Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

A UNIFIED RACE ALGORITHM FOR OFFLINE PARAMETER TUNING

Tim van Dijk
Martijn Mes
Marco Schutten

Dep. Industrial Engineering and Business Information Systems
University of Twente
Drienerlolaan 5
7522 NB Enschede, THE NETHERLANDS

Joaquim Gromicho

Dep. Algorithmic Innovation
ORTEC Software Development
Houtsingel 5
2719 EA Zoetermeer, THE NETHERLANDS

ABSTRACT

This paper proposes uRace, a unified race algorithm for efficient offline parameter tuning of deterministic algorithms. We build on the similarity between a stochastic simulation environment and offline tuning of deterministic algorithms, where the stochastic element in the latter is the unknown problem instance given to the algorithm. Inspired by techniques from the simulation optimization literature, uRace enforces fair comparisons among parameter configurations by evaluating their performance on the same training instances. It relies on rapid statistical elimination of inferior parameter configurations and an increasingly localized search of the parameter space to quickly identify good parameter settings. We empirically evaluate uRace by applying it to a parameterized algorithmic framework for loading problems at ORTEC, a global provider of software solutions for complex decision-making problems, and obtain competitive results on a set of practical problem instances from one of the world's largest multinationals in consumer packaged goods.

1 INTRODUCTION

In this paper, we focus on efficient offline parameter tuning of deterministic algorithms with parameter spaces that may be enormous and include categorical parameters. The performance of a parameterized algorithm on specific problem instances depends on the quality of its parameter configuration. Moreover, the same parameter settings likely perform differently on distinct problem instances. This suggests the need for a tuning algorithm that quickly identifies good parameter settings for a parameterized algorithm such that its performance on a specific type of instance is optimized. We propose an innovative unified race algorithm (uRace) to satisfy this need.

The development of uRace finds its origin at ORTEC, a global provider of planning and optimization software solutions. One of their software solutions is ORTEC Pallet and Load Building (OPLB), which aims to facilitate efficient pallet and container loading for customers worldwide. The list of customers for OPLB includes many well-known multinationals, each of which use the software to find good loading plans

that optimize their specific objective criterion while satisfying all constraints. Figure 1 shows examples of the resulting load visualizations for two different applications.

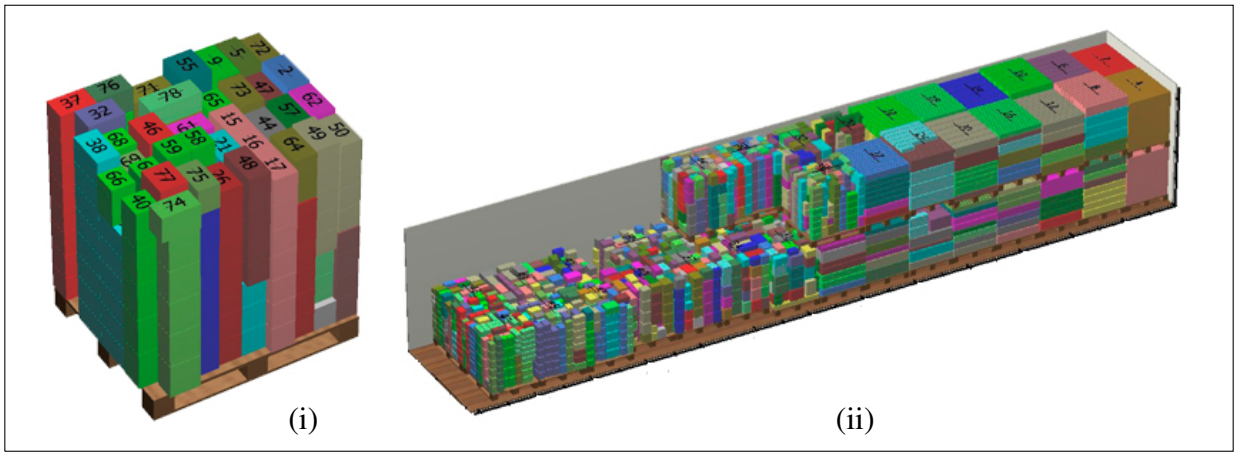

Figure 1: Example of a visualization of (i) a load of boxes on a pallet and (ii) a load of (pick)pallets in a container.

OPLB contains a heavily parameterized algorithmic framework that consists of two parts: (i) a greedy construction phase, which incrementally and deterministically constructs a solution, and (ii) a local search phase, which iteratively destructs several of the last executed item placements, selects an alternative item placement at random, and subsequently reconstructs using the same greedy construction. Particularly the greedy construction phase can be customized with much degree of freedom by tuning its parameters. The parameters are mainly categorical sorting parameters that sort the options for item placement on various preferences, e.g., sorting the remaining items on "length" in "ascending" order, or the empty subspaces in the container on "volume" in "descending" order. In total, there are approximately 6×10^{39} possible parameter configurations. The item placement preference lists, and therefore the parameter settings in the greedy construction phase, govern the incremental construction. As a result, these parameter settings determine the quality of the initial solution. Finally, the local search phase aims to improve on the initial solution, introducing a stochastic element to the solution process. However, as a result of the aforementioned neighborhood structure in the local search algorithm, the greedy construction is reused in every iteration, which means that the quality of the initial solution translates well to the quality of the final solution. For this reason, we exclude the local search from our parameter tuning efforts, as this radically speeds up our measurements. Moreover, the deterministic nature of the greedy construction precludes any stochastic noise from our observations, which greatly benefits the learning process of our tuning algorithm.

We distinguish two different approaches through which we may seek optimal parameter settings for the greedy construction phase of OPLB. First, we can opt for problem instance tuning, in which case we tune the parameters for the specific problem instance the customer currently faces, the online problem. Second, we can opt for problem family tuning, also denoted by offline tuning, in which case we tune the parameters for a problem the customer *typically* faces, simulated with a set of representative training instances. In essence, problem instance tuning is an extreme of problem family tuning where the training set coincides with the online problem, hence produces better results in case the required computation time is not an issue. However, for problem instance tuning in our practical setting, the computationally expensive tuning process occurs every time a customer wants to load a container, which imposes severe limitations on the tuning time. Indeed, any self-learning tuning process would be computationally problematic for problem instance tuning, and our enormous search space makes complete enumeration computationally prohibitive. Therefore, we prefer to tune the parameters before the customer wants to load a container, whenever computational resources are amply available, on a set of training instances (e.g., recent historic

instances from that customer). The resulting parameter configuration can subsequently be applied to any occurring problem instance at the customer site without any further tuning.

Evidently, problem family tuning is only viable when a configuration that performs well on the training set also performs well on similar, yet unseen problems. To verify this, we uniformly draw a sample of parameter configurations from our parameter space and evaluate their performance on a set of similar loading problems from one of ORTEC's customers. Figure 2 illustrates two effects that suggest that such a generalization of quality holds for parameter configurations in OPLB. The graph on the left shows the relation between the average performance on a training set from the customer and the performance on one, randomly selected, other instance of this customer. The strong positive correlation implies that, provided we have a representative training set, the average performance on the training set is a good indication of the performance on the online problem. Moreover, the graph on the right shows the relation between the average performance on the complete set of instances and the coefficient of variation (CV) of the performance across the set, where a low CV indicates a high robustness. When looking at the correlation between the average and variance in volume utility, we observe a strong correlation (Pearson coefficient of -0.87, p-value = 0.000), which indicates that while optimizing the average performance of a parameter configuration on the training set, we simultaneously optimize its robustness, and therefore the extent to which its quality generalizes to the online problem. These are crucial results for our research, as they advocate the use of problem family tuning.

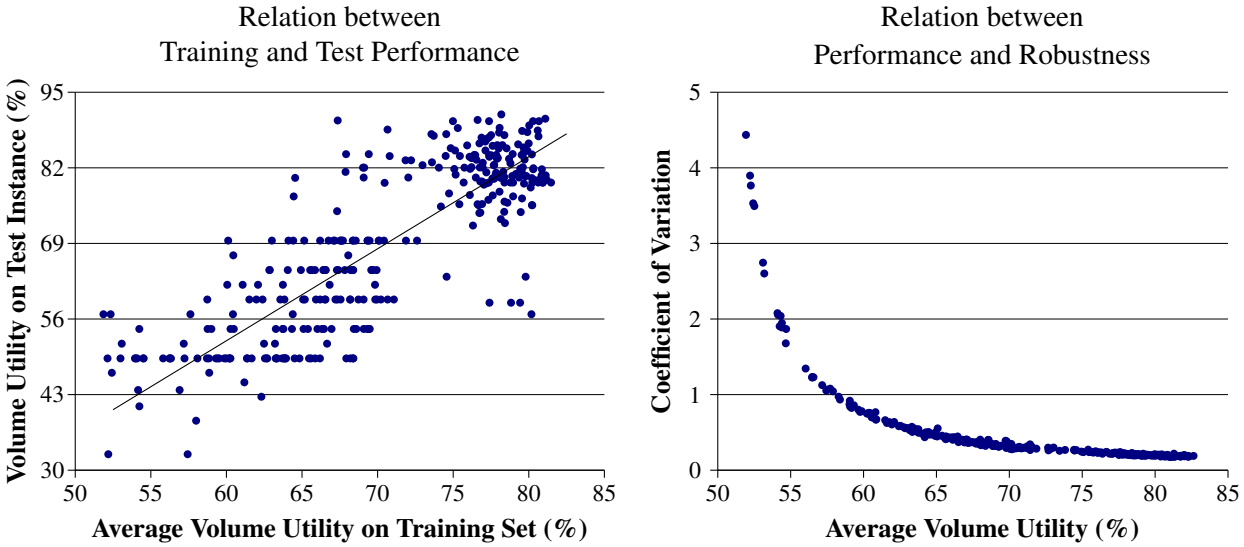

Figure 2: Illustration of the viability of problem family tuning of the parameters in OPLB. Each dot represents one randomly sampled parameter configuration. The performance is expressed as the volume utility (%) of the loaded container. The coefficient of variation is expressed as the ratio of the sample standard deviation to the sample mean of the performance distribution on the training set.

The contribution of this paper is the development of uRace, a unified race algorithm for efficient offline parameter tuning of deterministic algorithms. We apply uRace to a parameterized loading algorithm within OPLB. On a set of practical problem instances from a multinational in consumer packaged goods, we obtain competitive results compared to well known algorithms in the literature. Moreover, as a result of the similar stochastic element involved in stochastic simulations as in problem family tuning, uRace can also be used for simulation optimization purposes.

The remainder of this paper is structured as follows. In Section 2, we present some notable work that aids in the development of a problem family tuning method. In Section 3, we present uRace. In Section 4,

we provide the results of our computational experiments. Finally, in Section 5, we provide our conclusions regarding theoretical and practical relevance, and offer suggestions for further research.

2 PROBLEM SETTING

In the field of parameter tuning, we are interested in "a staggeringly difficult yet beautifully concise problem" (Lizotte 2008):

$$\arg\max_{\theta\in\Theta} f(\theta) \tag{1}$$

When solving Equation 1, the aim is to quickly identify, from a set of alternatives Θ, the one alternative θ^* that gives the highest (expected) return. This problem appears in many forms in the literature. In the field of simulation optimization, the function $f(\theta) = \mathbb{E}[F(\theta,\omega)]$ follows from a simulation run with sample path ω and sample performance $F(\theta,\omega)$ (Fu, Glover, and April 2005). For example, $F(\theta,\omega)$ gives the simulated profit of a shop, considering stochastic customer choice and demand, where the corresponding set Θ would be the possible choices in product variants to stock (Hong and Nelson 2009).

The literature covers many relevant techniques to solve Equation 1. In sequential measurement techniques, for example, the search for the best alternative is sequentially guided towards the next measurement depending on previous observations. An example of a relatively simple sequential measurement technique is an upper confidence bound (UCB) policy, in which, consistently, the alternative with the highest UCB in terms of return is measured, resulting in continuously shrinking performance distributions (Chang, Fu, Hu, and Marcus 2007). Typically, such simple heuristics require each alternative to be measured at least once, which becomes computationally prohibitive for larger parameter spaces.

More sophisticated techniques are able to tackle larger search spaces. In such approaches, a priori knowledge on interrelations in the parameter space is used to design correlated beliefs about the effect of the measured performance of some alternative on the expected performance of some other alternative, such that estimates can be given even for alternatives that have not been measured before. Such extrapolation of the observations on one alternative to the beliefs of other alternatives is called generalization. Most sequential measurement policies that are applicable to large search spaces rely heavily on generalization techniques. For example, the knowledge gradient policy (Frazier, Powell, and Dayanik 2008) myopically maximizes the expected increment in the value of information by continuously directing the walk towards the point that will reveal the most information on other unseen points. While the updated knowledge gradient policy by the same authors assumes a priori knowledge on correlations in the belief structure of Θ, the hierarchical knowledge gradient (Mes, Powell, and Frazier 2011) requires a known aggregation structure of the parameter space, which is generally easier to obtain. In the related field of Bayesian global optimization of continuous functions, one would place a prior belief on the function f and rely on the laws of probability to update these beliefs to posterior distributions based on accumulated observed data. The response-surface methodology (RSM) (Law 2007), e.g., takes a Bayesian approach to fit a second order response surface to a series of experiments, conducted using a simple experimental design, and subsequently finds the optimal point on this surface. Most sequential measurement policies are designed for stochastic experiments with noisy measurements, but some can be applied to deterministic cases, and others are designed specifically for noise-free measurements. The efficient global optimization (EGO) method (Jones, Schonlau, and Welch 1998), e.g., assumes deterministic measurements and attempts to fit a smooth, differentiable function through previous measurements while continuously selecting points based on expected improvement of the objective function.

Our specific tuning problem possesses a few important characteristics that determine the suitability of tuning techniques. First, our parameter space, hence our search space of alternatives Θ, is enormous, which eliminates the use of any tuning technique that requires a full factorial design or needs to rank all possible decisions before making the decision. Second, our parameter space is of categorical nature, such that $\theta \in \Theta$ is a vector of categorical attributes, which complicates the use of generalization techniques.

Techniques such as RSM and EGO assume a spatial relationship between the parameter space on the one hand, and a supposedly smooth performance landscape on the other hand, which they attempt to exploit to guide the tuning process. In many practical applications with numerical variables, this assumption is reasonable. However, for categorical variables, which only have a symbolic meaning and do not have a natural way to be encoded, the assumption of a regular or smooth performance landscape is not likely to hold. Indeed, it is not trivial to quantify the distance between categorical parameter values (e.g., sorting items by volume or by priority), let alone to assume that this unquantifiable distance corresponds with similar distances in the performance landscape. Third, recall that our problem family tuning process occurs on a set of training instances, which we denote by $\mathscr{X}$, prior to knowing the online problem instance at the customer site. That is, we effectively simulate reality (the unseen online problem instance) with similar, representative situations (a training set with other problem instances in the same problem family), hence introducing a stochastic element to our tuning process. We obtain information about a configuration θ by evaluating its quality on a training instance x_i in $\mathscr{X}$. Finally, $f : \Theta \to \mathbb{R}$ is an algorithm that translates the input vector into a single value, indicating the solution quality of the resulting loading plan on a particular problem instance, where every individual measurement $f(\theta|x_i)$ is deterministic.

Our objective is to identify the parameter configuration θ that maximizes the expected value of f on $\mathscr{X}$, which, as Figure 2 suggests, coincides with our ultimate objective to optimize the loading solution on the online problem instance at the customer site. We obtain the offline algorithm configuration problem (Birattari 2009):

$$\arg\max_{\theta\in\Theta} \mathbb{E}_{\mathscr{X}}\left[f(\theta)\right] = \arg\max_{\theta\in\Theta} \frac{1}{|\mathscr{X}|}\sum_{i=1}^{|\mathscr{X}|} f(\theta|x_i). \tag{2}$$

Tuning methods specifically designed to efficiently solve Equation 2 are race algorithms. In a race algorithm, parameter configurations are evaluated in batches, on a sequence of problem instances, with the aim to eliminate candidate configurations based on statistical inferiority, such that an increasing number of configurations "drop out of the race". This speeds up the tuning process and allows more productive allocation of computation time to the most promising configurations.

One of the first successful race algorithms, named F-Race, is presented by Birattari, Stützle, Paquette, and Varrentrapp (2002). F-Race starts with a set Θ_r of candidate configurations from which it tries to exclude inferior members as soon as sufficient statistical evidence is collected until either one configuration remains or some predefined computation budget is exhausted. An obvious drawback of F-Race is that the winning configuration is always a member of the initial set Θ_r. Therefore, Θ_r need be obtained through a full factorial design, or at least be a large subset of Θ, which becomes computationally prohibitive for large parameter spaces. To tackle this issue, Balaprakash, Birattari, and Stützle (2007) propose an extension of F-Race, iterative F-Race (I/F-Race), which consists of multiple, iterated applications of F-Race. In the first iteration, a multi-variate uniform distribution is used to randomly sample the initial set of candidate configurations Θ_r. In each subsequent iteration, a new set Θ_r is constructed based on the most promising configurations from the previous iteration. Through the iterated nature of a race algorithm, the requirement of a full factorial design is removed. Rather than requiring information about all members of Θ, a subset $\Theta_r \in \Theta$ is used that gradually changes in composition. A suitable technique to reconstruct the set Θ_r based on previous observations, in light of the categorical parameters in OPLB, is the use of biased sampling distributions (Birattari, Stützle, Paquette, and Varrentrapp 2002). This technique does not require the existence of a smooth and predictable performance landscape nor a full factorial design, and randomly samples parameter settings for new measurements, while continuously adapting the sampling distributions based on previous measurements. For categorical parameters, new values are sampled from discrete sampling distributions with a bias towards the parameter settings from known configurations that have shown good performance, the remaining "racers" from the previous iteration. This bias can be increased throughout the tuning process to facilitate an increasingly localized search.

We apply our tuning algorithm to loading problems. All loading problems share an identical structure, which can be summarized as follows. We have two sets of elements, a set of large objects (e.g., pallets or containers) and a set of small items that need to be placed in or on these large objects. The small items are defined in one, two, or three geometric dimensions (and may possess other characteristics such as weight and value). The loading problem is solved by selecting small items, grouping them into one or more subsets, and assigning each of them to one of the large objects such that a given objective function is optimized and the following geometric restrictions are met: (a) all small items of the subset lie entirely within the large object, and (b) the small items do not overlap. Other constraints may apply on top of these, e.g., fixed orientations and support, axle weight, and stackability constraints. The literature covers a wide scope of solution methods to tackle loading problems. For example, George and Robinson (1980) propose a wall building approach in which the large object is filled with vertical cuboid layers, "walls", along the width of the container. Bischoff, Janetz, and Ratcliff (1995) propose a horizontal layer approach, in which the container is filled with horizontal cuboid layers that each cover the largest possible fraction of the surface underneath. For a recent state-of-the-art review on loading problems, we refer to Bortfeldt and Wäscher (2013).

3 DESIGN OF A TUNING ALGORITHM

Our tuning algorithm is inspired by I/F-Race (see Section 2). However, we propose an important structural change to utilize obtained information more efficiently. As said, I/F-Race enforces fair comparisons, by letting each iteration be an autonomous F-Race. Information from previous iterations is used to generate a set of configurations to commence the next F-Race, but the exact results of individual measurements are discarded. To enable the use of this valuable information, while still ensuring fair comparisons, we prefer to first evaluate every newly generated configuration on the same problem instances as those previously solved by its parent configuration. Since this removes the stochastic element inherent to problem instance selection, this gives us more information about the new configurations than measurements on new problem instances would.

The result of this structural change is that we create one unified race algorithm, rather than a sequence of autonomous F-Race algorithms. We consider each iteration to be a batch measurement of all candidate racers on a single problem instance, and we generate new configurations immediately whenever statistically inferior configurations are eliminated in our sample. This allows for more natural allocation of computation time to either exploration (i.e., exploring new points in the solution space) or exploitation (i.e., acquiring more reliable information on known solution points). Indeed, at the start of the tuning process the configurations are likely relatively diverse in their performance, so we expect many inferior configurations, naturally allocating computation time to exploration. As the tuning process proceeds, and we approach the best configuration θ^*, we expect the candidates to be closer to one another in terms of performance. Therefore, by only exploring new options in case of statistical inferiority, computation time allocation naturally tends towards exploitation.

The basic structure of uRace is presented in Algorithm 1. Each iteration j starts with a set of "racing" configurations Θ_r, where we let the designated size C_τ of Θ_r depend on the fraction τ of computation budget spent and the size of our parameter space (see Section 3.3). In each iteration, all configurations in Θ_r are tested for statistical inferiority based on previous observations. If no statistical inferiority is detected, all configurations are measured on a new problem instance x_j, which is drawn randomly from the training set $\mathscr{X}$ without replacement (as long as $|\mathscr{X}| > 0$). Any inferior configuration is eliminated, and immediately replaced by a newly generated child configuration θ^{child}. This newly generated configuration is first measured on the same stream of problem instances as its parent, as long as it is not found to be statistically inferior. Once we have generated sufficient non-inferior child configurations such that the set of racing configurations Θ_r is back to its designated size, we continue with a new iteration. When all problem instances have been used ($|\mathscr{X}| = 0$) and Θ_r has its designated size, we remove the worst configuration from Θ_r (even when this configuration is not statistically inferior) and continue exploration.

Algorithm 1 Basic Structure of uRace

input
$\mathscr{X}$: training set
$\mathscr{Y}$: parameter space
$f(\theta, x_i) : \mathscr{Y} \to \mathbb{R}$: algorithm

tunable parameters
B : tuning budget
α : level of significance
β : degree of bias

initialization
$\Theta_r \sim \text{LHS}(\mathscr{Y}, C_0)$
$j \leftarrow 1$

output
θ^* : best configuration in Θ on $\mathscr{X}$

while tuning budget left **do**
  $\Theta_r \leftarrow \Theta_r \setminus \Theta_{\text{inferior},\alpha}$
  while tuning budget left **and** $|\Theta_r| < C_\tau$ **do**
    $\theta^{\text{child}} \sim \text{regenerate}(\mathscr{Y}, \Theta_r, \beta)$
    for every instance x_i solved by parent **do**
      evaluate $f\left(\theta^{\text{child}} | x_i\right)$
    end for
    if θ^{child} non-inferior **then** $\Theta_r \leftarrow \Theta_r \cup \theta^{\text{child}}$ **end if**
  end while
  while $|\Theta_r| > C_\tau$ **do** eliminate worst $\theta \in \Theta_r$ **end while**
  if $j < |\mathscr{X}|$ **then**
    for all $\theta \in \Theta_r$ **do** evaluate $f(\theta | x_j)$ **end for**
    $j \leftarrow j + 1$
  else
    eliminate worst $\theta \in \Theta_r$
  end if
end while

The tuning algorithm requires as input a set of training instances $\mathscr{X}$, the parameter space $\mathscr{Y}$, and the underlying algorithmic framework f. The user is free to set the tuning budget B, enabling easy regulation of the computation time spent on the tuning procedure. Moreover, we have two tunable parameters, α and β, that can be used to affect the exploration-exploitation tradeoff and the locality of the search. We return to these parameters in Section 3.2 and Section 3.3, respectively.

The remainder of this section describes uRace in detail, covering the three main components of an iterated race algorithm. First, Section 3.1 discusses the construction of the initial set Θ_r. Second, Section 3.2 discusses the statistical test that detects inferior members within Θ_r. Third, Section 3.3 discusses the method to construct subsequent (sets of) candidate configurations based on the remaining configurations from the previous iteration.

3.1 Initial Generation

To compose our initial set of parameter configurations, we need to select a set $\Theta_r \subset \Theta$. The initial set of configurations forms the roots of all successively generated parameter configurations, so it is desirable that it includes points from all regions in the parameter space. Latin Hypercube Sampling (LHS), first proposed by McKay, Conover, and Beckman (1979), is a method to construct such a representative subset by drawing a sample that uniformly covers the entire multi-dimensional parameter space. LHS selects n values from k parameters by dividing the range of each parameter into n non-overlapping, equally spaced intervals, which is problematic for categorical parameters. To adapt the traditional LHS to be applied to categorical parameters, we propose a straightforward mapping structure. That is, instead of uniformly drawing a sample of size n directly from the k categorical parameter spaces, we use a single, continuous, one-dimensional space $[0, n]$. From this space, we randomly draw n values, one from each integer interval. Subsequently, we map the drawn values to each of the k categorical parameter spaces, and randomly combine the resulting categorical values with the values of other parameters to obtain n k-dimensional input vectors, our initial set of configurations. Using this categorical adaptation of LHS, we enforce an even spread of our initial sample over our entire multi-dimensional parameter space.

3.2 Statistical Inferiority Test

As soon as sufficient statistical evidence is collected that a configuration is inferior to at least one other configuration in Θ_r, we eliminate the inferior configuration. As said, we enact a batch measurement policy, where we let all racers solve the same sequence of training instances. An analysis of our problem space reveals that we cannot rely on the usual assumptions of normality and equal variances. Therefore, we employ a non-parametric alternative for the pairwise Student t-test: the paired signed-rank test, proposed by Wilcoxon (1945). We perform one-tailed tests, and compare, after every batch measurement, the best configuration (highest mean performance) with all others.

The level of significance α determines the severity of the test. The higher we set α, the less evidence we require to exclude configurations. This speeds up the procedure, but also increases the probability of falsely rejecting the null hypothesis or, in other words, eliminating non-inferior configurations. An analysis of our problem space suggest that the robustness of the configurations is of such magnitude that even for relatively high levels of significance (i.e., $0.1 \leq \alpha \leq 0.3$), unlawful rejection of the null hypothesis hardly occurs.

Moreover, as a result of the structure of uRace, the value of α also has an important effect on the aforementioned exploration-exploitation tradeoff. As said, the structure of uRace enforces that the tradeoff naturally tends towards exploitation as the tuning process proceeds, but the value of α determines the severity of this transition. For example, in the limit that $\alpha \to 0$, we can only obtain sufficient statistical evidence by determining the configuration's true mean performance. That means that configurations are never eliminated, new configurations are never evolved, our algorithm converges to exhaustive evaluation, and all computation time is undesirably spent on exploitation of the initial sample. On the other hand, as $\alpha \to 1$, our one-tailed tests ensure that all configurations (except the very best one) are continuously eliminated. That means that all computation time is spent on exploration, and we undesirably end up making an implementation decision based on measurement on a single problem instance. Initial tests show that a good exploration-exploitation tradeoff for our parameter space can be achieved using $\alpha = 0.20$.

3.3 Regeneration

Every time a statistically inferior configuration is eliminated, a newly generated child configuration, which resembles some known, well-performing parent configuration, takes its place. This regeneration procedure consists of two parts.

First, we sample, from the set of remaining configurations Θ_r, one parent configuration θ^{parent}. We start with a uniform distribution over all remaining configurations (following our explorative search), and assign increasing probability to the very best configurations (increasingly biasing our search towards the most promising areas in our parameter space) as the tuning process proceeds.

Second, from the parent, we evolve one child configuration θ^{child}, for which we use the biased sampling distribution method (see Section 2). Judging from the high degree of synergy in our parameter space, we should not use a single distribution to sample all child configurations. Such a method is blind for parameter interactions, which prove to be abundant in our parameter space. Instead, we propose to assign individual sampling distributions to every parent configuration, which we bias based on the settings of that configuration and its ancestors only, such that every configuration carries a sampling distribution over $\mathscr{Y}$, and we sample a child configuration using its selected parent's distribution. We start with a k-variate uniform distribution over each parameter space, and for each configuration we evolve a discrete k-variate sampling distribution biased towards the settings taken by itself and its ancestors. The strength of the bias determines how locally we search around known configurations. We bias the sampling distributions such that a child configuration is most likely to inherit half of the parameters from its parent at the start of the tuning process (a relatively explorative search), and all but one of the parameters towards the end of the tuning process (i.e., increasingly localizing the search around the selected parent configuration). In between, the bias increases with τ^{β}, where τ is the fraction of tuning time spent and $\beta > 0$ is the degree of bias we

impose. Note that, as the tuning process proceeds, τ^β increases from 0 to 1. The value for β determines the shape of this gradual process. Linear, exponential, and logarithmic functions are acquired by letting $\beta = 1$, $\beta > 1$, and $0 < \beta < 1$ respectively. Initial tests show that a good locality of search in our parameter space can be achieved with an S-shaped function acquired by using the function $\beta(\tau) = 4(1-\tau)^2$.

We let the designated size C_τ of Θ_r depend on the size of our parameter space and the remaining tuning time. We set $C_0 = k$ at the start of uRace, where k is the number of dimensions in our parameter space $\mathscr{Y}$, and let it gradually decrease to 2 as the algorithm proceeds, using $C_\tau = k - \tau^{\beta(\tau)}(k-2)$. By letting the size of Θ_r decrease, we not only increasingly localize our search around the select parent configuration, but also increasingly bias our search towards the most promising parent configurations, as newly generated child configurations evolve from a smaller number of the best remaining configurations. As mentioned before, if at some point during the tuning process, the current size of Θ_r exceeds its designated size while no inferior racers are detected, the worst configurations are eliminated.

To summarize this section, Table 1 provides an overview of the settings we use for uRace.

Table 1: The components of uRace, the unified race algorithm we propose to develop.

Component	Method
Initial Sampling	Latin Hypercube Sampling
Number of Parallel Configurations	$C_\tau = k - \tau^{\beta(\tau)}(k-2)$
Measurement Policy	Batch Measurements
Statistical Inferiority Test	Wilcoxon Paired Signed-Rank Test with $\alpha = 0.2$
Parent Selection	Rank-Based Sampling with Increasing Bias
Child Generation	Biased Sampling Distributions for Individual Configurations with Increasing Bias
Degree of Bias	$\beta(\tau) = 4(1-\tau)^2$

4 RESULTS

In this section, we provide the results from our empirical evaluation of the performance of uRace that we obtained by letting it tune the parameters of OPLB and subsequently solving a set of loading problems.

The loading problems we evaluate with OPLB are practical problem instances from one of the world's largest multinationals in consumer packaged goods. They are input minimization problems, which means that the full set of small items needs to be accommodated whilst minimizing the selection of large objects utilized. The small items are first loaded onto pick-pallets (i.e., pallets with multiple individual items in any composition) or stock-pallets (i.e., pallets with a fixed load), and the pallets are subsequently loaded into containers. The primary objective is to minimize the number of containers that are required to load all items. Secondary objectives are lower pallet usage, higher stability (i.e., the ratio of the top surface versus the bottom surface of the pick-pallets), and shorter running time.

We compare OPLB with its parameters tuned with uRace with the two well-known construction algorithms from the loading literature that are described in Section 2. Essentially, these algorithms can be seen as specific instances of OPLB, where the parameters are tuned manually by the authors of the respective papers. Hence, the main difference with our parameter configuration is that we tuned the parameters automatically with uRace. We tuned the parameters on a training set of similar, yet distinct loading problems from the same multinational with $|\mathscr{X}| = 20$, $k = 22$ (and therefore $\Theta_r = 22$ at the start of uRace), and $B = 24$ hours.

Table 2 shows the performances of OPLB with the three alternative parameter configurations. The configuration found with uRace outperforms the other two configurations on all then individual test instances, saving $5\% - 15\%$ on container and pallet usage.

Table 2: Performance of OPLB with its parameters tuned with uRace, compared with known construction heuristics, on a set of ten practical loading problem instances from a well-known multinational. The average relative performance gains of uRace are given with a 95% confidence interval.

	Containers		Pallets	
uRace	1.7	-	44.2	-
George and Robinson (1980)	2.0	$-13.3\% \pm 11.4\%$	47.4	$-6.9\% \pm 0.9\%$
Bischoff, Janetz, and Ratcliff (1995)	2.0	$-13.3\% \pm 11.4\%$	47.2	$-6.2\% \pm 1.2\%$

5 DISCUSSION & CONCLUSION

In this paper, we proposed uRace, an innovative tuning algorithm to solve the offline algorithm configuration problem. The unified nature of uRace ensures more efficient usage of previously obtained information, and allows important characteristics of the tuning process, such as the exploration-exploitation tradeoff and the locality of the search to be easily adjustable, therefore making uRace readily applicable to a variety of parameter spaces. Moreover, the applicability of uRace extends to other types of tuning. Table 3 shows the suitability of uRace for four types of tuning on a Likert scale. For example, problem instance tuning of stochastic algorithms is in essence a very similar type of tuning. In uRace, fair comparisons between configurations are enforced by simply applying the configurations to the same problem instances. For problem instance tuning of stochastic algorithms, we may similarly enforce fair comparisons between configurations through the use of common random numbers in the stochastic algorithm by imposing identical, synchronous, random number streams (Law 2007).

Table 3: Suitability of uRace for different types of tuning on a Likert scale.

	Type of Parameterized Algorithm	
	Deterministic	Stochastic
Problem Family Tuning	++	+-
Problem Instance Tuning	–	+

For problem family tuning of stochastic algorithms, there are two simultaneous stochastic processes that influence the tuning. Dependent on the application, we may keep one of the two processes fixed (reducing the type of tuning to either problem instance tuning of stochastic algorithms or problem family tuning of deterministic algorithms) or vary both stochastic processes. In the latter case, uRace should be extended with repeated application of the same parameter configurations to the same problem instance.

Besides algorithms, other parameterized stochastic processes can be tuned similarly with uRace. This makes uRace also suitable for simulation optimization purposes. By viewing one additional time unit of simulation running time as a new measurement, and using fixed period batch runs (and therefore fixed length random number streams), such simulation environments can potentially be tuned very efficiently with uRace, eliminating and generating parameter configurations mid-run, while imposing the same random number streams for newly generated experimental settings.

As far as the practical relevance for ORTEC and its customers goes, uRace can be used to tune the parameters of OPLB, thereby automatically customizing the algorithmic framework for each specific customer. To this end, we require a set of training instances, typically a representative selection of the customer's historic instances. In addition, uRace may be used to periodically update the parameter configuration at the customer site, based on evaluation of more recent, more representative historic problem instances. In other words, the parameter configuration, and therefore the loading algorithm would automatically adjust to the most recent developments at the customer site.

For future research, it would be interesting to compare the performance of OPLB in combination with uRace with the state-of-the-art loading algorithms discussed by Bortfeldt and Wäscher (2013). To this end, the popularly used set of benchmark instances proposed by Bischoff and Ratcliff (1995) can be used, on which the performance of many existing loading algorithms are reported in the literature. Moreover, a

more extensive sensitivity analysis on the input parameters of uRace, α and β, for example by letting α vary with the remaining tuning time, may reveal their exact behavior and optimal settings.

Finally, we mention two possible extensions of uRace that are specific to problem family tuning of algorithms. First, an extension of the offline algorithm configuration problem of Equation 2, such that we seek for multiple, complementary parameter configurations on the training set, which are subsequently all evaluated on the online problem. Through such an approach, we may be able to improve the quality as well as the robustness of the solutions on the online problem. Second, an extension of the offline tuning effort, from which we take a small selection of well-performing configurations, with an additional online tuning effort, where we allow the parameter configuration (i.e., the construction heuristic) to change during the solution process, and seek for the optimal synergy between the construction heuristics.

ACKNOWLEDGMENTS

Our gratitude goes out to ORTEC for allowing us to conduct an empirical evaluation of the performance of uRace through application to one of their customer's loading problem instances.

REFERENCES

Balaprakash, P., M. Birattari, and T. Stützle. 2007. "Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement". *Lecture Notes in Computer Science* 4771:108–122.

Birattari, M. 2009. *Tuning metaheuristics: A machine learning perspective*. 2nd ed. Berlin: Springer.

Birattari, M., T. Stützle, L. Paquette, and K. Varrentrapp. 2002. "A Racing Algorithm for Configuring Metaheuristics". In *Proceedings of the Genetic and Evolutionary Computation Conference*, 11–18.

Bischoff, E., F. Janetz, and M. Ratcliff. 1995. "Loading Pallets with Non-identical items". *European Journal of Operations Research* 84:681–692.

Bischoff, E., and M. Ratcliff. 1995. "Issues in the Development of Approached to Container Loading". *Omega* 23:377–390.

Bortfeldt, A., and G. Wäscher. 2013. "Constraints in Container Loading - A State-of-the-art Review". *European Journal of Operations Research* 229:1–20.

Chang, H., M. Fu, J. Hu, and S. Marcus. 2007. *Simulation-based Algorithms for Markov Decision Process*. Berlin: Springer.

Frazier, P., W. Powell, and S. Dayanik. 2008. "A knowledge-gradient Policy for Sequential Information Collection". *SIAM Journal on Control and Optimization* 47:2410–2439.

Fu, M. C., F. Glover, and J. April. 2005. "Simulation Optimization: A Review, New Developments, and Applications". In *Proceedings of the 2005 Winter Simulation Conference*, 83–95.

George, J., and D. Robinson. 1980. "A Heuristic for Packing Boxes into a Container". *Computers & Operations Research* 7:147–156.

Hong, L., and B. Nelson. 2009. "A Brief Introduction to Optimization via Simulation". In *Proceedings of the 2009 Winter Simulation Conference*, 76–85.

Jones, D., M. Schonlau, and W. Welch. 1998. "Efficient Global Optimization of Expensive Black-Box Functions". *Journal of Global Optimization* 13:455–492.

Law, A. M. 2007. *Simulation Modeling & Analysis*. 4th ed. New York: McGraw-Hill, Inc.

Lizotte, D. 2008. *Practical Bayesian Optimization*. Ph.D. thesis, University of Alberta, Edmonton, Canada.

McKay, M., W. Conover, and R. Beckman. 1979. "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code". *Technometrics* 221:239–245.

Mes, M., W. Powell, and P. Frazier. 2011. "Hierarchical Knowledge Gradient for Sequential Sampling". *Journal of Machine Learning Research* 12:2931–2974.

Wilcoxon, F. 1945. "Individual Comparisons by Ranking Methods". *Biometric Bulletin* 1:80–83.

AUTHOR BIOGRAPHIES

TIM VAN DIJK is Junior OR Engineer at ORTEC Software Development in Zoetermeer, The Netherlands. He holds an M.S. in Industrial Engineering & Management from the University of Twente, the Netherlands (2014). During his Master's assignment, he worked at ORTEC on the problem of tuning the parameters of a loading algorithm, as described in this paper. His email address is timgvandijk@gmail.com.

MARTIJN MES is an Assistant Professor within the department Industrial Engineering and Business Information Systems at the University of Twente, the Netherlands. He holds an M.S. in Applied Mathematics (2002) and a Ph.D. in Industrial Engineering & Management from the University of Twente (2008). His research interests are health logistics, transportation, multi-agent systems, stochastic optimization, discrete-event simulation, and simulation optimization. His email address is m.r.k.mes@utwente.nl.

MARCO SCHUTTEN is an Associate Professor within the department Industrial Engineering and Business Information Systems at the University of Twente, the Netherlands. He holds an M.S. in Applied Mathematics (1992) and a Ph.D. in Industrial Engineering from the University of Twente (1996). His research interests include planning and scheduling in transportation, project, and production environments. His email address is m.schutten@utwente.nl.

JOAQUIM A. S. GROMICHO is Endowed Chair Professor of Applied Optimization in Operations Research at the VU University in Amsterdam and Algorithmic Innovation Lead at ORTEC Software Development in Zoetermeer, both in The Netherlands. He holds an M.S. in Statistics and Operations Research from the University of Lisbon in his home country Portugal (1990) and a Ph.D. in Optimization from the Erasmus University Rotterdam, The Netherlands (1995). His major research interest is in applied combinatorial optimization. His email address is j.a.dossantos.gromicho@vu.nl.