"fSeed:sreco","stddev"))
> sigma<-as.numeric(attr(VC,"sc"))
> MSAB <- sigma^2 + r * sigma.gamma^2
> Y.j. <- with(mamp.df,aggregate(yLog,list(alg=sreco),mean))
> s <- sqrt(2)*sqrt(MSAB/(q*r))
> T <- qtukey(1-0.05,h,(h-1)*(q-1))/sqrt(2)
> Y.j.$lower <- Y.j.$x - 0.5 * T * s
> Y.j.$upper <- Y.j.$x + 0.5 * T * s
> Y.j.

  alg     x lower upper
1   1 -5.82 -6.22 -5.42
2   2 -5.65 -6.06 -5.25
3   3 -5.18 -5.58 -4.77
4   4 -5.23 -5.63 -4.83
```

Note, that the intercept term $\hat{\mu} = -5.4694$ can be added to the estimated fixed effects to obtain the $Y.$ values, e.g., $-5.82 = -5.4694 - 0.35$ or $-5.65 = -5.4694 - 0.1841$.

The wrapper function `intervals()` from Chiarandini and Goegebeur [10] was used for visualizing these confidence intervals as shown in Fig. 10. Again,

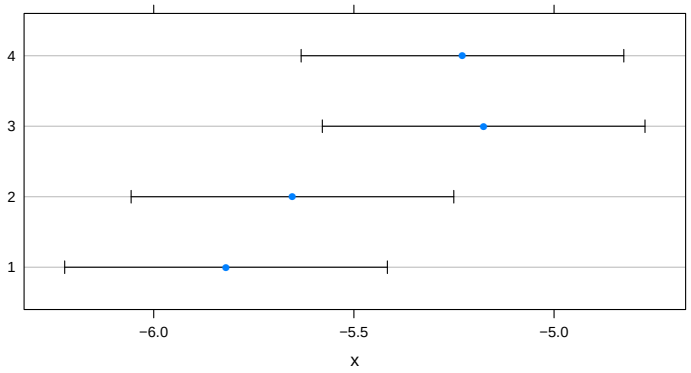


Fig. 10. Paired comparison plots. Results from four ES instances with different recombination operators are shown in this plot.

the largest difference occurs between *no* (1) and *intermediate* (3) recombination of the strategy variables.

Second Example MAMP: ES on Simple Test Data Set In the previous case study, one fixed factor was used. We now discuss the case MAMP with three fixed factors.

MAMP-1 Algorithm and Problem Instances

- `algorithm`: two ES mutation operators $\{1, 2\}$
- `algorithm`: four ES recombination operators for strategy variables $\{1, 2, 3, 4\}$
- `algorithm`: four ES recombination operators for object variables $\{1, 2, 3, 4\}$
- `instance`: nine instances randomly sampled from the class `(-0.1, 0.01, 0.001, 10.0, 10.0)`
- `replicates`: five

The 32 possible combinations give rise to 32 algorithms to test.

```
> str(mamp2.df)

'data.frame':        1440 obs. of  6 variables:
 $ y      : num  0.1218 0.0123 0.4072 0.2941 1.2331 ...
 $ mut    : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
 $ sreco  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
 $ oreco  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 2 2 2 2 2 ...
 $ fSeed  : Factor w/ 9 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ algSeed: Factor w/ 5 levels "1","2","3","4",..: 1 2 3 4 5 1 2 3 4 5 ...
```

Here, $2 \times 4 \times 4 \times 9 \times 5 = 1440$ algorithm runs were performed.

MAMP-2 Validation of the Model Assumptions The Q-Q plot reveals the non-normality of the original data. As in the previous case studies, a logarithmic transformation improves the normality. However, even the log transformed data show deviations from normality, especially in the tails of the distribution.

MAMP-3 Building the Model and ANOVA The variance decomposition, which was introduced in the previous case study, was used. The variability in the performance measure is decomposed according to the mixed-effects ANOVA model and the model equation (15) is used. We will use likelihood-ratio tests to determine significant factor and interaction effects.

MAMP-4a) Hypothesis Testing: Random Effects We include all second order interactions in our models.

```
> mamp2.lm <- lm(yLog ~ (mut + sreco + oreco)^2, data = mamp2.df)
> mamp2.lmer1 <- lmer(yLog ~ (mut + sreco + oreco)^2 + (1| fSeed),
  data = mamp2.df, REML = FALSE)
```

```
> mamp2.lmer2 <- lmer(yLog ~ (mut + sreco + oreco)^2 + (1| fSeed) +
  (1| fSeed:mut) +(1| fSeed:sreco) + (1| fSeed:oreco),
  data = mamp2.df, REML = FALSE)
> LRT <- as.numeric(2 * (logLik(mamp2.lmer2) - logLik(mamp2.lm)))
> 1-pchisq(LRT,1)

[1] 8.5e-12
```

The likelihood ratio test reveals that the random factor problem instance is significant and that there is at least one significant interaction between fixed algorithm factors and random problem instance factors. The analysis based on `anova()` gives a similar result.

```
> anova(mamp2.lmer2, mamp2.lmer1)

Data: mamp2.df
Models:
mamp2.lmer1: yLog ~ (mut + sreco + oreco)^2 + (1 | fSeed)
mamp2.lmer2: yLog ~ (mut + sreco + oreco)^2 + (1 | fSeed) + (1 | fSeed:mut) +
mamp2.lmer2:     (1 | fSeed:sreco) + (1 | fSeed:oreco)
            Df  AIC  BIC logLik Chisq Chi Df Pr(>Chisq)
mamp2.lmer1 25 5958 6090  -2954
mamp2.lmer2 28 5938 6086  -2941  26.1      3    9.1e-06
```

Therefore, we conclude that the random factor instance is significant.

MAMP-4b) Hypothesis Testing: Fixed Effects We consider the fixed effects next. The `LMERConvenienceFunctions` provides many tools for the analysis of mixed models. Here, we will use the `pamer.fnc()` for computing upper- and lower-bound p values for the ANOVA and the amount of deviance explained (%) for each fixed-effect of an `lmer` model.

```
> mamp2.fixed <- lmer(yLog ~ (mut + sreco + oreco)^2 + (1| fSeed) +
   (1| fSeed:mut) +(1| fSeed:sreco) + (1| fSeed:oreco)
   , data = mamp2.df)
> library(LMERConvenienceFunctions)
> pamer.fnc(mamp2.fixed)

            Df Sum Sq Mean Sq F value upper.den.df upper.p.val
mut          1 1046.1 1046.12 307.626         1417      0.0000
sreco        3   29.4    9.82   2.886         1417      0.0345
oreco        3   91.0   30.34   8.922         1417      0.0000
mut:sreco    3   11.6    3.85   1.134         1417      0.3343
mut:oreco    3  943.9  314.63  92.519         1417      0.0000
sreco:oreco  9   19.1    2.12   0.624         1417      0.7773
            lower.den.df lower.p.val expl.dev.(%)
mut                1318      0.0000       10.889
```

```
sreco              1318      0.0346        0.306
oreco              1318      0.0000        0.948
mut:sreco          1318      0.3343        0.120
mut:oreco          1318      0.0000        9.825
sreco:oreco        1318      0.7773        0.199
```

The analysis yields that `mut`, `oreco`, and their interaction might be significant. We use interaction plots (Fig. 11) to illustrate this behavior.

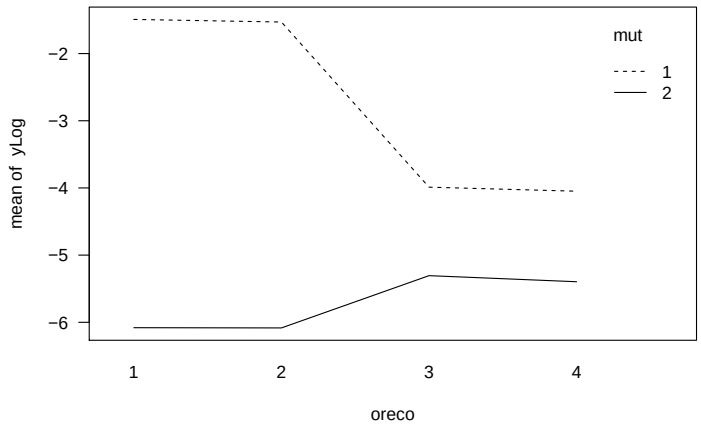


Fig. 11. Interaction plots. The solid line represents results with mutation, whereas the dotted line illustrates results obtained without mutation. Since we are considering a minimization problem, results with mutation are better than without mutation. Recombination shows possible interactions, e.g., modifying the recombination operator from dominant (2) to intermediate (3) improves the ES performance, if no mutation is used. It worsens the performance, if mutation is used.

The analysis clearly demonstrates that mutation should be used, whereas recombination worsens algorithm's performance. However, this result cannot be generalized, because we we considering a one-dimensional test function only. If no mutation is used, intermediate recombination of the object variables improves algorithm's performance.

MAMP-4c) Back-fitting The function `bfFixefLMER_F.fnc` back-fits an initial `lmer` model on upper- or lower-bound p values.

```
> mamp2.lmer3 <- lmer(yLog ~ (mut + sreco + oreco)^2 + (1| fSeed) +
   (1| fSeed:mut) +(1| fSeed:sreco) + (1| fSeed:oreco)  , data = mamp2.df)
```

First, we update initial model on trimmed data.

```
> df.trimmed = romr.fnc(mamp2.lmer3, mamp2.df, trim = 2.5)

n.removed = 50
percent.removed = 3.47

> mamp2.df = df.trimmed$data
> mamp2.lmer4 = update(mamp2.lmer3)
```

Next, we backfit fixed effects.

```
> mamp2.lmer5 = bfFixefLMER_F.fnc(mamp2.lmer4, log.file = FALSE
         , llrt = FALSE, alpha=0.005)

processing model terms of interaction level 2
        iteration 1
                p-value for term "sreco:oreco" = 0.5 > 0.005
                not part of higher-order interaction
                removing term
        iteration 2
                p-value for term "mut:sreco" = 0.0182 > 0.005
                not part of higher-order interaction
                removing term
processing model terms of interaction level 1
        iteration 3
                p-value for term "sreco" = 0.0057 > 0.005
                not part of higher-order interaction
                removing term
pruning random effects structure ...
        nothing to prune

> pamer.fnc(mamp2.lmer5)

          Df Sum Sq Mean Sq F value upper.den.df upper.p.val
mut        1 1701.0  1701.0  757.59         1382       0e+00
oreco      3   45.2    15.1    6.71         1382       2e-04
mut:oreco  3  916.5   305.5  136.07         1382       0e+00
          lower.den.df lower.p.val expl.dev.(%)
mut               1283       0e+00       20.363
oreco             1283       2e-04        0.541
mut:oreco         1283       0e+00       10.972
```

As in the full model (`mamp2.fixed`), most of the variance is explained by mutation and the interaction between mutation and recombination of the object variables. This situation was also illustrated in Fig. 11.

MAMP-5 Confidence Intervals and Prediction We will consider the average algorithm performance on the nine problem instances in Fig. 12. These data are aggregated to determine confidence intervals, which are plotted in Fig. 13. Both figures support the assumption that mutation improves the algorithm's performance. An evolution strategy with mutation, no recombination of strategy variables, and discrete recombination of the object variables performs reasonably well. Again, the `R` code written by Marco Chiarandini and Yuri Goegebeur [10] was used to determine confidence intervals and to generate the plots.

6 Summary and Outlook

This paper tries to find answers for the following fundamental questions in experimental research.

(Q-1) How to generate problem instances?
(Q-2) How to generalize experimental results?

In order to answer question (Q-1), we propose a three-stage-approach:

1. Describing the real-world system and its data
2. Feature extraction and model construction
3. Instance generation

We demonstrated how real-world problem instances with features from the time-series domain can be generated. In this setting, the proposed approach works very good. Since this approach uses a model, say M, to generate new problem instances, one conceptual problem arises: This approach is not applicable, if the final goal is the determination of a model for the data, because M is per definition the best model in this case and the search for good models will result in M. But there is a simple solution to this problem. In this case, the feature extraction and model generation should be skipped and the original data should be modified by adding some noise or performing transformations on the data. However, if applicable, the model-based approach is preferred, because it sheds some light on the underlying problem structure. For example, seasonality effects can be precisely modified, which results in an better understanding of the real-world problem and its structure.

As a test-function set randomly generated test functions are used. Algorithms with different parameterizations are tested on this set of problem instances. This experimental setup requires modified statistics, so-called random-effects models or mixed models. This approach may lead to objective evaluations and comparisons. If normality assumptions are met, confidence intervals can be determined, which "forecast" the behavior of an algorithm on unseen problem instances. Furthermore, results can be generalized in real-world settings. This gives an answer to question (Q-2).

Note, the underlying algorithm and problem designs were chosen for didactical purpose. These data are suitable for illustrating key features of the proposed methods. Therefore, algorithm and problem designs were selected as simple as

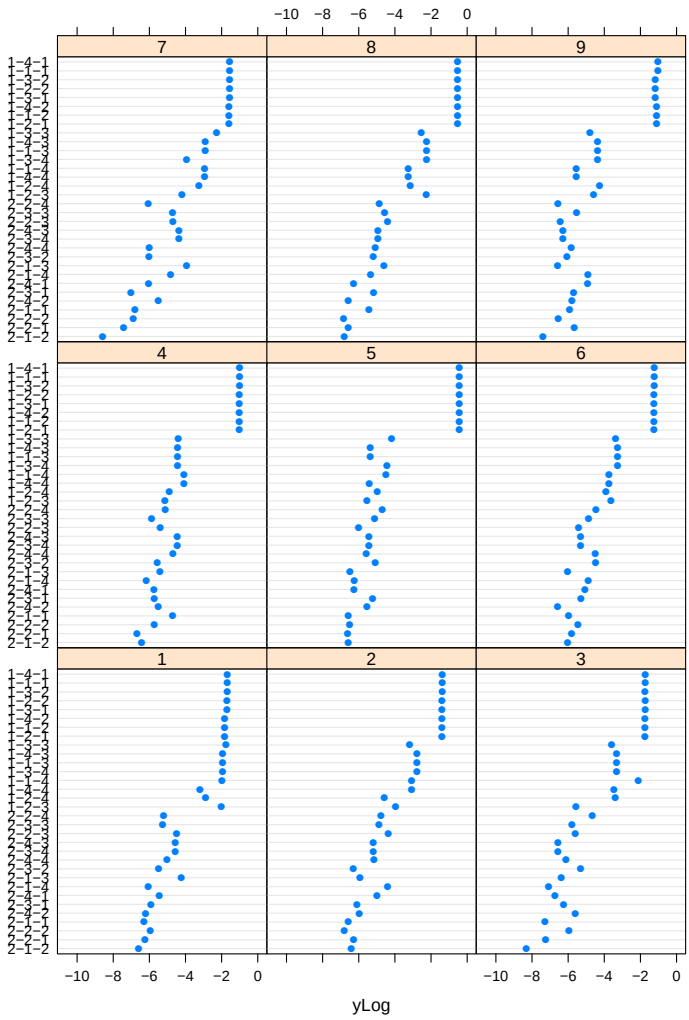


Fig. 12. Comparison of the mean values. Algorithms are classified as follows: `mut-sreco-oreco` with `mut` $\in \{no, yes\}$ and `reco` $\in \{no, discr, inter1, inter2\}$. Algorithm instance `2-1-2` performs reasonably well, i.e., mutation, no recombination of strategy variables and discrete recombination of object variables.

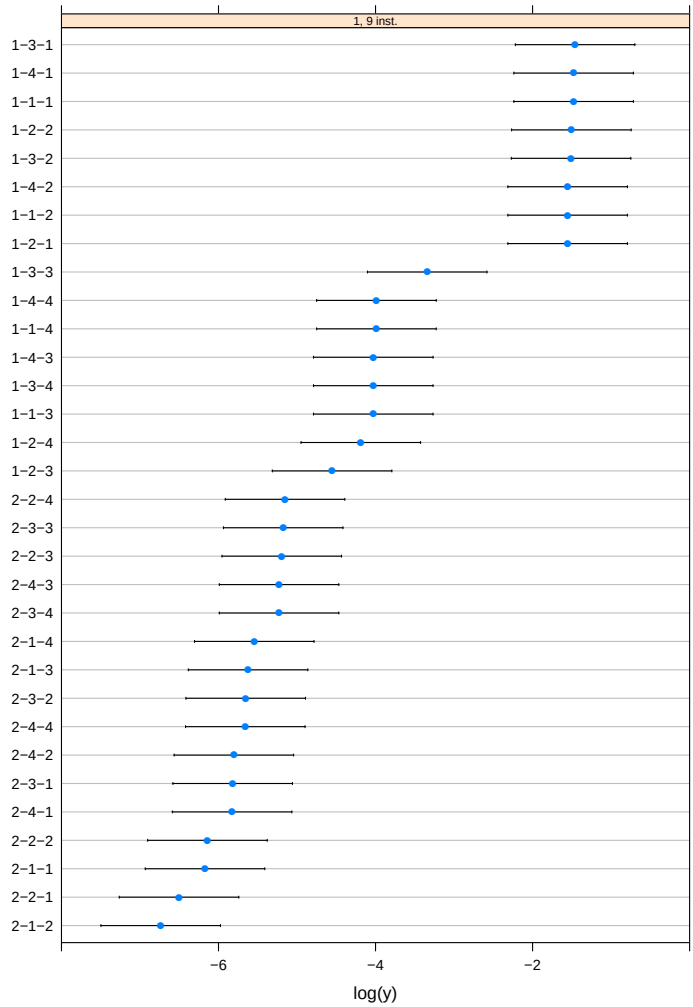


Fig. 13. Comparison of the confidence intervals. Algorithms are classified as in Fig. 12: `mut-sreco-oreco` with `mut` $\in \{no, yes\}$ and `reco` $\in \{no, discr, inter1, inter2\}$. 2-1-2 performs reasonably well, i.e., mutation, no recombination of strategy variables and discrete recombination of object variables. Note, here we are considering the problem class Π_1 in contrast to Fig. 12, where nine instances of this problem class were compared.

possible. It was not our intention to present a detailed analysis of search heuristics in this paper.

Tuning procedures such as sequential parameter optimization [3] can benefit from this framework as follows: The algorithm is tuned as usually on a fixed set of test problem instances. In a second step, the generalizability of the results has to be demonstrated on randomly generated problem instances. Future investigations might consider structural properties of the set of problem instances, e.g., linearity: if $\pi_1 \in \Pi$ and $p_2 \in \Pi$, then $(a\pi_1 + b\pi_2) \in \Pi$? And, last but not least, the concept of *algorithm based validation* [12, 4] will be used for further investigations.

The software, which was used in this study, will be integrated into the `R` package SPOT [6].

Acknowledgments This work has been kindly supported by the Federal Ministry of Education and Research (BMBF) under the grants MCIOP (FKZ 17N0311) and CIMO (FKZ 17002X11). In addition, the paper and the corresponding `R` code is based on Marco Chiarandini's and Yuri Goegebeur's publication *Mixed Models for the Analysis of Optimization Algorithms* [10]. The author highly appreciates their work.

References

1. T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation—The New Experimentalism*. Natural Computing Series. Springer, Berlin, Heidelberg, New York, 2006.
2. T. Bartz-Beielstein, M. Friese, B. Naujoks, and M. Zaefferer. SPOT applied to non-stochastic optimization problems—an experimental study. In K. Rodriguez and C. Blum, editors, *GECCO 2012 Late breaking abstracts workshop*, pages 645–646, Philadelphia, Pennsylvania, USA, July 2012. ACM.
3. T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. The sequential parameter optimization toolbox. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–360. Springer, Berlin, Heidelberg, New York, 2010.
4. T. Bartz-Beielstein, S. Markon, and M. Preuß. Algorithm based validation of a simplified elevator group controller model. In T. Ibaraki, editor, *Proceedings 5th Metaheuristics International Conference (MIC'03)*, pages 06/1–06/13 (CD–ROM), Kyoto, Japan, 2003.
5. T. Bartz-Beielstein and M. Preuss. Automatic and interactive tuning of algorithms. In N. Krasnogor and P. L. Lanzi, editors, *GECCO (Companion)*, pages 1361–1380. ACM, 2011.
6. T. Bartz-Beielstein and M. Zaefferer. A gentle introduction to sequential parameter optimization. Technical Report TR 01/2012, CIplus, 2012.
7. D. M. Bates. *lme4: Mixed-effects modeling with R*. 2010.
8. H.-G. Beyer and H.-P. Schwefel. Evolution strategies—A comprehensive introduction. *Natural Computing*, 1:3–52, 2002.
9. G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis, Forecasting and Control*. Holden-Day, 1976.

10. M. Chiarandini and Y. Goegebeur. Mixed models for the analysis of optimization algorithms. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 225–264. Springer, Germany, 2010. Preliminary version available as *Tech. Rep.* DMF-2009-07-001 at the The Danish Mathematical Society.
11. O. Flasch, T. Bartz-Beielstein, D. B. 1, W. Kantschik, and C. von Strachwitz. Results of the GECCO 2011 industrial challenge: Optimizing foreign exchange trading strategies. CIOP Technical Report 10/11, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Scienceand Engineering Science, December 2011.
12. O. François and C. Lavergne. Design of evolutionary algorithms—a statistical perspective. *IEEE Transactions on Evolutionary Computation*, 5(2):129–148, April 2001.
13. M. Gallagher and B. Yuan. A general-purpose tunable landscape generator. *IEEE transactions on evolutionary computation*, 10(5):590–603, 2006.
14. D. C. Montgomery. *Design and Analysis of Experiments*. Wiley, New York NY, 5th edition, 2001.
15. H.-P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester, U.K., 1981.
16. M. Zaefferer. Optimization and empirical analysis of an event detection software for water quality monitoring. Master's thesis, Cologne University of Applied Sciences, May 2012.

Kontakt/Impressum

Diese Veröffentlichungen erscheinen im Rahmen der Schriftenreihe "CIplus". Alle Veröffentlichungen dieser Reihe können unter
www.ciplus-research.de
oder unter
http://opus.bsz-bw.de/fhk/index.php?la=de
abgerufen werden.

Köln, Januar 2012

Herausgeber / Editorship

Prof. Dr. Thomas Bartz-Beielstein,
Prof. Dr. Wolfgang Konen,
Prof. Dr. Horst Stenzel,
Dr. Boris Naujoks
Institute of Computer Science,
Faculty of Computer Science and Engineering Science,
Cologne University of Applied Sciences,
Steinmüllerallee 1,
51643 Gummersbach
url: www.ciplus-research.de

Schriftleitung und Ansprechpartner/ Contact editor's office

Prof. Dr. Thomas Bartz-Beielstein,
Institute of Computer Science,
Faculty of Computer Science and Engineering Science,
Cologne University of Applied Sciences,
Steinmüllerallee 1, 51643 Gummersbach
phone: +49 2261 8196 6391
url: http://www.gm.fh-koeln.de/∼bartz/
eMail: thomas.bartz-beielstein@fh-koeln.de

ISSN (online) 2194-2870