Latin Hypercube Designs with Branching and Nested Factors for Initialization of Automatic Algorithm Configuration

Simon Wessing simon.wessing@tu-dortmund.de
Computer Science Department, Technische Universität Dortmund, Germany

Manuel López-Ibáñez manuel.lopez-ibanez@manchester.ac.uk
Alliance Manchester Business School, University of Manchester, UK

Abstract

The configuration of algorithms is a laborious and difficult process. Thus, it is advisable to automate this task by using appropriate automatic configuration methods. The irace method is among the most widely used in the literature. By default, irace initializes its search process via uniform sampling of algorithm configurations. Although better initialization methods exist in the literature, the mixed-variable (numerical and categorical) nature of typical parameter spaces and the presence of conditional parameters make most of the methods not applicable in practice. Here, we present an improved initialization method that overcomes these limitations by employing concepts from the design and analysis of computer experiments with branching and nested factors. Our results show that this initialization method is not only better, in some scenarios, than the uniform sampling used by the current version of irace, but also better than other initialization methods present in other automatic configuration methods.

Keywords

automatic algorithm configuration, sampling, branching and nested designs, racing

1 Introduction

General automatic configuration methods are becoming an essential tool in the design and analysis of optimization algorithms (Bartz-Beielstein, 2006; Bezerra et al., 2016; Birattari, 2009; Hoos, 2012). Iterated racing (Balaprakash et al., 2007; Birattari et al., 2002) and, in particular, the elitist variant implemented by the irace package (López-Ibáñez et al., 2016), are among the most successful automatic configuration methods available in the literature. One key characteristic of irace, and other widely used methods such as sequential model-based algorithm configuration (SMAC) (Hutter et al., 2011), is its ability to handle complex parameter spaces containing both numerical (integer- and real-valued) and categorical (combinatorial) parameters, and also parameters that may be conditional to particular values of other parameters. For example, setting the value "Simulated annealing" of a parameter that specifies a local search may conditionally enable an additional "temperature" parameter. On the other hand, the complexity of such parameter spaces makes the exploration of the search space particularly challenging. The use of full factorial designs is impractical (Balaprakash et al., 2007) except for the most trivial parameter spaces. As a result, irace and most other methods generate the initial configurations by performing uniform sampling within the domain of each parameter, starting from the unconditional parameters and continuing down the hierarchy of conditions (Balaprakash et al., 2007). Although this approach is simple and effective, it has several drawbacks, such as the possibility of under-exploring/over-exploring some of the conditional parameters. More advanced

approaches, such as Latin hypercube designs (LHD), should in principle lead to more balanced exploration of the parameter space. However, the most basic approaches available in the literature are not well suited for such complex parameter spaces. In this paper, we investigate several LHD strategies on both well-established and newly designed algorithm configuration scenarios.

2 The Algorithm Configuration Problem

Modern algorithms in optimization, machine learning and other contexts often present a large number of parameters, either meant to be set by users according to their particular application context or encoding design decisions, default behaviors and "magic" constants fixed by the algorithm designer that could be set differently. The problem of setting these parameters to their optimal value for a particular application context may be formalized as follows.

Let us assume a parametrized *target algorithm* with n parameters conforming a parameter space $X = \{X_j, j = 1, \ldots, n\}$, each parameter X_j may be either categorical, i.e., with a discrete and typically small number of choices and no relative order among them, or numerical (real-valued or integral), i.e., with a rather large number of possible values within some range and an implicit order among them. Therefore, the domain of a categorical parameter would be given as $X_c \in D_{X_c} = \{x_{c,1}, \ldots, x_{c,k_c}\}$, while the domain of a numerical parameter would be given as $X_n \in D_{X_n} = [\underline{x}_n, \overline{x}_n]$. In practical algorithms, some parameters are often conditional on particular values of other parameters. For example, parameter X_1 may only have an effect on the target algorithm if parameter X_2 has a certain value. A configuration of the target algorithm is an assignment of a value to each parameter that is not conditionally disabled, that is, $\theta = \{x_1 \in D_{X_1}, \ldots, x_n \in D_{X_n}\}$, and Θ denotes the set of all possible configurations of the algorithm.

Let us also assume that the *target algorithm* is designed to tackle instances of some abstract problem, such as the quadratic assignment problem (QAP) (Çela, 1998). Although it may be possible to define in advance the set of problem instances of practical interest, the actual sequence of instances to be solved when the algorithm is deployed is unknown, and it can be seen as a random variable $\mathcal{I}$ from which instances may be sampled. When tackling an instance i with a configuration θ of the target algorithm, we obtain a cost measure $c(\theta, i)$ that must be minimized, without loss of generality. Examples of cost measures are the best assignment cost found for a QAP instance within a given time limit or the computation time required to find the optimal solution of a given QAP instance. If the target algorithm is stochastic, as it is often the case for evolutionary algorithms and other metaheuristics, this cost measure $c(\theta, i)$ is a single realization of a random variable $\mathcal{C}(\theta, i)$.

Since we actually do not know which instances will be solved in practice, the goal in algorithm configuration is to optimize some statistical parameter c_θ of the family of cost measures $\mathcal{C}(\theta, i)$, where i is sampled from the random variable $\mathcal{I}$. A typical definition of this statistical parameter is the expected cost of θ for any training instance $i \in \mathcal{I}$, i.e., $c_\theta = E[\mathcal{C}(\theta, i) \mid i \in \mathcal{I}]$. Thus, the optimal solution of the algorithm configuration problem is given by $\theta^* = \arg\min_{\theta \in \Theta} c_\theta$.

In practice, the precise value of c_θ can only be estimated by sampling first from $\mathcal{I}$ and then from $\mathcal{C}(\theta, i)$, that is, by selecting a set of training instances and executing a configuration of the target algorithm on them. Methods for automatic algorithm configuration differ in how they search for configurations to evaluate, how the training instances and cost measure are sampled, and how c_θ is estimated from the available samples. In the next section, we describe in detail Iterated Racing (irace), one of such methods.

3 Iterated Racing

The term *racing* describes a family of procedures for selecting the best among several alternatives over a number of stochastic (noisy) evaluations (Maron and Moore, 1997). Its main appli-

Algorithm 1 Pseudocode of I/F-Race

Require: $I = \{I_1, I_2, \dots\} \sim \mathcal{I}$,
parameter space: X,
cost measure: $\mathcal{C} : \Theta \times \mathcal{I} \to \mathbb{R}$, $\mathcal{C}(\theta, i) \in \mathbb{R}$,
tuning budget: B

1: $\Theta_1 \leftarrow \mathsf{Initialization}()$
2: $t \leftarrow 1$
3: $\Theta^{\text{elite}} \leftarrow \mathsf{Race}(\Theta_t)$
4: **while** $B^{\text{used}} < B$ **do**
5: $\quad t \leftarrow t + 1$
6: $\quad \mathcal{M} \leftarrow \mathsf{UpdateModel}(\Theta^{\text{elite}})$
7: $\quad \Theta^{\text{new}} \leftarrow \mathsf{Sample}(\mathcal{M})$
8: $\quad \Theta_t \leftarrow \Theta^{\text{new}} \cup \Theta^{\text{elite}}$
9: $\quad \Theta^{\text{elite}} \leftarrow \mathsf{Race}(\Theta_t)$
10: **end while**
11: **Output:** best configuration found from Θ^{elite}

cation context involves the evaluation of a finite number of alternative choices (such as candidate solutions, algorithm configurations, machine learning models, etc.) over a sequence of test points (such as, respectively, noisy function evaluations, problem instances, training datasets, etc.). As soon as there is evidence that some choices are worse than the best one, the former are eliminated and the race continues evaluating the surviving ones on additional test points. The goal is to quickly discard poor performing alternatives, while evaluating the best-performing ones on a higher number of test points in order to identify the best one overall. The various racing procedures mainly differ on the particular statistical tests or confidence bounds used for elimination, with some methods being more statistically conservative than others. Racing procedures have been applied to model selection in machine learning (Maron and Moore, 1997), the evaluation of candidate tours in the probabilistic TSP (Birattari et al., 2006), and the configuration of algorithmic parameters (Birattari et al., 2002; Yuan and Gallagher, 2004), among other applications. In the context of automatic algorithm configuration, the use of the Friedman test and its associated post-hoc tests or the use of pairwise t-tests without p-value correction for multiple comparisons have shown good results in practical scenarios (Birattari, 2009; Birattari et al., 2002). For large parameter spaces, it is infeasible to include all possible parameter configurations within a single race, thus iterated racing procedures combine heuristic search and racing to explore the parameter space and identify high-performing parameter configurations.

The original I/F-Race proposal (Balaprakash et al., 2007) iterates between sampling new candidate configurations from a sampling model and racing these configurations to identify the best ones. These *elite* configurations are then used to modify the sampling model in order to bias the generation of new configurations towards the best ones found so far. This process is repeated until a maximum computational budget is reached and the best configurations found are returned to the user.

Algorithm 1 gives a high-level description of the I/F-Race algorithm. As a first step, a population of candidate solutions is initialized by uniform sampling (line 1) from the parameter space X. The size of this initial population (Θ_1) is dynamically computed, according to the number of decision variables and the computational budget (B). The configurations of this population are then raced (line 3) as described above. During the race, configurations are evaluated on a number of training instances and the worst-performing configurations are discarded.

The remaining configurations, called *elites* (Θ^{elite}), are then used to update a sampling model $\mathcal{M}$ from which new configurations are probabilistically sampled (line 7). The variance of this sampling model is successively reduced as the number of iterations increases in order to focus the search around the best configurations found. The number of new configurations (Θ^{new}) sampled is dynamically decided by irace depending on the remaining budget of evaluations and the number of races performed so far. The new configurations, together with the current elite ones, are raced again (line 9) on new training instances. Sampling and racing are iterated until reaching a maximum number of target algorithm runs or another termination criterion evaluations (B).

Starting from version 2.0, the irace package (López-Ibáñez et al., 2016) implements an *elitist* variant of the above procedure. In this elitist iterated racing (henceforth simply called irace), the result of the evaluations is transferred across successive races, which was not the case in the original I/F-Race proposal. More importantly, elite configurations remain protected from being discarded until all other configurations in the race have been evaluated on the same number of training instances. This prevents discarding the best-so-far configuration, which may have been evaluated on tens of instances in previous races, after seeing only a few instances in the new race. On the other hand, new non-elite configurations are discarded as usual, that is, without evaluating them on as many instances as the elite ones. To counter-balance this advantage of the elite configurations, each race starts by evaluating elite and non-elite configurations on at least one new instances. This prevents the search getting stuck on the same elites always performing better on a few number of instances.

3.1 Initialization by Random Uniform Sampling

As mentioned above, the initialization method in irace (line 1) is based on random uniform sampling of the parameter space. First, parameters are ranked according to the hierarchy of conditionals that enable them. That is, all parameters that are not conditional on any other parameter are ranked first, parameters that are only conditional on unconditional parameters are ranked second, and so on and so forth. Next, starting from the first ranked parameters, each parameter X_j is considered iteratively by sampling a value uniformly from its domain D_{X_j}. In the case of conditional parameters, if the condition for enabling it is not satisfied by the already sampled parameters, the parameter requires no value and no sampling is done. The final result is a configuration where all parameters whose conditions are satisfied have a value within their domain.

The above procedure has the advantages of being simple and always producing valid configurations. On the other hand, although unconditional parameters are uniformly sampled, the conditional parameters are often not. Instead, the number of values sampled per conditional parameter depends on the particular conditions and the chance that those conditions are precisely satisfied by the uniform sampling.[1] Moreover, even in the case of unconditional parameters, a finite number of samples drawn randomly uniform is naturally quite nonuniform, due to the independence of the individual draws. Therefore, in this work we investigate alternative initialization methods based on Latin hypercube designs.

3.2 Related Work

Various algorithms exist for automatic algorithm configuration. Some of them use fractional factorial designs to initialize the search for good parameters (Adenso-Díaz and Laguna, 2006;

[1] In principle, other random sampling strategies are possible, for example, by considering "no value" as a possible value of the domain and sampling uniformly all parameters at once. Although such strategy may lead to a more uniform sampling of conditional parameters, ensuring that conditions are satisfied after sampling may prove very difficult and require a costly rejection or repair procedure, which will necessarily bias the sampling.

Coy et al., 2001), others recommend LHDs (Bartz-Beielstein et al., 2010). However, only a few approaches besides irace, such as SMAC (Hutter and Ramage, 2015), are capable of handling conditional parameters.

The R package ParamHelpers (Bischl et al., 2017) provides some sampling capabilities for designs with nested factors, i.e., conditional parameters. It delegates the sampling in each nested subspace to a latin hypercube sampling (LHS) function from the lhs package (Carnell, 2016). Thus, the properties of the combined design cannot be optimized, as we will do in the following. Instead, the approach taken in ParamHelpers has some similarity to partially stratified sampling (Shields and Zhang, 2016). The latter also partitions the space into lower-dimensional subspaces and the full-dimensional sample is obtained by binding the lower-dimensional samples together, but without any notion of a hierarchical structure of the space. ParamHelpers also uses the LHS functions to sample integer parameters, and only afterwards maps the real values to the final discrete ones. This is another disadvantage, because potential optimizations of the LHS distribution will be broken by this procedure.

4 Design and Analysis of Computer Experiments with Branching and Nested Factors

Latin hypercube designs (Audze and Eglājs, 1977; McKay et al., 1979) are a classical approach for generating space-filling designs for computer experiments. An LHD is defined as a set of points $D = \{\vec{z}_1, \ldots, \vec{z}_N\}$, where each set $\{z_{1,j}, \ldots, z_{N,j}\}$, $j = 1, \ldots, n$, is a random permutation of the numbers $1, \ldots, N$. D can then be scaled to the region of interest to obtain a set $P = \{\vec{x}_1, \ldots, \vec{x}_N\}$ for the sampling.

LHDs can be interpreted as an extreme case of partially stratified sampling (Shields and Zhang, 2016), in which n one-dimensional subspaces are stratified independently, and then bound together. LHDs possess very uniform one-dimensional projections of the points, which reduce variance associated with main effects (i.e., the effect of an independent variable on a dependent variable averaging across the levels of any other independent variables) of Monte Carlo estimators (i.e., an aggregated value obtained from a discrete sample of points). In other words, LHDs can reduce variance in cases where the interaction between variables is low, or where some variable has no or only weak influence on the response f. In our application, f represents the performance of a configured algorithm on a test set of problem instances, the variables are the algorithmic parameters to be configured, and Monte Carlo estimators are obtained by running algorithm configurations on problem instances.

Unfortunately, the uniformity of random LHDs in the n-dimensional space is generally not better than random. To also reduce variance associated with interaction effects, we have to increase the uniformity of this distribution. A conceivable approach to do this would be to maximize the minimal distance between points in the LHD ("maximin" approach), but if the points are not perturbed, this criterion yields many ties. To further discriminate among them, one could also regard the second-smallest, third-smallest, ...distance, as proposed by Morris and Mitchell (1995). These lexicographic comparisons are often avoided as well, because they are not amenable to treatment by gradient methods. Instead, potential energy criteria $E_\lambda(P)$, from the related area of molecular conformation problems (Müller and Sbalzarini, 2012), are common substitutes. These energy functions are useful regularizations for the continuous search spaces of molecular conformation or sphere packing problems (Addis et al., 2008). They can also be used instead of discrepancy in numerical integration theory (Damelin et al., 2010). Their basic formulation reads as

$$E_\lambda(P) = \sum_{h=1}^{N} \sum_{i \neq h} \frac{1}{d(\vec{x}_h, \vec{x}_i)^\lambda} \ , \tag{1}$$

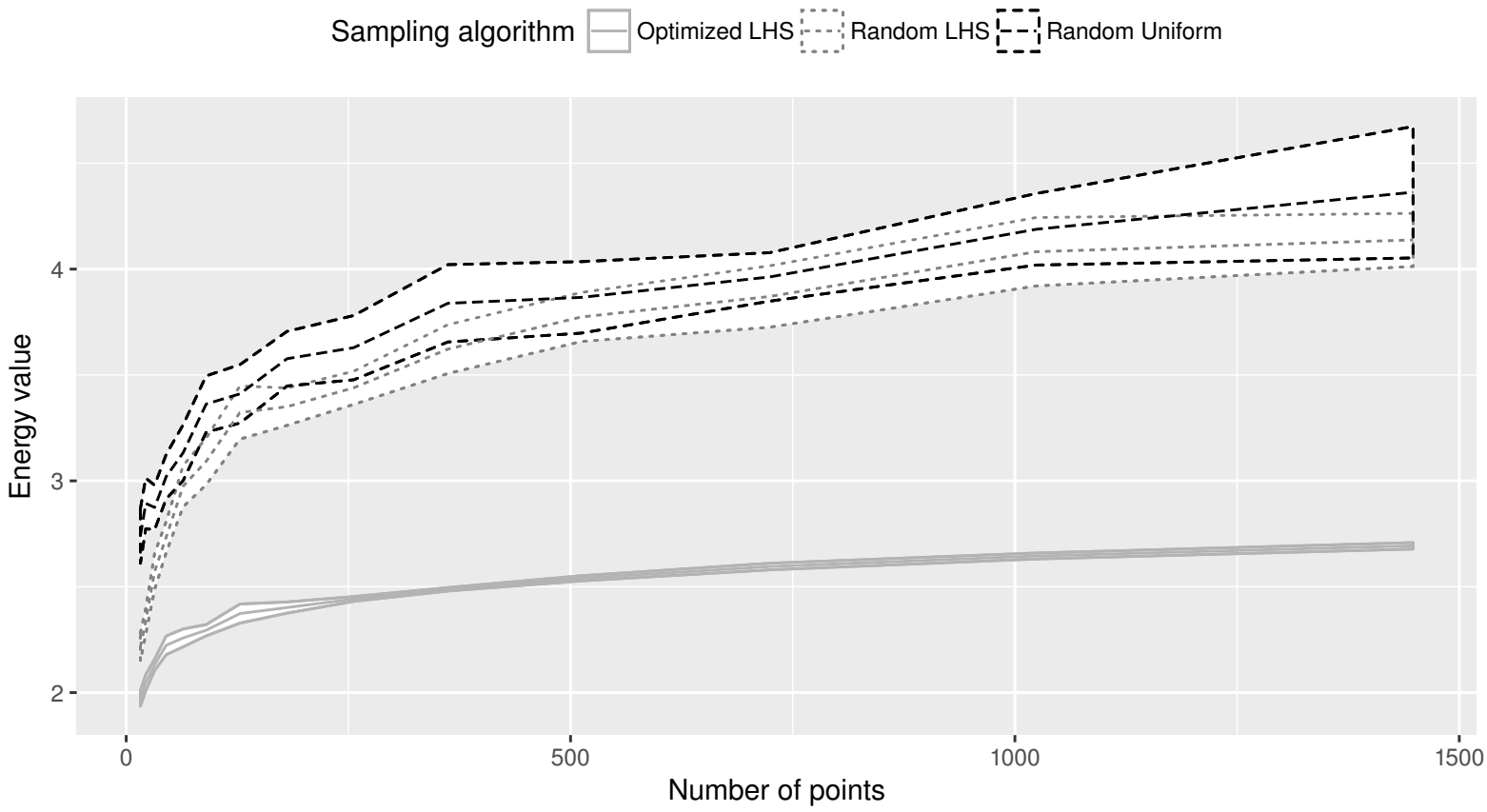


Figure 1: Mean energy values and their 95% confidence intervals against the number of points for different sampling methods (lower is better). The samples were drawn in $[0, 1]^5$.

where the $d(\vec{x}_h, \vec{x}_i)$ are pairwise distances. Such a criterion was used by Morris and Mitchell (1995) with various values of λ. Audze and Eglãjs (1977) are widely credited as the first ones to use such an energy criterion for LHDs, but only with the special case $\lambda = 2$. Figure 1 compares (normalized) energy values of random uniform sampling, random LHS, and optimized LHS in the five-dimensional unit hypercube. It is obvious that the energy value of random LHS is not significantly different from random uniform sampling, except for very small sample sizes.

We would like to point out that in the discrete search space of LHDs, the regularization is not strictly necessary, and also direct search methods in continuous spaces can deal with lexicographic comparisons. Furthermore, setting the parameter λ is not completely trivial. It is well known that for $\lambda \to \infty$, minimizing (1) becomes equivalent to maximizing the minimal distance between all pairs of design points (Santner et al., 2003, p. 139). Hardin and Saff (2004) also showed that for n-dimensional manifolds, asymptotically uniformly distributed point sets minimize this energy if $\lambda \geq n$. But if λ is chosen smaller, the optimal point density increases towards the outer regions of the manifold, which is often undesired. While the works of Audze and Eglãjs (1977) and Morris and Mitchell (1995) predate the result of Hardin and Saff (2004) by many years, it seems that even today, this fact about λ has not been fully recognized by the LHD community (Hung et al., 2009; Pronzato and Müller, 2012). However, we will use an energy criterion in the following, and not the lexicographic comparison, especially as it is much easier to visualize.

4.1 Branching and Nested Designs

An experimental design where some factors exist only for certain levels of other factors is called a branching and nested design. The dependent factor is called nested and the factor it depends on is called a branching factor. In the algorithm configuration context, a nested factor is a conditional parameter, whereas the branching factor is the parameter that appears in the condition

that enables the conditional parameter. Hung et al. (2009) present two conflicting optimality criteria for such LHDs with branching and nested factors (BLHDs). The first one is a generalization of the energy criterion (1). They explicitly assume there are q qualitative branching factors $z_1, \ldots, z_q$, with each z_u having k_u levels and m_u nested factors under each of these different levels. Let t be the number of shared factors (i.e., unconditional parameters), then their generalized energy criterion is

$$\phi'_\lambda = \left(\sum_{\vec{g} \neq \vec{h}} \left[\frac{t}{d_x(\vec{g}, \vec{h})} \right]^\lambda + \sum_{u=1}^{q} \sum_{i=1}^{k_u} \sum_{g_{\delta_u} = h_{\delta_u} = z_{u,i}} \left[\frac{m_u + t}{d_{v_u}(\vec{g}, \vec{h}) + d_x(\vec{g}, \vec{h})} \right]^\lambda \right)^{1/\lambda} , \tag{2}$$

with $\vec{g}$ and $\vec{h}$ being n-dimensional vectors and d_x, d_v projected Manhattan distances between them, regarding the shared and nested subspaces, respectively. δ_u denotes the index of the u-th branching factor in the sequence of all factors. We have to modify this scenario slightly, because we want to admit arbitrary conditions for the nested factors in irace, not just dependencies on certain levels of qualitative branching factors. Actually, this even simplifies the formula slightly to

$$\phi_\lambda = \left(\frac{1}{\binom{N}{2}} \sum_{\vec{g} \neq \vec{h}} \left[\frac{t}{d_x(\vec{g}, \vec{h})} \right]^\lambda + \sum_{u=1}^{q} \frac{1}{\binom{|P_{c_u}|}{2}} \sum_{\substack{\vec{g} \neq \vec{h} \\ \vec{g}, \vec{h} \in P_{c_u}}} \left[\frac{m_u + t}{d_{v_u}(\vec{g}, \vec{h}) + d_x(\vec{g}, \vec{h})} \right]^\lambda \right)^{1/\lambda} , \tag{3}$$

when we now say that we have q distinct conditions $c_1, \ldots, c_q$, each c_u having m_u nested factors, and P_{c_u} denotes the subset of points for which condition c_u is fulfilled. We also divide the measure by the number of used distances, to remove its influence in the spirit of Santner et al. (2003, p. 139). We choose $\lambda = n+1$ here to achieve uniformity (Hardin and Saff, 2004). The total dimension is $n = t + s$, with $s = \sum_{u=1}^{q} m_u$.

A second sensible criterion, according to Hung et al. (2009), is to minimize the pairwise correlation among factors (generalized correlation criterion for BLHDs). Surprisingly, the original description is not completely clear with regard to which correlations are actually calculated. However, it is certain that at least $t(t-1)/2$ pairwise correlations ρ_{ij} between the t shared factors and st correlations between shared and conditional factors should be calculated. There may also be pairs of conditional factors with non-empty intersection, which we also take into account. As we are only interested in absolute correlation, the values are squared. Our adapted correlation criterion thus reads

$$\rho^2 = \frac{\sum_{i=2}^{st} \sum_{j=1}^{i-1} \rho_{ij}^2}{st(st-1)/2} . \tag{4}$$

5 Multilevel Optimization of irace Configurations

Energy and correlation represent two objectives for the quality of experimental designs. Hung et al. (2009) derive lower and upper bounds to normalize (2), and then aggregate (4) and the normalized (2) into a weighted sum. They optimize the resulting scalar function with simulated annealing. We will study different approaches here instead, because Hung et al.'s normalization is cumbersome and contains the previously mentioned explicit assumption of branching factors. Besides the two individual criteria, we will consider a Pareto dominance criterion and $\phi_\lambda + \log_{10}(\rho^2)$ as a much simpler aggregated function.

The conditional parameters in irace do not necessarily depend on categorical parameters as assumed in Hung et al. (2009), but can also depend on numerical parameters $X_n \in D_{X_n}$, which are part of the BLHD. Thus, by varying one part of the solution, one might invalidate

another part. In other words, the number of points sampled for some conditional parameters of the BLHD may depend on the chosen location of coordinates in other shared or conditional parameters of the BLHD. A possible approach to tackle this problem is to use multilevel optimization, thanks to the acyclic nature of these dependencies (Deb and Sinha, 2009). We begin with optimizing the unconditional parameters at the lowest level. The optimized design in this subspace is then fixed for the subsequent iterations with conditional parameters. The number of levels is conceptually unlimited and follows in practice from the dependency structure of the parameters.

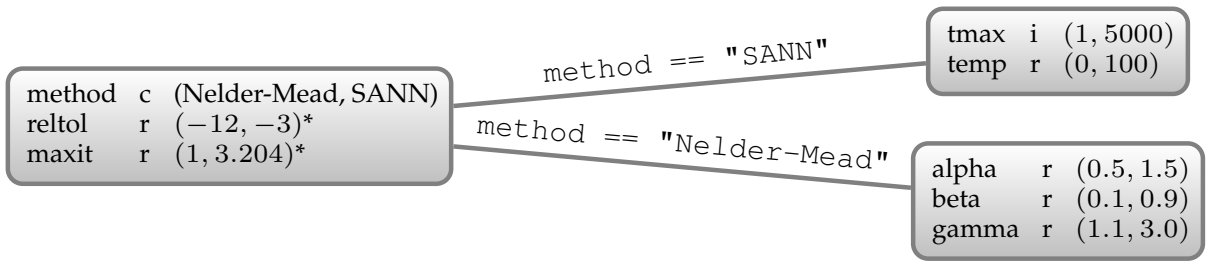
Thanks to the fixed data for previous levels, also the dimension and the number of points for the current level are fixed, and we only have to optimize a conventional LHD. To do the optimization, we use a simple $(1+1)$ evolutionary algorithm. The genotype can be represented by an integer-valued matrix, containing the LHD in row-major order. This matrix is scaled to the parameter space and mapped to the appropriate places in the set of configurations Θ. The columns of the matrix hold the permutations mentioned in Sec. 4. As a mutation operator, we choose $\max\{1, \mathcal{B}(n, 1/n)\}$ columns randomly for modification, where $\mathcal{B}(n, 1/n)$ is the binomial distribution, and then apply the swap mutation to each chosen column, to retain the permutation property (Eiben and Smith, 2003, p. 45). In consequence, the identity is not permitted as a mutation.

6 Configuration scenarios

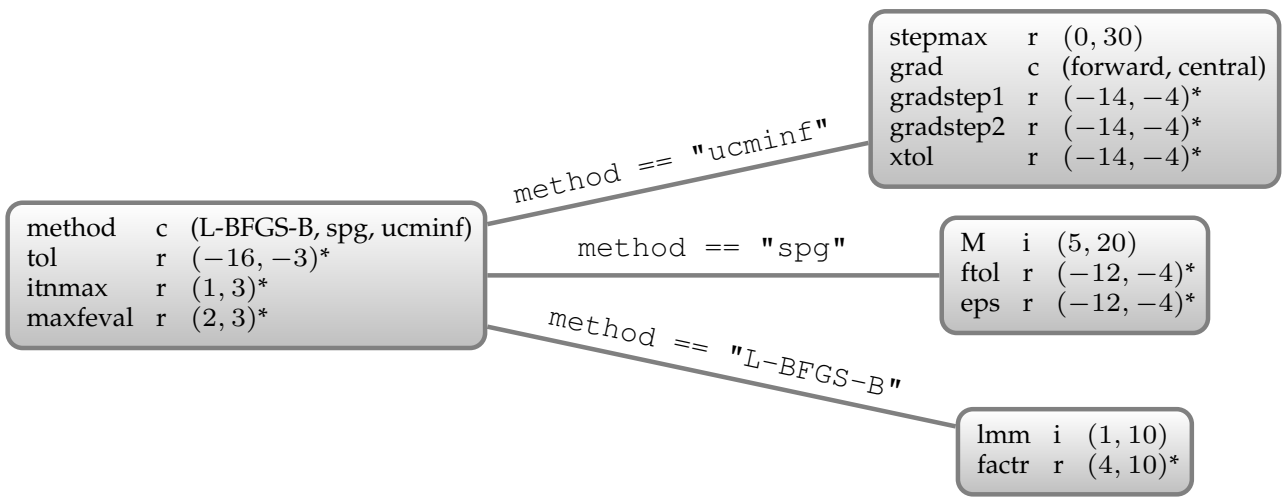
We evaluate experiments on three different configuration scenarios, the ACOQAP scenario (López-Ibáñez et al., 2018), which is a larger version of the the well-known ACOTSP scenario (Hutter et al., 2014; López-Ibáñez et al., 2016; Stützle, 2002), and two scenarios based on optimization algorithms from R's `optim` function and optimx package (Nash and Varadhan, 2011).

ACOQAP. This scenario applies a component-wise framework of various ant colony optimization (ACO) algorithms (López-Ibáñez et al., 2018) to instances of the quadratic assignment problem (QAP). The parameter space consists of 17 parameters, five of which are categorical and the rest are numerical. There are 10 nested parameters and four branching parameters. For all nested parameters, their condition contains only one branching parameter. There are five nested parameters that depend on the same parameter with different conditions, and another group of three nested parameters that depend on a different parameter with different conditions. We use a set of 50 random-structured QAP instances of size 100. This scenario is computationally expensive, because the algorithm configurations are run for 60 CPU-seconds on each instance. More details about this scenario can be found in the original publication (López-Ibáñez et al., 2018).

Optim and optimx. We have also designed two cheaper scenarios modeling restarted local search approaches in continuous optimization. This way, we could use general stopping criteria related to tolerance, number of iterations, and number of function evaluations as shared parameters. Conditional parameters are the individual parameters of the local searches. The parameter space of these two scenarios is described in Fig. 2. The first, smaller scenario (optim) uses the Nelder-Mead and simulated annealing algorithms available in R's standard library. Both are derivative-free methods. They are applied to 50 randomly weighted sums of Ackley's and Rosenbrock's functions in the search space $[-5, 5]^4$. This approach is chosen as a simple way to generate a large diverse set of problem instances. An optimization run is stopped after 1600 function evaluations, which is the same stopping criterion as in the regular single-objective optimization tracks of the black-box optimization competition (BBComp) (Loshchilov and Glasmachers, 2017). The other scenario (optimx) uses the SPG, UCMINF, and L-BFGS-B



(a) optim



(b) optimx

Figure 2: Parameter spaces for the two newly defined algorithm configuration scenarios. On the left hand side, we see the shared parameters. Edges annotated with the conditions lead to the nested factors. The three columns in each node show the parameter name, type (**r**eal, **i**nteger, **c**ategorical), and domain, as they appear in the parameter files for irace. Rows with a * indicate that a $\log_{10}$-transformation is applied.

algorithms, which are gradient and quasi-Newton methods available through R's optimx package (Nash and Varadhan, 2011). They are applied to the same problem instances used in the optim scenario, but here we stop each optimization run after 400 function evaluations, because they employ gradient information. This stopping criterion is the upper bound of the expensive single-objective tracks of BBComp.

7 Experiments

Research question. Does an optimized initial sampling lead to a measurable improvement in the performance of the best found configuration in this sample?

Setup. As described above, we evaluate six different sampling methods. These include four variants of the evolutionary multilevel optimization proposed in Section 5, which differ in the quality criteria used for optimizing the LHD: the *energy* criterion (ϕ_λ), the *correlation* criterion (ρ^2), the *weighted sum* of the two ($\phi_\lambda + \log_{10}(\rho^2)$), and a selection employing a *Pareto-dominance* relation based on the two criteria, where every improvement in terms of this dominance relation is accepted. As a reference method, the fifth initialization method is the improved LHS algorithm by Beachkofski and Grandhi (2002) available in the ParamHelpers package, and identified as *PH-ILHS* in the following. Finally, the sixth method is the *random uniform* sampling available in irace.

Pre-experimental planning. Using each sampling method, we generate a set of parameter configurations in the parameter space of each scenario. Before we run the configurations sampled on the actual problem instances, which is computationally costly, we do a sanity check on

the sampling methods by evaluating their resulting samples in terms of correlation and energy criteria. For this comparison, 100 configurations are sampled with each of the six sampling method for the three configuration scenarios. The budget for the four optimized variants is 500 evaluations of the quality criteria per condition. The whole process is replicated 50 times.

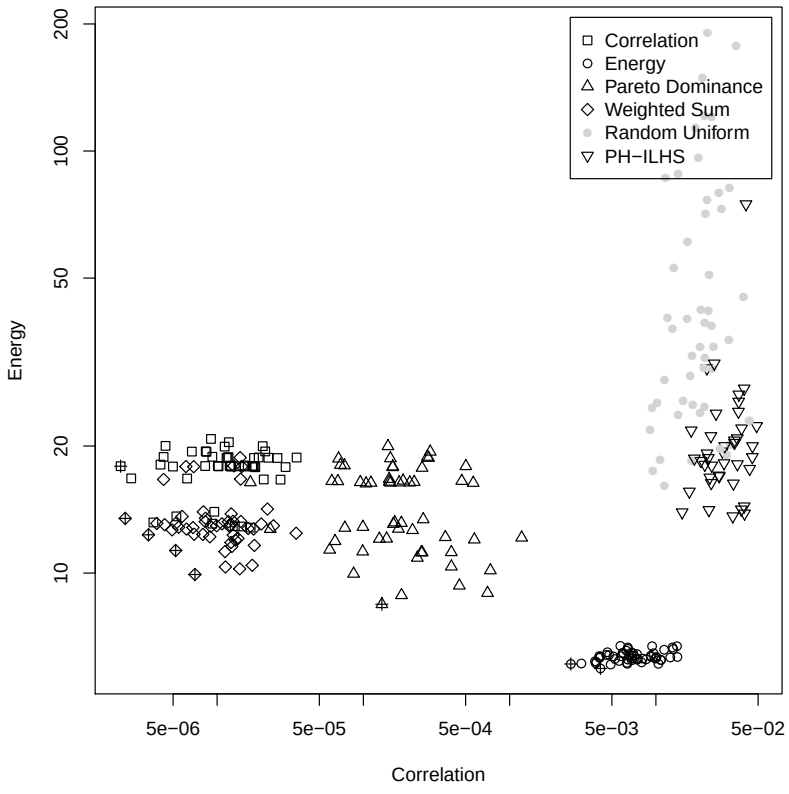
The results of this preliminary investigation are shown in Fig. 3. In every scenario, we see roughly the same effects, with the exception of a bimodal energy distribution for approaches including correlation as criterion in Fig. 3a. The method using Pareto dominance usually obtains slightly worse correlation values than using weighted sum, but slightly better energy values. The energy criterion naturally provides the lowest energy values, and also slightly improved correlation values compared to random uniform sampling. PH-ILHS achieves values similar to random uniform sampling.

Task. For each scenario, we again sample new configurations using each of the sampling methods, and we run each configuration on all the benchmark instances of the scenario. In particular, we sample 50 and 200 configurations (number of points in the LHD) for the optim scenario, however, we can only afford to sample 50 configurations for the optimx and ACOQAP scenarios, due to their much larger computational cost. We test two values for the budget assigned to the four optimized sampling variants, 500 and 2000 evaluations of the quality criteria per condition. We only focus here on the quality of the initial sample and not the quality after running irace, thus, we have no feedback loop and a separate training set of instances is not necessary for this experiment, as it would only introduce additional noise. The objective values returned by each configuration on all instances are averaged. The performance of the sampling method is then determined as the best mean objective value from all the configurations it sampled. By replicating this procedure a number of times, we obtain a mean estimator for this performance measure. In particular, we evaluate 500, 1000 and 50 replications for optim, optimx and ACOQAP, respectively. We consider a difference in means statistically significant when 95% confidence intervals shown in the figures do not overlap.

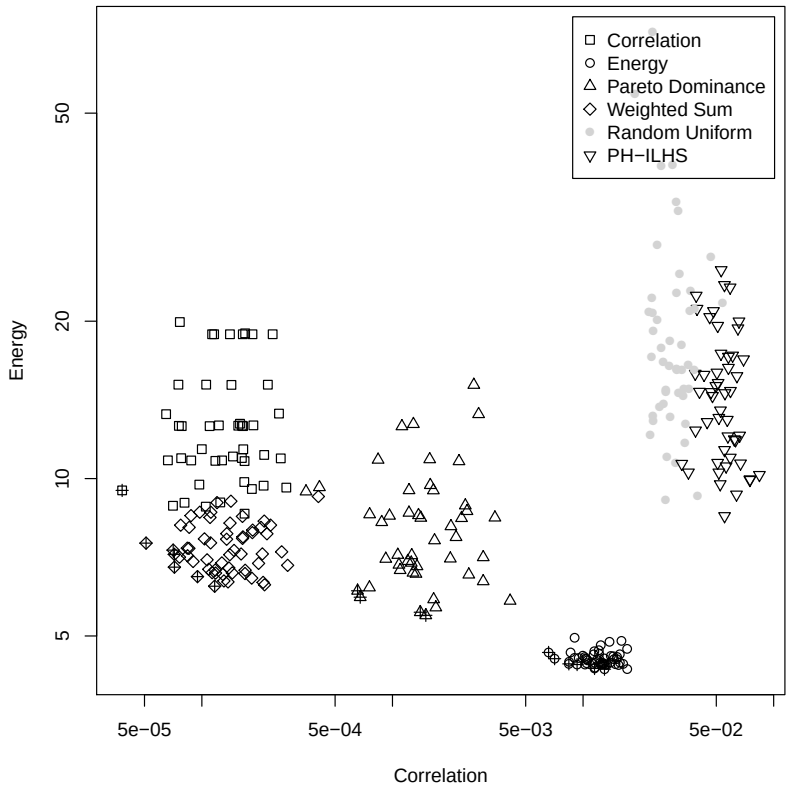
Results and observations. Figure 4 shows the mean values of the sampling methods on the optim scenario. The values for PH-ILHS and random uniform sampling are independent of the budget and thus listed under "budget: NA". We can see that optimization by energy always yields the best designs in this scenario. The effect is clearer for smaller numbers of points (configurations) and larger budget given to the optimized sampling. The performance of PH-ILHS is not significantly different from that of random uniform sampling. The optimx scenario is shown in Fig. 5, where we can also observe a positive effect of the optimized LHDs on performance. However, the variance is considerably higher and the ranking within the optimized variants is quite variable. Only few effects are statistically significant.

For the ACOQAP scenario, the high computational requirements prevented a similarly high number of replications as in the previous scenarios, hence only 50 replications were taken in this case. Figure 6 shows violin plots of the results, which uses kernel density estimation to compare the whole distributions of the sampled data (shown in white). Additionally, mean values and their confidence intervals are marked as before. Finally, Fig. 7 shows the runtime of the sampling methods on the ACOQAP scenario, which is the most computationally demanding. PH-ILHS is generally the fastest method, thanks to its C implementation. Random uniform sampling needs slightly more time, and the optimized variants need orders of magnitude more time.

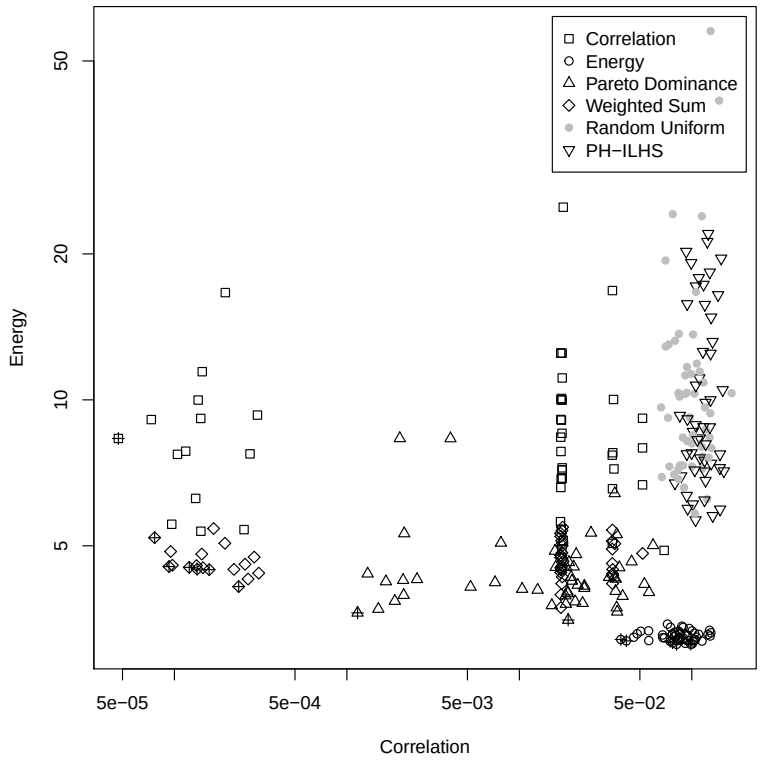
Discussion. Based on the first impressions in Fig. 3, correlation alone apparently should not be used as a criterion, as its designs are usually dominated by the weighted sum approach. However, in the other figures it does seem to also produce a slight performance improvement.



(a) optim



(b) optimx



(c) ACOQAP

Figure 3: Correlation and energy measures of 100 samples (configurations) generated by each sampling method. Non-dominated solutions are marked with "+". Both axes are on a log scale. Lower is better.

It is also not always the same variant that obtains the best performance. The experiments generally show that the optimized sampling is not always significantly better than random uniform sampling, but it is never significantly worse. In Fig. 6, this may partly be explained with the low number of replications together with the high variance of the results. It also depends on the parameter space of a scenario if the performance can be improved. The optimized sampling

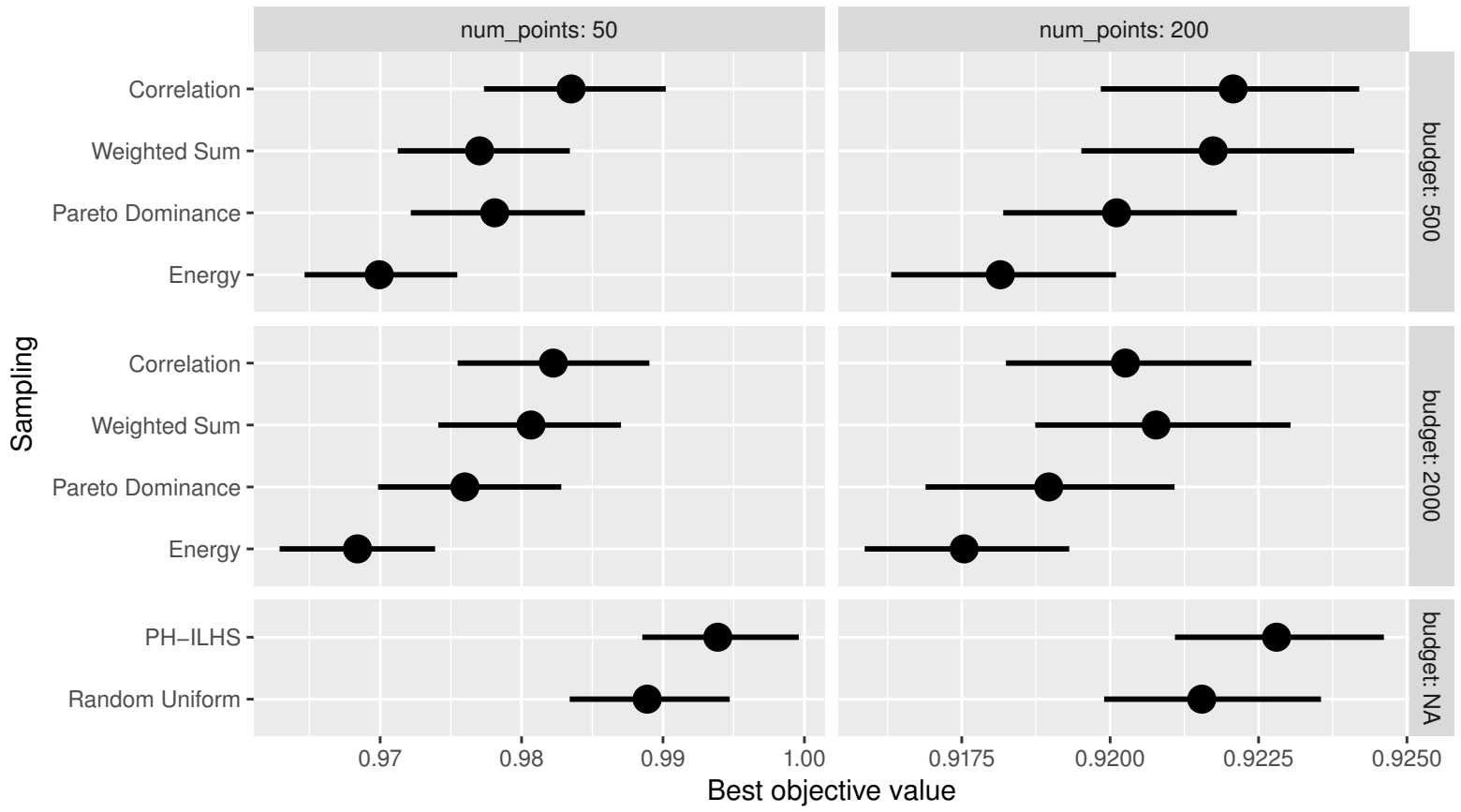


Figure 4: Mean values and bootstrapped 95% confidence intervals over 500 replications for sampling methods on the optim scenario (lower is better).

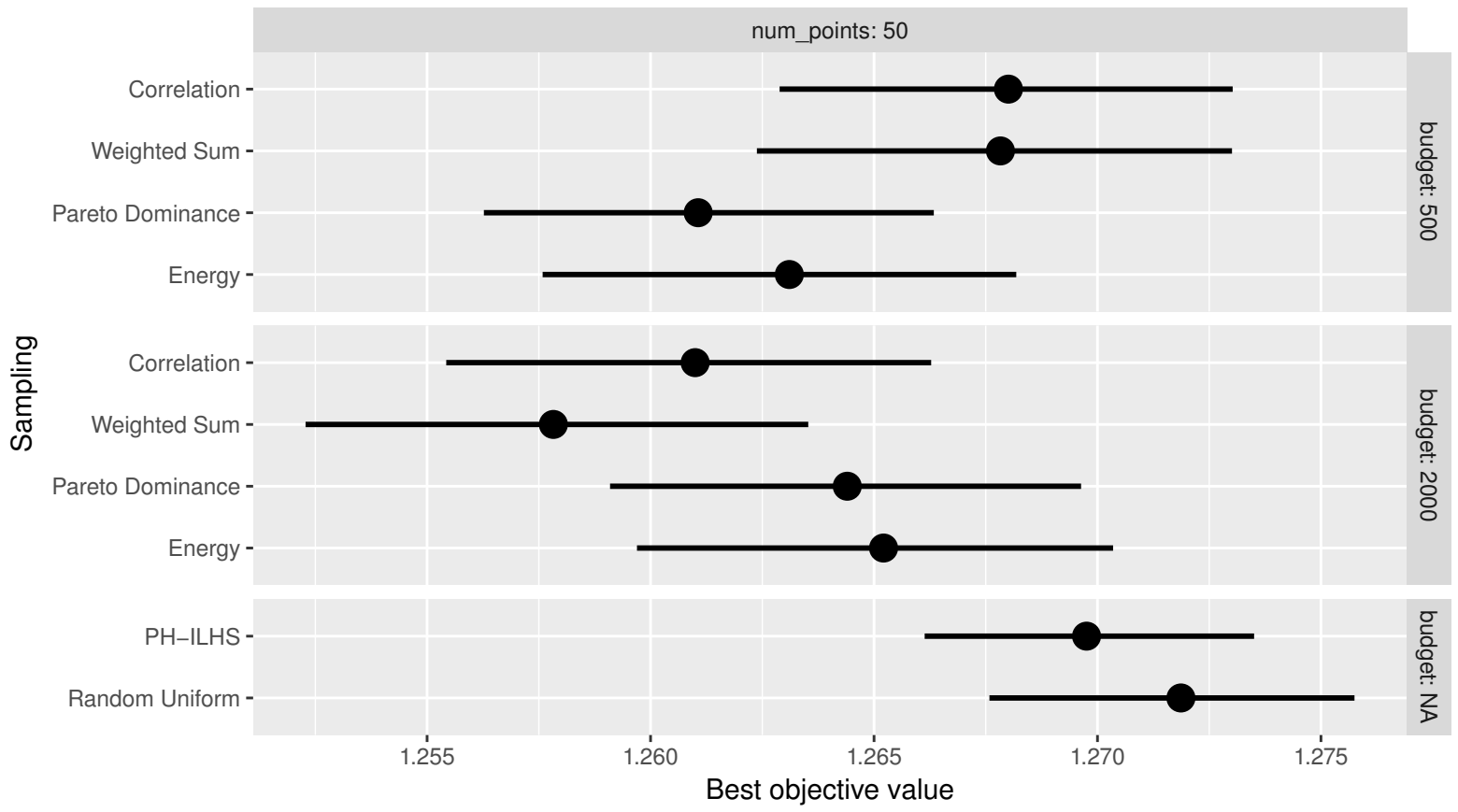


Figure 5: Mean values and bootstrapped 95% confidence intervals over 1000 replications for sampling methods on the optimx scenario (lower is better).

is interesting when the configured algorithms are so expensive that the runtime of the sam-

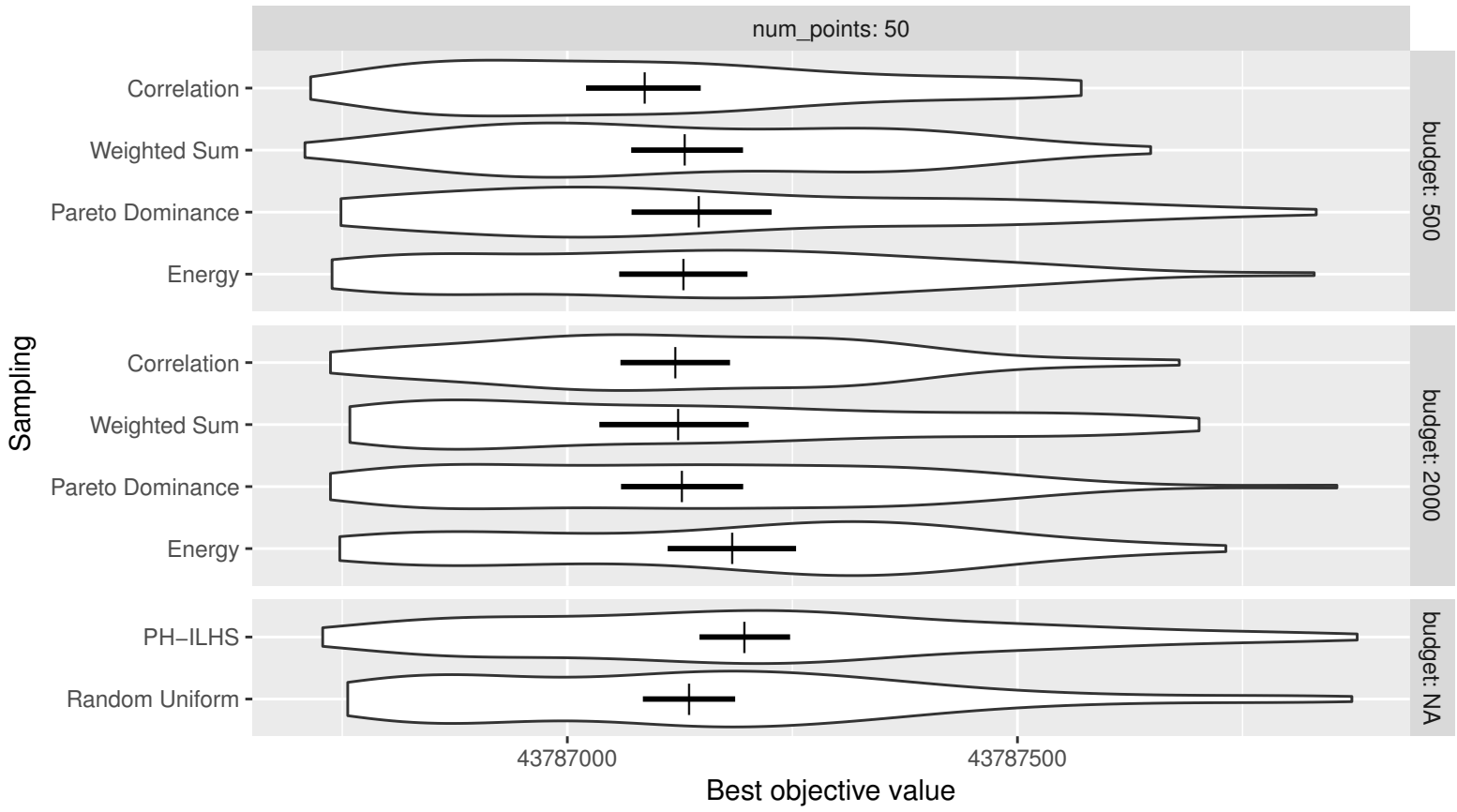


Figure 6: Distributions of the best objective value over 50 replications for sampling methods on the ACOQAP scenario (lower is better). The crosses indicate mean values and bootstrapped 95% confidence intervals.

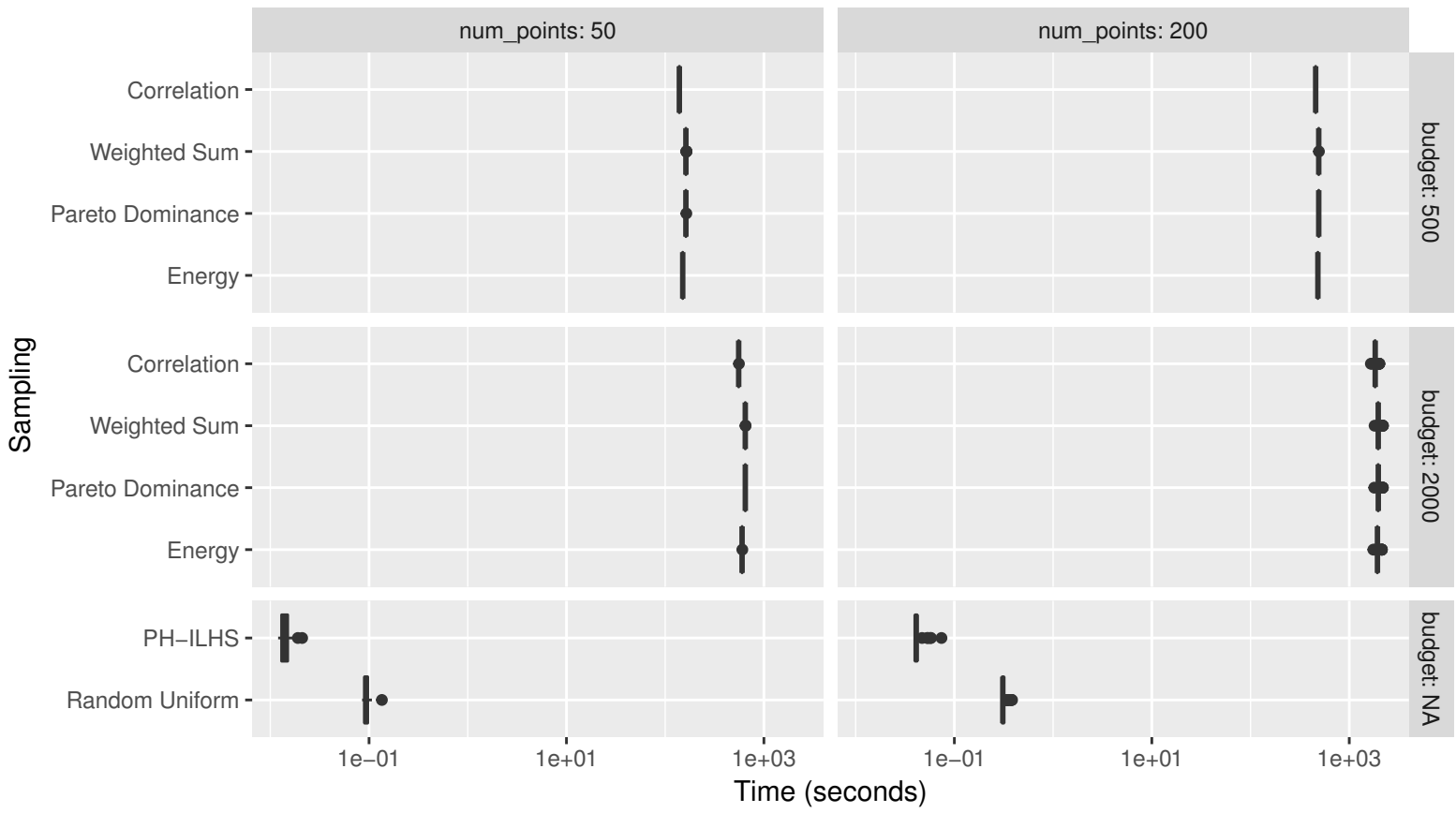


Figure 7: Computation time of the sampling methods (before evaluating the configurations) over 50 replications on the ACOQAP scenario.

pling can be neglected. Then, also the budget assigned to irace will be small and the optimized initialization relatively cheap in comparison. Nevertheless, as shown by the small runtime of PH-ILHS, implementing our proposed optimized sampling in C instead of R may yield a significant runtime reduction and allow their use in the configuration of computationally cheap algorithms. Also note that the runtime of the optimized sampling could still be greatly improved by caching the correlations and distances pertaining to the already fixed subspaces.

8 Conclusion

In this paper we have studied several alternatives for the initialization step of irace, although the results presented here can be used by most automatic configuration methods. In particular, we have compared the default initialization method used in irace, based on random uniform sampling, with several Latin hypercube sampling methods that are able to handle categorical and numerical parameters that may be conditional (nested) on the value of other (branching) parameters. The Latin hypercube designs (LHD) are produced by optimizing generalizations of classical optimality criteria for designs of computational experiments (Morris and Mitchell, 1995) with an evolutionary algorithm. Results show that the optimized sampling variants can produce better configurations in terms of performance averaged over a test set of problem instances than the default uniform sampling. In the worst case, with large parameter spaces and low number of configurations sampled, our results show that the optimized sampling may not be significantly better than random uniform sampling, however, it should never be worse, on average.

Configuration scenarios without nested factors should benefit from the variance reduction of the improved sampling as well, as they are a special case of the more general approach proposed here. When there are no conditional parameters, the methods produce conventional optimized LHD, which are state-of-the-art (Pronzato and Müller, 2012).

The current implementation in R of the proposed optimized sampling methods is much slower than the default random sampling of irace, thus, they should only replace the default initialization method in irace, and possibly other automatic configuration tools, in the case of a high runtime of the configured algorithm for which the additional computational time required by the sampling can be relatively long. Nevertheless, in terms of computational complexity, while the optimized LHDs are expensive, a non-optimized LHD has the same linear complexity as random uniform sampling and thus should be always preferred to it. Moreover, a more efficient implementation, e.g., in C, of the methods proposed here is likely to make any differences in computational time negligible in practice, as shown by the fact that PH-ILHS, implemented in C, is much faster than the random sampling implemented in R.

The best criterion to optimize the LHDs is also an open question that may well depend on the features of the parameter space, the number of points and the budget available for optimization. Our results show that in some scenarios, the energy criterion performs exceptionally well, whereas in other scenarios a weighted sum is more effective. A proper analysis of this question require the careful design of artificial configuration scenarios, which we leave for future research.

A possible direction for future research could be to focus on the sequential part, i.e. after the initial sampling, of algorithm configuration methods. The optimality criteria presented here are independent of LHDs and could also be used to determine infill points for a sequential sampling, if uniformity is sought. A further step would then be to employ a model-based approach for branching and nested designs as algorithm configuration method. The foundation for this approach is already laid, as Hung et al. (2009) have not only defined criteria for the initial branching and nested designs, but also developed a corresponding kernel for Kriging metamodels (also called Gaussian Processes) (Rasmussen and Williams, 2006). Consequently,

this kernel could be applied to do sequential model-based optimization on the nested parameter space. A modified version of irace that includes all the sampling methods evaluated here is publicly available for further analysis.[2]

References

Addis, B., Locatelli, M., and Schoen, F. (2008). Disk packing in a square: A new global optimization approach. *INFORMS Journal on Computing*, 20(4):516–524.

Adenso-Díaz, B. and Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, 54(1):99–114.

Audze, P. and Eglãjs, V. (1977). New approach to the design of multifactor experiments. *Problems of Dynamics and Strengths*, 35:104–107. (in Russian).

Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In Bartz-Beielstein, T., Blesa, M. J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., and Sampels, M., editors, *Hybrid Metaheuristics*, volume 4771 of *LNCS*, pages 108–122. Springer.

Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation: The New Experimentalism*. Springer, Berlin, Germany.

Bartz-Beielstein, T., Flasch, O., Koch, P., and Konen, W. (2010). SPOT: A toolbox for interactive and automatic tuning in the R environment. In *Proceedings 20. Workshop Computational Intelligence*, Karlsruhe. KIT Scientific Publishing.

Beachkofski, B. and Grandhi, R. (2002). Improved distributed hypercube sampling. In *Proceedings of the 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. AIAA paper 2002-1274, American Institute of Aeronautics and Astronautics.

Bezerra, L. C. T., López-Ibáñez, M., and Stützle, T. (2016). Automatic component-wise design of multi-objective evolutionary algorithms. *IEEE Trans. Evol. Comput.*, 20(3):403–417.

Birattari, M. (2009). *Tuning Metaheuristics: A Machine Learning Perspective*, volume 197 of *Studies in Computational Intelligence*. Springer, Berlin/Heidelberg, Germany.

Birattari, M., Balaprakash, P., and Dorigo, M. (2006). The ACO/F-RACE algorithm for combinatorial optimization under uncertainty. In Doerner, K. F., Gendreau, M., Greistorfer, P., Gutjahr, W. J., Hartl, R. F., and Reimann, M., editors, *Metaheuristics – Progress in Complex Systems Optimization*, volume 39 of *Operations Research/Computer Science Interfaces Series*, pages 189–203. Springer, New York, NY.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In Langdon, W. B. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA.

Bischl, B., Lang, M., Bossek, J., Horn, D., Schork, K., Richter, J., and Kerschke, P. (2017). *ParamHelpers : Helpers for Parameters in Black-Box Optimization, Tuning and Machine Learning*. R package version 1.10.

Carnell, R. (2016). *lhs: Latin Hypercube Samples*. R package version 0.14.

Çela, E. (1998). *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

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

Damelin, S. B., Hickernell, F. J., Ragozin, D. L., and Zeng, X. (2010). On energy, discrepancy and group invariant measures on measurable subsets of Euclidean space. *Journal of Fourier Analysis and Applications*, 16(6):813–839.

[2] https://github.com/MLopez-Ibanez/iracelhs

Deb, K. and Sinha, A. (2009). Solving bilevel multi-objective optimization problems using evolutionary algorithms. In Ehrgott, M., Fonseca, C. M., Gandibleux, X., Hao, J.-K., and Sevaux, M., editors, *EMO*, volume 5467 of *LNCS*, pages 110–124. Springer.

Eiben, A. E. and Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Springer.

Hardin, D. P. and Saff, E. B. (2004). Discretizing manifolds via minimum energy points. *Notices of the American Mathematical Society*, 51(10):1186–1194.

Hoos, H. H. (2012). Programming by optimization. *Commun. ACM*, 55(2):70–80.

Hung, Y., Joseph, V. R., and Melkote, S. N. (2009). Design and analysis of computer experiments with branching and nested factors. *Technometrics*, 51(4):354–365.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In Coello Coello, C. A., editor, *Learning and Intelligent Optimization, 5th International Conference, LION 5*, volume 6683 of *LNCS*, pages 507–523. Springer.

Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M. T., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2014). AClib: a benchmark library for algorithm configuration. In Pardalos, P. M., Resende, M. G. C., Vogiatzis, C., and Walteros, J. L., editors, *LION*, volume 8426 of *LNCS*, pages 36–40. Springer.

Hutter, F. and Ramage, S. (2015). *Manual for SMAC*. University of British Columbia. SMAC version 2.10.03.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

López-Ibáñez, M., Stützle, T., and Dorigo, M. (2018). Ant colony optimization: A component-wise overview. In Martí, R., Pardalos, P. M., and Resende, M. G. C., editors, *Handbook of Heuristics*, pages 1–37. Springer International Publishing.

Loshchilov, I. and Glasmachers, T. (2017). Black box optimization competition.

Maron, O. and Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Research*, 11(1–5):193–225.

McKay, M. D., Beckman, R. J., and Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245.

Morris, M. D. and Mitchell, T. J. (1995). Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference*, 43(3):381–402.

Müller, C. L. and Sbalzarini, I. F. (2012). Energy landscapes of atomic clusters as black box optimization benchmarks. *Evolutionary Computation*, 20(4):543–573.

Nash, J. and Varadhan, R. (2011). Unifying optimization algorithms to aid software system users: optimx for R. *Journal of Statistical Software*, 43(9):1–14.

Pronzato, L. and Müller, W. G. (2012). Design of computer experiments: space filling and beyond. *Statistics and Computing*, 22(3):681–701.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Santner, T. J., Williams, B. J., and Notz, W. I. (2003). *The Design and Analysis of Computer Experiments*. Springer.

Shields, M. D. and Zhang, J. (2016). The generalization of latin hypercube sampling. *Reliability Engineering & System Safety*, 148:96–108.

Stützle, T. (2002). ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem.

Yuan, B. and Gallagher, M. (2004). Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In Yao, X. et al., editors, *PPSN*, volume 3242 of *LNCS*, pages 172–181. Springer.