Towards landscape-aware automatic algorithm configuration: preliminary experiments on neutral and rugged landscapes

Arnaud Liefooghe, Bilel Derbel, Sébastien Verel, Hernan Aguirre, Kiyoshi Tanaka

Towards Landscape-Aware Automatic Algorithm Configuration: Preliminary Experiments on Neutral and Rugged Landscapes

Arnaud Liefooghe[1,2], Bilel Derbel[1,2], Sébastien Verel[3], Hernán Aguirre[4], and Kiyoshi Tanaka[4]

[1] Univ. Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL, F-59000 Lille, France
[2] Inria Lille – Nord Europe, F-59650 Villeneuve d'Ascq, France
[3] Univ. Littoral Côte d'Opale, LISIC, F-62100 Calais, France
[4] Shinshu University, Faculty of Engineering, Nagano, Japan

Abstract. The proper setting of algorithm parameters is a well-known issue that gave rise to recent research investigations from the (offline) automatic algorithm configuration perspective. Besides, the characteristics of the target optimization problem is also a key aspect to elicit the behavior of a dedicated algorithm, and as often considered from a landscape analysis perspective. In this paper, we show that fitness landscape analysis can open a whole set of new research opportunities for increasing the effectiveness of existing automatic algorithm configuration methods. Specifically, we show that using landscape features in iterated racing both (i) at the training phase, to compute multiple elite configurations explicitly mapped with different feature values, and (ii) at the production phase, to decide which configuration to use on a feature basis, provides significantly better results compared against the standard landscape-oblivious approach. Our first experimental investigations on NK-landscapes, considered as a benchmark family having controllable features in terms of ruggedness and neutrality, and tackled using a memetic algorithm with tunable population size and variation operators, show that a landscape-aware approach is a viable alternative to handle the heterogeneity of (black-box) combinatorial optimization problems.

1 Introduction

Following the advent of increasingly complex problems coming from different application fields, and implying optimization scenarios with different properties, the optimization community is continuously pushing towards the design of novel techniques that are both effective when tackling a particular problem instance, and as generic as possible in order to be flexibly adapted to a variety of problem classes. In particular, evolutionary algorithms are extremely effective to deal with a broad range of black-box optimization problems, which is one of the major reasons of their widespread uptake. Nonetheless, and despite the tremendous knowledge gained on the design of general-purpose techniques, this success can be seriously impacted by the choice of the algorithm components and parameters. For example, when designing a genetic algorithm, one has to specify what crossover and mutation rates to set in order to reach a good performance, as well as the choice of the variation operators. Moreover, it is a fact that the robustness of an algorithm, in terms of the best reachable performance, can be directly related to to the characteristics of the problem instances being tackled. In this respect, a number of paradigms, techniques and dedicated software tools from automatic algorithm configuration have been proposed in order to alleviate the design of algorithms from the challenging and crucially important issue of setting their parameters [1–5]. Similarly, a huge body of literature from fitness landscape analysis was devoted to eliciting the features that make a problem instance fundamentally different from another, and to better grasp the behavior of evolutionary algorithms. In this paper, we aim at providing a first step in bridging automatic algorithm configuration with fitness landscape analysis, towards the achievement of a more powerful offline tuning framework.

Automatic Algorithm Configuration. Informally speaking, given a number of algorithm parameters (that might be numerical, discrete, or categorial), (offline) automatic algorithm configuration seeks a good configuration, that is a particular choice of the parameter values that best suits the solving of some *a priori* unknown instances [2]. Clearly, the motivation is not only to get rid from the burden of a manual calibration or the bias of personal and ad-hoc configuration processes, but more importantly to set up a principled approach for algorithm design, allowing to systematically explore their strengths and weaknesses when tackling a whole family of problems. In this context, several approaches have been proposed, ranging from racing [1, 2] to statistics [3], experimental design [4], and heuristic search [5].

In this paper, we focus on the iterated racing method, which is gaining a lot of popularity, especially thanks to the flexibility of the user-friendly `irace` software [6]. Racing approaches, as most existing automated algorithm configuration methods, can be viewed from a machine learning perspective as operating in a training phase followed by a test or a production phase. Based on some given instances forming the training set, the training phase is intended to learn a good configuration that would hopefully perform well when experimented later, on some new unseen instances coming from the production phase. Roughly speaking, different configurations are first evaluated in parallel by racing, and those that are performing poorly are then discarded until one single configuration remains. Since the parameter space can be huge and an exhaustive search on the training set of instances prohibitive, a biased sampling procedure is typically implemented in order to cleverly select which configurations are to be evaluated. More specifically to iterated racing [6], the sampling distribution associated with each input parameter is updated at each iteration based on some statistical tests on the performance of running the considered configurations on some instances chosen from the input training set. It has been pointed out that the way the parameter sampling procedure and the statistical evaluation of the performance of different configurations plays a key role in guiding the iterated racing process towards the most promising configurations [6]. However, and as for any machine learning technique, the properties of the training set is a key issue in order to guarantee a high accuracy of the output configuration.

To our best knowledge, this issue has been studied only to a small extent in the context of automatic algorithm configuration. In fact, although one can safely claim that a set of available instances are already known *a priori* for a particular problem class, they might have fundamentally different structural properties, thus making them not homogeneous enough to be tackled using a single configuration. The heterogeneity of training instances was discussed briefly in [6] in the context of a tuning scenario implying SAT instances and `irace`. It was argued that such a scenario can constitute a real challenge for algorithm configuration. We also argue that a single output parameter configuration might not be suitable for the target algorithm to best suit a whole set of instances having different properties. In this paper, we rather advocate for the computation of a set of configurations, not a single one, that can then be mapped accurately with respect to the characteristics of an instance. Notice that, in iterated racing, a whole set of elite configurations can be provided as output – the set of configurations that were found to statistically have similar performance, which actually happens in many tuning scenarios, especially when the number of parameters is large. Nevertheless, it is still unclear which configuration has to be chosen in practice. Additionally, it often happens that the structural properties of a production instance, that is an instance on which the algorithm was not tuned beforehand, require a seemingly different parameter settings to reach optimal performance. This is for example typically the case in black-box optimization, where no assumption is made on the structure of the fitness function. This is precisely where fitness landscape analysis comes into play.

Fitness Landscape Analysis. When tackling black-box optimization problems, for which expert domain knowledge is typically not available, a fundamental issue is to understand what makes a problem instance difficult to solve. Similarly, it is essential to elicit the performance of a randomized search heuristic in light of the structural properties of the tackled problem. In this respect, fitness landscapes analysis [7, 8] provides a set of general-purpose tools and a principled approach to systematically investigate the characteristics of an optimization problem in an attempt to guide algorithm designers towards a more in-depth understanding of the search behavior, and thus towards more effective algorithms. A typical issue addressed in fitness landscape analysis consists in studying how the performance of a given algorithm configuration can be impacted in light of insightful features from the considered problem instances. In particular, different general-purpose features were studied for this purpose [8], and such landscape features have prove their interest in successfully distinguishing between instances [9]. The general idea developed in this paper is that such features can actually serve to differentiate which parameter configuration can be more suitable for a particular problem instance, both during the training phase and during the production phase of automatic algorithm configuration. In other words, since it might be useless to search for just one single parameter configuration for an heterogeneous instance set, an alternative solution would be to consider a whole set of configurations that are explicitly associated with some elicited computable instance features. We, in fact, claim that such an idea is useful to enhance the robustness of the output configuration.

Contributions. The contributions of this paper can be stated following the next aspects:

- We adopt a landscape-oriented methodology to strengthen the accuracy of automated algorithm configuration. By partitioning the training set into different groups based on the value of landscape features, we conduct an independent training phase in parallel for each group, thus ending up with multiple algorithm configurations corresponding to the different groups. At the production phase, the appropriate configuration is selected based on the feature value of the considered instance. As a byproduct, we derive a novel landscape-aware methodology to complement existing automatic algorithm configuration in deciding on a suitable parameter setting.

- We validate the proposed landscape-aware methodology through an empirical study on the well-established benchmark family of NK-landscapes. This problem class allows us to model a black-box optimization scenario with a variety of problem instances coming from the same (pseudo-boolean) domain, but with seemingly different intrinsic characteristics. By construction, a number of features, that are often found to impact the performance of evolutionary algorithms, are in fact made controllable. This results in a particularly interesting adversary benchmark for studying the challenges that automated algorithm configuration has to face when tackling heterogenous instances. In particular, we focus on the behavior of iterated racing when tackling problems with a variable degree of ruggedness and neutrality.
- By fairly taking the extra computational cost induced by our methodology into account, we investigate the gain of deciding which parameter configuration to choose for an unseen production instance based on general-purpose low-cost computable features. Our empirical findings reveal that landscape-aware iterated racing is able to find better configurations when experimented in a conventional memetic algorithm with tunable population size, variation operators, crossover and mutation rates.

Positioning. Our work shares similarities with previous attempts from automatic configuration. In Hydra [10], a portfolio builder is used together with an automatic configuration method in order to construct a portfolio of algorithm configurations. The portfolio builder typically uses problem features to discard or add new configurations found by automatic configuration, and the method was proved effective when experimented with SAT specific tools. However, it requires both a suitable portfolio builder and a domain-specific knowledge, which can constitute a bottleneck in practice for black-box optimization. In SMAC [11], landscape features are used within the tuning process as a subset of input variables in order to construct a model predicting algorithm performance, but a single recommended algorithm configuration is returned for the whole instance set. In ISAC [12], features are used for instance-specific algorithm configuration, but the authors consider problem-specific features, whereas our proposal attempts to address black-box optimization problems.

Outline. For the sake of presentation and completeness, we first start by describing in Section 2 the rationale behind NK-landscapes, as well as by defining some general-purpose features that we shall use in order to empirically revisit the characteristics of NK-landscapes. In Section 3, which is the core of the paper, we describe the proposed landscape-aware methodology for automatic algorithm configuration and experimentally investigate its accuracy on NK-landscapes. In Section 4, we conclude the paper while providing some future research questions.

2 Initial Considerations on Pseudo-Boolean Landscapes

2.1 NK-, $\mathrm{NK_q}$- and $\mathrm{NK_p}$-Landscapes

The family of NK-landscapes constitutes a problem-independent model used for constructing multimodal benchmark instances with variable *ruggedness* [13]. The fitness function f is a pseudo-boolean function $f : \{0,1\}^N \to [0,1]$ to be maximized. Candidate solutions are binary strings of size N, i.e. the solution space is $X := \{0,1\}^N$. The fitness value $f(x)$ of a solution $x = (x_1, \ldots, x_i, \ldots, x_N)$ is an average value of the individual contributions associated with each variable x_i. Indeed, for each x_i, $i \in [\![1, N]\!]$, a component function $f_i : \{0,1\}^{K+1} \to [0,1]$ assigns a positive contribution for every combination of x_i and its K *epistatic interactions* $\{x_{i_1}, \ldots, x_{i_K}\}$. Thus, the individual contribution of a variable x_i depends on the value of x_i, and on the values of $K < N$ other binary variables $\{x_{j_1}, \ldots, x_{j_K}\}$. The problem can be formalized as follows:

$$\underset{x \in \{0,1\}^N}{\arg\max}\ f(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x_i, x_{i_1}, \ldots, x_{i_K})$$

The epistatic interactions, i.e. the K variables that influence the contribution of x_i, are here set uniformly at random among the $(N-1)$ other variables, following the random model from [13]. By increasing the number of epistatic interactions K from 0 to $(N-1)$, NK-landscapes can be gradually set from smooth to rugged. It is worth noticing that this is intended to provide a family of black-box benchmark functions that allow to study challenging aspects that can make a practical combinatorial optimization problem instance difficult to solve, such as ruggedness or multimodality [7, 13, 14].

Moreover, NK-landscape were shown to be extendable to optimization scenarios in the presence of different degrees of *neutrality*, which is also a critical issue when dealing with combinatorial optimization problems [15, 16]. Accordingly, Newman [17] and Barnett [18] introduced a controllable level of *neutrality* as follows. In the so-called quantized $\mathrm{NK_q}$-landscapes [17], the f_i-values are generated following a discrete uniform distribution $[\![0, q-1]\!]$, and are scaled down by a factor of $\frac{1}{q-1}$. In the so-called probabilistic $\mathrm{NK_p}$-landscapes [18], the f_i-values are set to 0 with a probability p, and otherwise generated as in the original NK-landscapes with a probability $(1-p)$, where p is a benchmark parameter. To summarize, we shall consider $\mathrm{NK_{q|p}}$-landscapes as described above, where it is expected that the larger K the higher the level of ruggedness, and that the smaller q (respectively the larger p) the higher the level of neutrality.

2.2 $\mathrm{NK_{q|p}}$-Landscapes Features

As mentioned earlier, fitness landscape analysis aims at studying the topology of a combinatorial optimization problem by gathering important information such as ruggedness or multimodality [7, 14]. It is important to remark that such an information is typically *not* available *a priori*, when effectively solving a given unseen problem instance. Actually, in a typical black-box optimization scenario, even the parameters that originate a particular problem instance might not be available. With respect to the $\mathrm{NK_{q|p}}$-benchmark family, we might typically consider a configuration scenario where the instance parameter values such as K, p or q, are not known by the optimizer. In this context, a fitness landscape analysis might allow us to extract valuable information on the structural properties of an instance. For this purpose, we first report some general-purpose properties of the considered $\mathrm{NK_{q|p}}$ benchmarks by taking inspiration from [18]. Our goal is also to provide empirical evidence that this benchmark family is rather heterogenous, and is indeed a good adversary candidate for evaluating the behavior of automatic algorithm configuration. We consider an instance dataset of 800 $\mathrm{NK_{q|p}}$-landscapes with a problem size $N \in [\![500, 2\,000]\!]$, an epistatic degree $K \in [\![0, 10]\!]$, and a neutral degree $q \in [\![2, 10]\!]$ for $\mathrm{NK_q}$-landscapes, respectively $p \in [0.60, 0.93]$ for

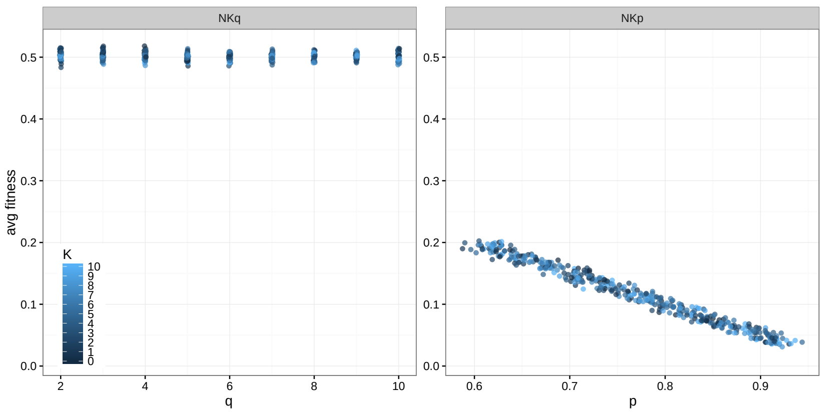

Fig. 1. Scatter plot of mufit (average fitness value) as a function of p and q for all instances.

NK$_\text{p}$-landscapes. The range of the parameters q and p have been chosen in order to obtain a similar range of neutral degrees on NK$_\text{q}$- and NK$_\text{p}$-landscapes. A total of 800 instances are considered, with one instance generated at random for each parameter combination. Half of the instances correspond to NK$_\text{q}$-landscapes, while the other half are NK$_\text{p}$-landscapes. The parameters have been generated from a design of experiments based on a latin hypercube sampling.

Formally, a fitness landscape is defined by a triplet $(X, \mathcal{N}, f)$, such that X is a set of admissible solutions (the search space), $\mathcal{N} : X \to 2^X$ is a neighborhood relation between solutions, and $f : X \to \mathbb{R}$ is a black-box fitness function, here assumed to be maximized. A simple sampling technique for examining features from the landscape is to perform a *random walk* over the landscape. More specifically, an infinite random walk is an ordered sequence $\langle x_0, x_1, \ldots \rangle$ of solutions such that $x_0 \in X$, and x_t is a neighboring solution selected uniformly at random from $\mathcal{N}(x_{t-1})$. In the same spirit than for the heterogeneous scenario mentioned in [6], a first feature that we might consider is the average fitness value of a random walk, which can be approximated by means of a finite random walk $\langle x_0, x_1, \ldots, x_\ell \rangle$ of length ℓ as follows: $\bar{f} = \frac{1}{\ell} \sum_{t=1}^{\ell} f(x_t)$. The average fitness value encountered along a random walk can actually be used to differentiate a given set of instances. This is exactly what we report in Fig. 1 for the NK$_{\text{q}|\text{p}}$-landscapes, where ℓ is set to $1\,000$. We can observe that NK$_\text{q}$-landscapes clearly differ from NK$_\text{p}$-landscapes, as the range of average fitness values is substantially different. While the instances generated with different $q-$values appear to be rather uniform in terms of average fitness value (independently of K), the average fitness value is in contrast decreasing linearly as a function of p. This provides a first hint on the differences that we might encounter in the landscape of different instances.

In order to go further in the analysis, the autocorrelation [14] between the fitness values of consecutive solutions in a random walk can be used to characterize an important feature of an instance, namely its ruggedness. We consider the following approximation to estimate the so-called autocorrelation coefficient $\hat{r}(k)$:

$$\hat{r}(k) = \frac{\sum_{t=1}^{\ell-k} (f(x_t) - \bar{f}) \cdot (f(x_{t+k}) - \bar{f})}{\sum_{t=1}^{\ell} (f(x_t) - \bar{f})^2}$$

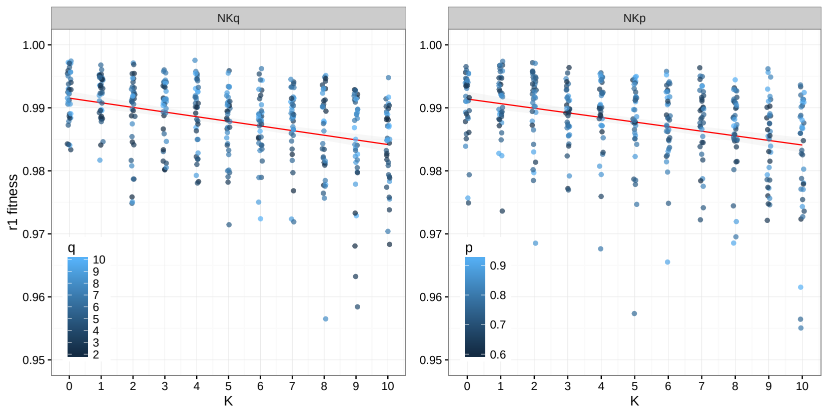

Fig. 2. Scatter plot of rho1fit (first autocorrelation coefficient) as a function of K for all instances.

We use the first autocorrelation coefficient $r(1)$ to characterize ruggedness: the larger $r(1)$, the smoother the landscape [14]. We report in Fig. 2 this coefficient as a function of K. As expected, we can observe that the first autocorrelation coefficient tends to decrease with the degree of non-linearity. This means that the larger K, the more likely to fall into a local optimum. Notice that this tendency is the same for both NK$_q$- and NK$_p$-landscapes.

At last, we shall examine a feature capturing the degree of neutrality, which explicitly relates to parameters p and q in NK$_{q|p}$-landscapes. Given a solution x, we denote a neighboring solution $x' \in \mathcal{N}(x)$ as a *neutral neighbor* if it has the same fitness value: $f(x') = f(x)$ [15]. The *neutral degree* of a solution is then defined as the number of its neutral neighbors. Consequently, different statistics can be used to quantify the neutral degree of a given instance, following different sampling strategies that induce different computational costs. Since we shall fairly include the cost of computing such features later when addressing the effectiveness of an algorithm configuration method, we consider a new estimator that solely looks at consecutive solutions along a random walk. More specifically, let $NN = \{(x_i, x_{i+1}) \mid f(x_i) = f(x_{i+1}),\ i \in \{0, \ldots, \ell - 1\}\}$ be the set of pairs of solutions with the same fitness value in the random walk. We consider the following low-cost feature to render neutrality: rateeq $= \frac{|NN|}{\ell}$, which is the proportion of pairs of neutral neighbors along the random walk. In Fig. 3, we report the neutral degree of the considered instances as a function of the different parameters K, q and p. The neutral degree decreases (resp. increases) with q (resp. p), which is with no surprise given the definition of these two parameters in NK$_{q|p}$-landscapes. However, a notable observation is that the neutral degree is relatively higher for NK$_p$-landscapes (up to 0.8) compared against NK$_q$-landscapes (up to 0.6), which is yet another interesting information about the heterogeneity of these instances. Interestingly, we clearly see that the neutral degree is not only dependent on parameters q or p, but also on the degree of non-linearity K, as previously pointed out in [18]. Actually, the higher the value of K, the lower the neutral degree. We also remark that for instances with a high level of non-linearity K, the difference in the range of neutrality between NK$_p$- and NK$_q$-landscapes decreases significantly, and the neutral degree appears to be roughly the same.

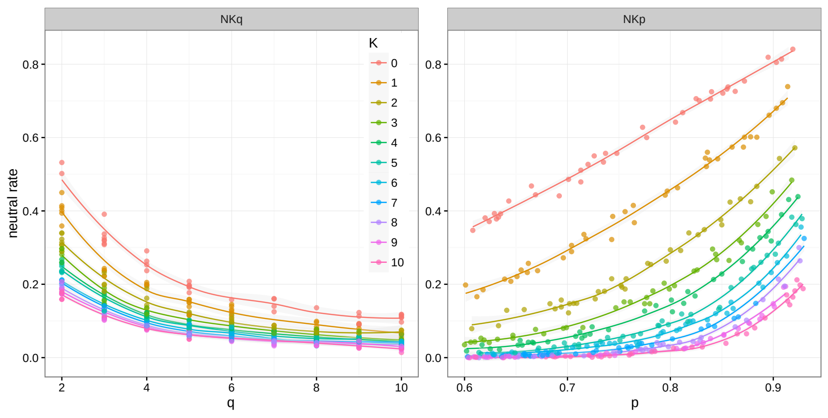

Fig. 3. Scatter plot of rateeq (neutral degree) as a function of p and q for all instances.

To conclude this section, let us emphasis that, although $\text{NK}_{\text{q|p}}$-landscapes belong to the same problem family, they are seemingly different as they expose different degrees of ruggedness and neutrality. This is likely to be the case in practice for other problem classes, where one can expect different instances to have different properties, and hence to expose different degrees of difficulty. In this respect, a reasonable hypothesis is that the optimal setting of the considered optimization algorithm depends on instance properties. This is precisely what we address in the remainder of this paper.

3 Feature-based Algorithm Configuration

In this section, we describe a feature-based algorithm configuration methodology, and provide an empirical evidence of its benefits when tuning a standard memetic algorithm.

3.1 Feature-Aware Iterated Racing

For completeness, we first start recalling the main steps of conventional iterated racing as performed in `irace` [6]. Our interest in this approach stems from its successful application in tuning different optimization techniques for a rather wide range of optimization problems [6]. The input of `irace` is a set of parameters $\theta = \{x_1, \ldots, x_n\}$ from the algorithm to be configured, and a set of training instances $\mathcal{I} = \{I_1, \ldots, I_k\}$. The output is typically a set of elite configurations $\theta^* = \{\theta_1, \ldots, \theta_r\}$ that allow the target algorithm to perform at its best with respect to some performance metric. Notice that `irace` is actually a stochastic search process performing in the parameter space, and hence no guarantee is actually provided on the optimal performance of the output configuration. That said, `irace` consists in three main steps that are repeated sequentially as follows, until a termination condition is met. First, some configurations are sampled according to a particular probability distribution. The best configurations are then selected using a racing procedure [6]. More specifically, the sampled configurations are evaluated for a number of steps by executing the algorithm with the parameter setting mapping to those configurations. At each step of the race, one instance from $\mathcal{I}$ is considered. The configurations that were found to perform statistically worse than others are then discarded, and the race continues with the surviving configurations. Finally, the distribution from where the configurations are sampled from is updated in order to bias the search towards the most promising configurations found in previous iterations. As will be detailed later, we use a standard termination criterion which is a user-defined computational budget, in terms of a number of algorithm execution. The performance metric is simply the quality of the best solution found during an algorithm execution.

At this stage, it is important to remark that `irace` is intended to be a general-purpose tuning approach. In particular, no assumption is made from the set of input training instances $\mathcal{I}$. Following the same motivations from the no-free lunch theorem, the idea developed in this paper is precisely that there cannot exist a unique optimal configuration for a whole set of instances. Consequently, `irace` can only output a configuration representing a good compromise with respect to the characteristics of all training instances. This is to contrast with an ideal case where one wants the output configuration to perform in an accurate manner to an unseen production instance, independently of its intrinsic properties. In this paper, we hence argue that a methodology where some knowledge about the landscape is considered as a helpful information from which the algorithm configuration can valuably benefit, can be of special interest. To provide an empirical evidence of the soundness of the previous claim, we propose a rather simple, yet efficient, procedure as described in the next paragraph.

We consider that an instance is characterized by the value of some landscape feature. We hypothesis that instances having similar feature values are likely to expose a similar difficulty for the target optimization algorithm, and that it can then be configured similarly for those instances. Let us denote by $\mathsf{feat}(I)$ the value of feature feat for instance I. Since we might have numerical, discrete, or even categorial features, we assume for now that we are able to classify an instance I into a unique class according to its feature value $\mathsf{feat}(I)$. Let us assume as well that we have s such classes, where s is a pre-defined parameter. We then proceed as follows: (i) we partition the training set into s groups according to the feature values, i.e. $\mathcal{I} = \mathcal{I}^1 \cup \mathcal{I}^2 \cup \ldots \cup \mathcal{I}^s$, where $\mathcal{I}^i$ contains instances from the same class; and (ii) we run `irace` independently, using every partition $\mathcal{I}^i$ separately as an input training set. Since `irace` is then executed s times on the s training sets, we obtain as output s elites configurations: $\theta_1^* \cup \theta_2^* \cup \ldots \cup \theta_s^*$, where θ_j^* maps to instances of class $j \in \{1, \ldots, s\}$. Since these output configurations are hence explicitly related to the feature class, it becomes straightforward to decide which elite configuration to choose when experiencing a new unseen production instance. More specifically, given a new unseen test instance, we first compute its feature class j, and we simply consider the elite configuration θ_j^*, computed by `irace` beforehand, in order to effectively set the parameters of the optimization algorithm for this unseen instance. Designing insightful problem features is to be understood as a challenging issue in practice, and it is worth noticing that the general-purpose landscape features for black-box combinatorial optimization that we consider in this paper do not require any expert domain knowledge. The proposed methodology is to be viewed as a first step towards the design of more sophisticated approaches, as will be discussed in more details in the conclusions. Our main goal is in fact to study at which extent a landscape-aware automatic algorithm configuration methodology could be beneficial.

Table 1. Parameter space for tuning the Memetic Algorithm (MA) for $\mathsf{NK}_{\mathsf{q|p}}$-landscapes.

parameter	**domain**	**type**
population size	$\{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1\,024, 2\,048\}$	ordinal
crossover operator	$\{$unif, 1−point, 2−point$\}$	categorical
crossover rate	$\{0.00, 0.05, 0.1, \ldots, 0.95, 1.00\}$	ordinal
mutation rate	c/N, s.t. $c \in \{0.0, 0.5, 1.0, \ldots, 9.5, 10.0\}$	ordinal

Up to now, we did not address the cost of computing the feature values, nor the computational effort devoted to the tuning task. This is an important issue when evaluating the proposed methodology. For fairness, we split the available budget B equally over the s runs of `irace`, i.e. each run $j \in \{1, \ldots, s\}$ of `irace` with $\mathcal{I}^j$ uses as termination condition a maximum number of algorithm runs which is set to B/s. Additionally, we consider to subtract the cost of computing the feature from the computational effort devoted to execute the algorithm on a given instance, both at the training phase of `irace`, but more importantly at the test or production phase, when computing the class of a new unseen instance. This is to be specified in more details in our experimental setup.

3.2 Memetic Algorithm and Parameter Space

As a case study, and in order to highlight the relevance of the previously-described methodology, we consider the configuration of the main components of a memetic algorithm (MA) similar to [19] as one alternative to solve the class of $\mathsf{NK}_{\mathsf{q|p}}$-landscapes. The MA evolves a population of candidate solutions represented as binary strings. Starting from a randomly-generated population P of size μ, the MA proceed in consecutive iterations. At each iteration, two solutions from the current population are selected using a binary tournament selection, and a new offspring is created by means of crossover followed by mutation. The crossover is applied with a fixed probability r_c. The mutation consists in flipping each bit with a probability r_m. We then use a local search to enhance the so-obtained offspring. Specifically, a first-improvement hill-climbing algorithm is implemented. Solutions at hamming distance 1 are examined in a random order, and the the first improving neighbor is selected until a local optimum is found. After a set of μ offspring solutions are created in this manner, a generational replacement is performed. The newly-generated solutions becomes the current population and the best individual from the old population replaces the worst solution if it is better than the best newly generated offspring. The algorithm terminates after a fixed number of fitness function evaluations.

The parameter space for the automatic design of the MA is given in Table 1. We consider to tune the population size, which is known to be a critical issue in evolutionary computation. We hence choose a set of values ranging from very small (1) to very large (2 048). For crossover, we consider three well-established binary string operators, namely one-point crossover, two-point crossover, and uniform crossover. The possible values for the crossover rate (r_c) ranges from 0 (no crossover) to 1 (crossover always performed). The possible values for the mutation rate (r_m) are set as a function of N (the bit-string size), and controls the number of bits that are flipped in average. Although some of these parameters could have been specified as real or integer parameters, we decided to discretize them in order to reduce the size of the parameter space in `irace`.

3.3 Experimental Setup

We use the `irace` R-package [6], that provides the reference implementation of iterated racing. As training instances, we consider the same set of 800 instances as described previously in Section 2. We consider two types of features: (i) the benchmark parameters from $\text{NK}_{\text{q|p}}$-landscapes: N, K, p or q, and the type of neutrality, where the first three are numerical and the last one is categorial (i.e. quantized or probabilistic), and (ii) the general-purpose features as discussed in Section 2, namely the average fitness mufit, the first autocorrelation coefficient $\mathsf{rho1fit}$, and the neutral rate rateeq, all computed based on a random walk of budget $\ell = 1\,000$. In order to partition the training set, we consider a one-dimensional simple strategy that takes each feature separately, and then splits the instances into a fixed number of clusters with equal range of that feature values (see Table 2). This simple partitioning strategy is to be viewed as a first step towards more sophisticated clustering strategies involving more than one feature at a time, that is left for future research. Except for the feature involving the type of instance (and where the number of clusters is two), we choose to partition the training instances into four clusters. Notice also that since neutrality can be controlled independently by parameter p or q, we combine these parameters to constitute one feature denoted $\mathsf{p|q}$, for which we also have four groups: two from NK_q- and two from NK_p-landscapes. For the test phase, we independently generate a test set of 200 instances, following the same experimental design discussed in Section 2. These additional instances are used to test the accuracy of the output configurations and are not available for `irace` during the training phase. As one can appreciate in Table 2, the instances from the training set and the test set are actually well balanced over the different clusters.

Following [6], we use `irace` with a tuning budget of 20 000 algorithm runs, where each run of the MA performs 100 000 calls to the fitness function. As previously mentioned, when the proposed feature-based methodology is experimented, we split the budget equally over the different clusters. Since we need to perform a random walk beforehand to compute the features mufit, $\mathsf{rho1fit}$, rateeq, we subtract 1 000 fitness function calls from the overall MA budget, both during the training and the test phases, in order to tune the MA in production-like conditions. Notice that, although K, p and q are typically not available for the algorithm, we still include them in our experiments for the sake of illustrating the gain one can expect from the proposed methodology.

3.4 Experimental Results

In Table 2, we report the best configuration (the first one in the elite set) found when running `irace` with the whole set of training instances, which is considered as a baseline approach (first row in the Table). We thereby report the best configurations found when combining `irace` with the proposed feature-based methodology. The most notable observation at this stage of the analysis is that a uniform crossover is always preferred, except for the second group of instances partitioned with respect to $\mathsf{rho1fit}$, together with a relatively high crossover rate (except for the third group of instances partitioned by K). However, the best-found population size varies substantially when comparing the output of the baseline `irace` and the proposed methodology. We can also remark that, when adopting a feature-based tuning methodology, the mutation rate is higher compared against the baseline setting. Although it is difficult to correlate these

Table 2. First elite configuration found by irace for each feature cluster. The first row corresponds to the configuration found when considering the whole training set.

problem feature	**cluster**	**feature range**	**# inst.** (training , test)	**pop. size**	**crossover operator**	**cross. rate**	**mut. rate**
*	—		(800 , 200)	32	uniform	0.95	5.5
N	**#0:** N	$\in [\ 501\ ,\ 877\ )$	(200 , 50)	16	uniform	0.95	6.5
	#1: N	$\in [\ 877\ ,\ 1\,253\ )$	(200 , 51)	32	uniform	1.00	6.5
	#2: N	$\in [\ 1\,253\ ,\ 1\,627\ )$	(200 , 49)	64	uniform	0.75	6.5
	#3: N	$\in [\ 1\,627\ ,\ 2\,000\ ]$	(200 , 50)	64	uniform	1.00	8.5
K	**#0:** K	$\in [\ 0\ ,\ 3\ )$	(218 , 54)	256	uniform	1.00	7.5
	#1: K	$\in [\ 3\ ,\ 6\ )$	(218 , 55)	64	uniform	0.95	7.0
	#2: K	$\in [\ 6\ ,\ 9\ )$	(219 , 54)	32	uniform	0.30	7.0
	#3: K	$\in [\ 9\ ,\ 10\ ]$	(145 , 37)	16	uniform	1.00	6.0
type	**#0:** type	$= \mathrm{NK_q}$	(400 , 100)	32	uniform	0.75	7.0
	#1: type	$= \mathrm{NK_p}$	(400 , 100)	64	uniform	0.85	7.5
p | q	**#0:** param	$\in [\ 0.600\ ,\ 0.765\ )$	(200 , 50)	64	uniform	1.00	8.0
	#1: param	$\in [\ 0.765\ ,\ 0.930\ ]$	(200 , 50)	32	uniform	0.95	7.0
	#2: param	$\in [\ 2.000\ ,\ 6.000\ ]$	(222 , 55)	32	uniform	0.80	6.5
	#3: param	$\in [\ 7.000\ ,\ 10.000\ ]$	(178 , 45)	64	uniform	0.95	7.0
avg fitness	**#0:** mufit	$\in [\ 0.031\ ,\ 0.117\ )$	(200 , 49)	64	uniform	0.95	7.5
	#1: mufit	$\in [\ 0.117\ ,\ 0.486\ )$	(200 , 51)	64	uniform	0.75	8.0
	#2: mufit	$\in [\ 0.486\ ,\ 0.501\ )$	(200 , 59)	32	uniform	0.90	6.5
	#3: mufit	$\in [\ 0.501\ ,\ 0.519\ ]$	(200 , 41)	32	uniform	0.85	7.5
r1 fitness	**#0:** rho1fit	$\in [\ 0.955\ ,\ 0.985\ )$	(200 , 50)	32	uniform	0.95	6.5
	#1: rho1fit	$\in [\ 0.985\ ,\ 0.989\ )$	(200 , 60)	32	1−point	0.90	7.5
	#2: rho1fit	$\in [\ 0.989\ ,\ 0.993\ )$	(200 , 46)	64	uniform	1.00	7.5
	#3: rho1fit	$\in [\ 0.993\ ,\ 0.998\ ]$	(200 , 44)	32	uniform	0.95	7.5
neutral rate	**#0:** rateeq	$\in [\ 0.000\ ,\ 0.044\ )$	(205 , 55)	16	uniform	0.80	7.0
	#1: rateeq	$\in [\ 0.044\ ,\ 0.085\ )$	(197 , 48)	64	uniform	0.90	6.5
	#2: rateeq	$\in [\ 0.085\ ,\ 0.193\ )$	(198 , 47)	16	uniform	1.00	6.5
	#3: rateeq	$\in [\ 0.193\ ,\ 0.841\ ]$	(200 , 50)	128	uniform	0.95	7.5

observations with the considered $\mathrm{NK_{q|p}}$-landscapes, we can clearly see that `irace` is able to seemingly find different configurations, depending on how the input training test is partitioned. We attribute this to the fact that instances belonging to the same group are expected to expose less heterogeneity for the configuration procedure.

To go further into the analysis, we evaluate, for each individual feature, how the feature-based methodology performs against the configuration obtained when mixing all the instances as in baseline `irace`. To do so, we examine the performance of the MA when experimented on 200 independently-generated testing instances. We execute the MA with every configuration for 30 runs on each test instance, while subtracting the cost of computing the features to the budget allocated to MA whenever necessary. In Fig. 4, we report the number of test instances where the configuration found by feature-based `irace` allows the MA to perform significantly better (resp. worst, and insignificantly different) than when configured using the output of baseline `irace`. For the pairwise comparison of configurations on the same instance, we use a Wilcoxon signed rank statistical test with a p-value of 0.05 and a Bonferroni correction. Overall, the proposed methodology appears to effectively enhance the baseline one, since the number of instances on which the feature-based configuration provides better results is significantly higher than the baseline configuration, independently of the considered feature. This is confirmed by the basic statistics reported in Fig. 5, comparing baseline

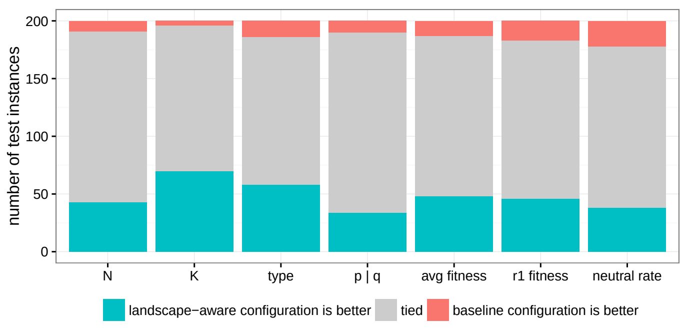

Fig. 4. Number of test instances where the landscape-aware configuration with respect to each feature is significantly better, tied or worse than the baseline configuration.

`irace` against `irace` using the feature-based partitioning. More precisely, on the left subfigure, we show the number of instances where the corresponding configuration is not statistically outperformed by any other. In the right subfigure, we report the number of times a given MA configuration is statistically outperformed by another. For a given configuration, a dot corresponds to the average rank over all test instances, where a value of 0 means that a specific configuration was actually never outperformed by any other on any test instance. Interestingly, baseline `irace` appears to identify the configuration with the largest rank. We can also see that the feature-based configuration methodology performs at its best when using K, which suggests that the non-linearity and the ruggedness of the instances is one of the most important feature one has to take into account when configuring the MA. The problem size N and the average fitness value avg fitness are also among the most insightful features when searching for a good configuration of the MA. Notice also that feature rho1fit, which is intended to approximate the ruggedness of an instance, does not allow `irace` to perform as well as with K, although it still has a better overall ranking compared to baseline `irace`. This suggests that alternative features that could approximate the ruggedness of a given instance more accurately would be worth investigating in the future.

The previous statistics aggregate the instances over the whole test set. In Fig. 6, we report a more detailed description on the relative behavior of feature-based `irace`.

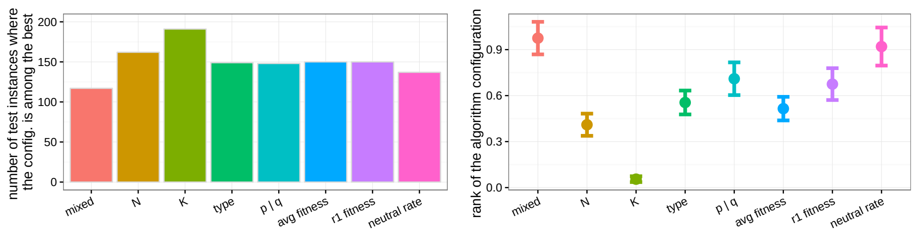

Fig. 5. Number of test instances (out of 200) where the baseline configuration (mixing all training instances) and each feature-based configuration (partitioning training instances) is not statistically outperformed by any other (left), and rank of each configuration over all test instances (right).

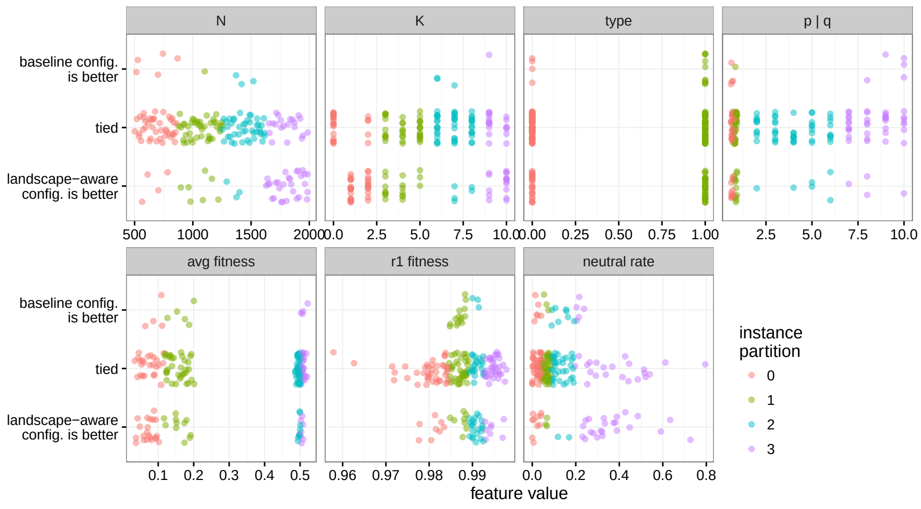

Fig. 6. Detailed distribution of test instances where the landscape-aware configuration with respect to each feature is significantly better, tied or worse than the baseline configuration, as a function of the feature value.

Specifically, the x-axis of each subfigure refers to the corresponding feature values from all test instances. Then, for each instance, the y-axis indicates whether configuring the MA with baseline `irace` provides statistically better (resp. worst, tied) performance than the proposed methodology. This allows us to investigate in more details the distribution of instances where we are able to improve or to worsen the performance of baseline `irace` by feature values. We clearly see that, overall, the feature-based methodology allows to enhance `irace`, independently of the feature values, and then independently of the characteristics of the considered instance. This is of high importance, since we can then claim that a landscape-aware automatic algorithm configuration effectively allows to improve parameter accuracy for a relatively large spectrum of heterogeneous instances.

At last, we report in Fig. 7 the results of cross-validating the performance of the different configurations that `irace` is able to obtain for each partition, with respect to a particular feature. Specifically, the x-axis refers to the group of test instances obtained by partitioning, i.e. four groups except for type. Then, for each group of test instances, we compare all other configurations that `irace` is able to find when considering either the whole set of training instances or a specific subgroup of training instances. The number of test instances where the corresponding configuration is not statistically outperformed by any other is reported in the y-axis. One should expect that, when running the algorithm configuration obtained specifically for the group of training instances to which the test instance belongs to, the performance is at its best relatively to other configurations. This is precisely what Fig. 7 is aiming to elicit. In fact, we are able to appreciate that the best-found algorithm configuration for a given group of instances is actually the best one, with some exceptions that we can likely attribute to the randomness of the algorithm configuration process itself.

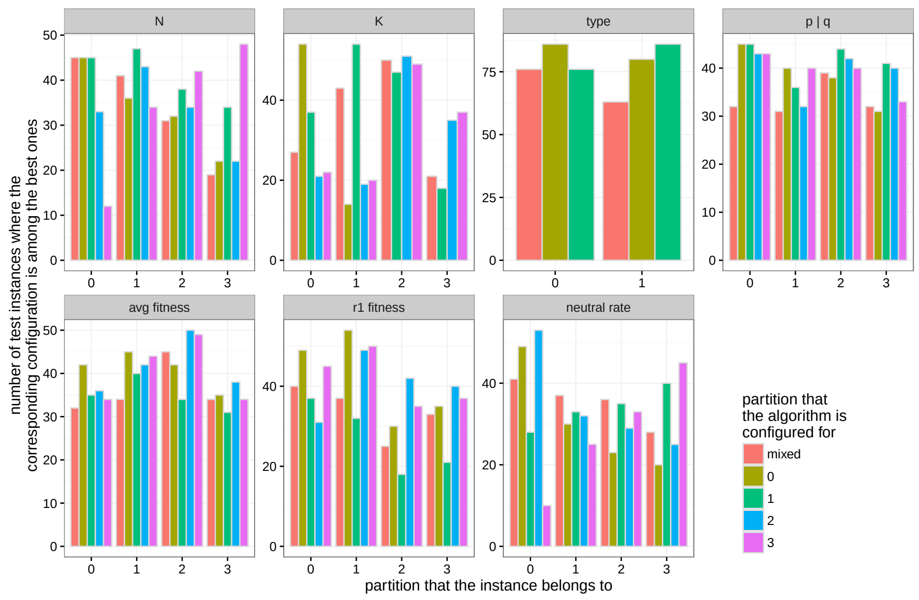

Fig. 7. Number of test instances where each landscape-aware configuration is not outperformed by any other, as a function of the feature group.

4 Conclusions

We provided a first step towards a more systematic investigation of the design of landscape-aware enhanced automatic algorithm configuration methods, which is to be understood as a a baseline for future improvements. By using the well-established iterated racing procedure to tune a standard memetic algorithm for the benchmark family of NK-landscapes, our empirical findings show that partitioning instances with respect to feature values enables to obtain more robust algorithm configurations when facing a heterogeneous set of instances. Besides, the proposed approach opens several new research questions. Firstly, the simple partitioning procedure that we adopted in this paper can be extended in different ways. Considering a multi-dimensional approach, where training instances are clustered by using multiple landscape features simultaneously, is of special interest in order to capture the similarities and differences of instances from different inter-dependent and orthogonal perspectives. Additionally, the number of groups was fixed empirically in our study, such as the global budget allowed for the whole tuning process. We believe that a more systematic investigation on the granularity of the partitioning procedure and its relation with the available budget will lead to new insightful results on the accuracy of landscape-aware algorithm configuration. Notice that the granularity of the partitioning actually opens nice opportunities for distributing the flow of the tuning procedure over different parallel cooperating entities, thus improving the quality and runtime of offline algorithm configuration, which is actually known to be time consuming. Secondly, the methodology adopted in this work does not change the way the tuning process is conducted, but simply considers the tuning procedure as a black-box mechanism. Nevertheless, we believe that the same idea of using landscape analysis to characterize instances can be seemingly used inside the tuning procedure itself, thus ending-up with new algorithm configuration methods. With respect to iterated racing, one particularly promising idea consists in carefully choosing the instances where some configuration should race at every iteration based on the features values of the instances experimented in previous iterations. At last, it would be interesting to benchmark and extend our work with other scenarios, such as different algorithms, different problems, different domains, or different tuners, and to compare our methodology with approaches from [10–12]. A particularly challenging issue is to highlight which general-purpose features can allow to provide the highest insights, and then the most accurate configurations.

Acknowledgments. We are grateful to M. López-Ibáñez for fruitful suggestions on the paper.

References

1. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Genetic and Evolutionary Computation Conference. (2002) 11–18
2. Birattari, M.: Tuning Metaheuristics: A Machine Learning Perspective. Springer (2009)
3. Bartz-Beielstein, T.: Experimental Research in Evolutionary Computation. Springer (2006)
4. Adenso-Diaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental designs and local search. Oper. Res. **54**(1) (2006) 99–114
5. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. J. Artif. Int. Res. **36**(1) (2009) 267–306
6. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Oper Res Pers **3** (2016) 43–58
7. Merz, P.: Advanced fitness landscape analysis and the performance of memetic algorithms. Evolutionary Computation **12**(3) (2004) 303–325
8. Richter, H., Engelbrecht, A., eds.: Recent Advances in the Theory and Application of Fitness Landscapes. Emergence, Complexity and Computation. Springer (2014)
9. Smith-Miles, K., Lopes, L.: Measuring instance difficulty for combinatorial optimization problems. Comput Oper Res **39**(5) (2012) 875–889
10. Xu, L., Hoos, H.H., Leyton-Brown, K.: Hydra: Automatically configuring algorithms for portfolio-based selection. In: Conference on Artificial Intelligence. (2010) 210–216
11. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Learning and Intelligent OptimizatioN. (2011) 507—523
12. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC – instance-specific algorithm configuration. In: European Conference on Artificial Intelligence. (2010) 751–756
13. Kauffman, S.A.: The Origins of Order. Oxford University Press (1993)
14. Weinberger, E.D.: Correlated and uncorrelatated fitness landscapes and how to tell the difference. Biol Cybern **63**(5) (1990) 325–336
15. Verel, S., Collard, P., Clergue, M.: Scuba search : when selection meets innovation. In: Congress on Evolutionary Computation. (2004) 924–931
16. Marmion, M.É., Dhaenens, C., Jourdan, L., Liefooghe, A., Verel, S.: On the neutrality of flowshop scheduling fitness landscapes. In: Learning and Intelligent OptimizatioN. (2011) 238–252
17. Newman, M., Engelhardt, R.: Effect of neutral selection on the evolution of molecular species. Proc. R. Soc. London B. **256** (1998) 1333–1338
18. Barnett, L.: Ruggedness and neutrality - the nkp family of fitness landscapes. In: International Conference on Artificial Life. (1998) 18–27
19. Pelikan, M.: Analysis of estimation of distribution algorithms and genetic algorithms on NK landscapes. In: Genetic and Evolutionary Computation Conference. (2008) 1033–1040