International Journal of Computer Applications (0975 - 8887)
Volume 177 - No.47, March 2020

Metaheuristic Tuning Generalisation by Cross-Validated Racing

Thiago Henrique Lemos Fonseca
Federal University of Santa Catarina
Department of Informatics and Statistics
Brazil

Alexandre Cesar Muniz de Oliveira
Universidade Federal do Maranho
Departament of Informatics
Brazil

ABSTRACT

Many tuning methods are based on concepts of sensitivity analysis combined with heuristics that tend to reduce the search space by eliminating less promising configurations. Nevertheless, tuning parameters is a task that requires specific and time-consuming experiments, especially when involving large problem instances. This is particularly due to existing methods were not designed to efficiently generalise a tuning of parameters to other instances that did not participate of the training process. In this paper, the recently proposed tuning method named Cross-Validated Racing (CVR) is revised in order to clarify theoretical fundamentals of tuning problem and expand the experiments to make possible evaluating its generalisation capacity against the reduction in the size of the training set. For validation, the Biased Random-Key Evolutionary Clustering Search (BRKeCS) is applied to solve scalable instance groups of Permutation Flow Shop Scheduling Problem. The computation results have demonstrated that CVR is robust in finding an effective parameter setting, requiring training process in only a half of total instance set.

General Terms

Machine Learning, Optimisation

Keywords

Tuning, Irace, Permutation Flow Shop Scheduling, BRKeCS, Evolutionary Clustering Search, Cross-Validated Racing Approach

1. INTRODUCTION

A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic global optimisation algorithms [29]. Metaheuristics are considered as higher level heuristics that are devised to guide other constructive heuristics or local searches to reduce the risk of being trapped into a poor local optimum[28].

One of the main characteristics of metaheuristics is their possibility of being instantiable to any problem formulated from quantitative decision variables and numerical functions that represent objective and constraints [30].

Adaptability can be achieved by a so-called performance parameters, which despite being an elegant algorithmic behaviour adaptation mechanism, it is also a challenge to provide the correct parameter adjustment in order to achieve competitive results [3].

Many metaheuristic tuning methods are based on concepts of sensitivity analysis combined with heuristics that tend to reduce the search space by eliminating less promising configurations [18]. Racing algorithms (RA) are tuning methods in which the performance evolution of a candidate configuration can be performed incrementally in a race competition among candidate configurations [3].

Nevertheless, tuning parameters is a task that requires time-consuming experiments, especially when involving large problem instances. Further more, even when applied to smaller instances, most existing methods are not designed to generalise the parameter tuning so that, in general, training on the entire dataset is required to obtain robustly tuned parameters [2, 18, 1].

This paper is devoted to revisit the recently proposed tuning method named Cross-Validated Racing (CVR) in order to clarify theoretical fundamentals of the method as well as to expand the experiments to make possible evaluating its generalisation capacity against the reduction in the size of the training set.

Generalisation, by a machine learning perspective, allows to learn the best tuning, from training instances, and to verify the performance of the target metaheuristics in instances out of training. This perspective differs from the usual one, since, in general, the practitioners are interested in super tuning algorithms so that they obtain high quality solutions for each instance. However, when the number of problem instances is significantly large, the instance-specific tuning can be very time-consuming. This can be especially relevant when dealing with large or complex instances for which the tuning experiment time can even be prohibitive.

CVR, therefore, comes to provide an alternative for automatic tuning of parameters from a small and representative training set, leaving the larger instances out of this costly process. This possibility is particularly important for the CVR itself since it is designed to run many races and consequently be slower than other tuning methods, when applied in all instances of interest.

For experiments, an application based on the recently proposed Biased Random-Key Evolutionary Clustering Search (BRKeCS) [6, 11] is employed to solve scalable instance groups of Permutation Flow Shop Scheduling Problem.

This paper is organised as follows: Section 2 presents the background on metaheuristic tuning problem, featuring some differences among some existing tuning packages. Section 3 is devoted to revisit Cross-Validated Racing, detailing each step of the tuning

method. In Section 4, the target optimisation problem, Permutation Flow Shop Problem, is presented as well as the target metaheuristic, Biased Random Key Evolutionary Clustering Search, highlighting the parameters to be tuned. The computational results are discussed in Section 5, especially with regard to the capacity of generalisation of the CVR when applied to part of the training instances. Finally, Section 6 is devoted to highlight the main findings and future research.

2. THEORETICAL BACKGROUND

A theoretical framework is now exposed seeking to include elements of understanding of the proposal, while being sufficiently generic to also encompass other approaches that are aimed at optimising the running time necessary to obtain quality solutions after the tuning process.

2.1 Tuning problem

In a tuning problem[3], a finite set of candidate configurations Θ is given along with a class of instances I. A $\theta \in \Theta$, chosen to perform at a time $t \in T$ on an instance $i \in I$ with probability $P_I(i)$, generates a cost c. The set C is a collection of c with the possible cost values of the best solution found in the execution of a θ on the set I.
$P_C(c|\theta,i)$ indicates the probability of c is the cost of the best solution by running for $t(i)$ seconds with configuration θ on instance i. The problem is to find, within a time T, the best setting ($\min_\theta$), according to a criterion C, when the measures P_I e P_C are unknown. Thus, the Tuning Problem can be described by the following components:

—Θ: set of candidate configurations;
—I: set of instances;
—P_I: probability of an instance i to be selected to be solved;
—$t : I \rightarrow \mathbb{R}$: the function that computes the computing time spent to solve a given instance;
—c: a random variable that represents the cost of the best solution found by running the θ setting on the instance i for $t(i)$ seconds;
—$C \subset \mathbb{R}$: range of possible values of c, i.e., cost of best solution found in a run θ configured on instance i;
—$P_C(c|\theta, i)$: probability of c is the cost of the best solution found by running the θ setting for $t(i)$ seconds on the i instance;
—$C(\theta) = C(\theta|\Theta, I, P_i, P_C, t)$: criteria to be optimised concerning to the desirability of θ;
—T: amount of time feasible for experimentation given a set of candidate configurations on a set of instances.

Based on these concepts, the Tuning Problem can be formally described as the 7-Tuple $\langle \Theta, I, PI, PC, t, C, T \rangle$ where the goal is given by:

$$\overline{\theta} = arg\min_{\theta} C(\theta) \quad . \tag{1}$$

It is expected to find the cost μ expressed by the integral:

$$\mu = \int c dP_C(c|\theta, i) dP_I(i) \quad . \tag{2}$$

The above expression cannot be computed analytically since the values of P_C e P_I are not known. However, the samples can be analysed and, according to these measurements, the quantities $\mu(\theta)$ can be estimated [27].

For this purpose, a set of experiments is performed with $|J| = N$. For each experiment $j \in J$, from cost c_j, the amount μ can be estimated by $\widehat{\mu}$:

$$\widehat{\mu} = \frac{1}{N} \sum_{j \in J} c_j \tag{3}$$

For example, given K distinct instances $i_1, i_2, ..., i_K$, with $K \leq N$, and the θ setting is executed n_1 times on instance i_1, n_2 times in instance i_2, and so on. This is equivalent to consider a set of J experiments that is partitioned into subsets $J_1, J_2, ..., J_K$, where $|J_k| = n_k$ e $\sum_k n_k = N$. Each j element in the generic subset J_k is an experiment consisting of running θ once on the instance i_k.

2.2 Some tuning packages

Formally, the definition of [3] covers the main tuning methods divided into two main distinctions: Model-Based and Model-Free approaches [13]. The former determines promising sample points to be investigated, aiming to improve the model. The later draws implicit conclusions based on heuristic rules where choices for interesting parameter vectors to be investigated are often guided by randomness or a simple experimental design.
In race approaches[3],[18], tuning algorithms are designed to provide a better allocation of computational resources between the candidate configurations. Therefore, such methods have superior performance than a typical brute force method. To do this, race algorithms sequentially evaluate candidate configurations and discard those of poorer quality as soon as sufficient statistical evidence is gathered against them. Eliminating lower profit candidates, such approaches tries to speed up the processing and evaluating more promising settings in more instances, thus obtaining more reliable estimates of the algorithm behaviour [9].
CALIBRA algorithm, in turn, uses Taguchis fractional factorial design of experiment coupled with a local search procedure, i.e., it combines exploration and exploitation to find good parameter settings [2]. Relevance Estimation and Value Calibration (REVAC) estimates the distribution of promising parameter vectors by means of information theory. REVAC can be classified as an Estimation of Distribution Algorithm (EDA) and is limited to continuous parameter domains [20].
ParamILS, in turn, is an abstract algorithmic description [14], for which the authors present two implementations: BasicILS and FocussedILS. The former compares simple estimates for the cost statistics of subsequent runs while the later attempts to overcome the overconfidence based on the training instances adaptively by choosing the number of training instances to be used for the evaluation of each parameter setting [14]. Both use iterative local search techniques (ILS) to guide the search for promising areas in the parameter search space. Recently, Variable Neighbourhood Search was proposed to equiped ParamILS instead of ILS [5].

3. PROPOSAL APPROACH : CROSS-VALIDATED RACING

CVR is a cross-tuning technique that employs cross-validation to evaluate the performance of different metaheuristic parameter settings considering different sets of instances in order to find a better overall fit with the shortest execution time from a generalisation perspective by Machine Learning. The approach can be graphically described by the Figure 1:
As can be observed on Figure 1, CVR consists basically of separating a certain percentage of whole problem instances for the

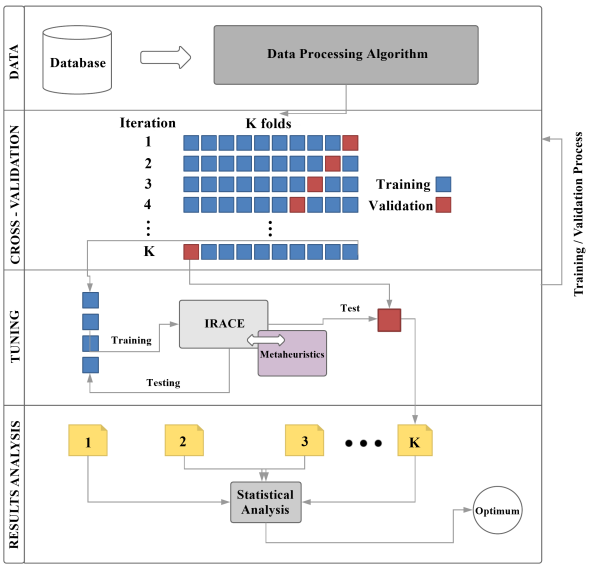

Fig. 1. CVR conceptual design: training instances are distributed in folds and k race iterations are performed over the folds until the configuration most likely to obtain quality solutions eliminates the inferior ones with a certain percentage of confidence.

training step, in which the best set of parameter values is held on. The training instances are then organised into k mutually exclusive folds. The training consists in performing k race tunings using the *leave-one-out* method. Through statistical analysis, the configurations most likely to be the best remain in the race until the best or best family is found. The performance parameters should be adequate enough to allow good performance of the target metaheuristic also when applied to the instances that were out of the training step.

3.1 Data pre-processing

The pre-processing step in CVR comprises the instance organisation in folds in order to make them containing representative subsets. It is expected the method to be robust concerning this first step, but the effect of a bad distribution along folds was not studied yet [11]. In this work, the instances are randomly distributed across the folds.

3.2 K-fold Cross-validation

The CVR estimates the learning method error for instances not included in the training step, that is, the residual error for new instances (not considered yet). In fact, the generalisation theory says that a machine M is able to generalise if, in the training set $T = [x_0, x_1, , x_n; y_0, y_1, , y_n]$ containing the answers $y_0, y_1, ..., y_n$, given by a supervisor for the inputs of values $x_0, x_1, ..., x_n$, the machine is able to predict the responses to the input variables that are not contained in T.
To analyse the estimation of CVR error it is necessary to consider the concepts of **bias** and **variance**. The bias of an estimator is defined as the value of the actual error minus the expected error value:

$$\epsilon - E[\widehat{\epsilon}] \tag{4}$$

The variance is determined by:

$$E[(\epsilon - E[\widehat{\epsilon})^2] \tag{5}$$

where the bias measures the mean accuracy of the estimated error, while the variance measures the variability of the estimated error.
In the k-fold cross-validation process, the instance base S_n is randomly partitioned into k groups of mutually exclusive instances of the same size $P = P_1, P_2, ..., P_k$ where $k-1$ groups are used by the estimation model in a training process and the remaining group for testing. The process is repeated until all groups have undergone a *training* and *validation* state. The estimated model error after validation is given by:

$$\widehat{\epsilon}(S_n, P) = \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{P_j} RACE(P_j, teste) \tag{6}$$

where $RACE(P_j, test)$ is the error resulting from the application of the target metaheuristic tuned with the parameter set P_j in test instance group. The error estimated by the cross validation is a mean of model errors trained with the respective partitions P_j.
It is expected that successive cross-validation repetitions will stabilise the estimation error, reducing the variance, increasing reliability in the acquired results.

3.3 Race tuning

With the generated sets of instances, the running algorithm is applied in a training and testing process. In the training process, the performance of a candidate setting θ is evaluated by being incrementally acquired. In fact, the empirical average:

$$\widehat{\mu_k}(\theta) = \sum_{j=1}^{k} c_j^{\theta}, \tag{7}$$

where results for k experiments is an estimate of the criterion given by:

$$\mu(\theta) = \int c dP_C(c|\theta, i) dP_I(i) \tag{8}$$

as long as the instances $i_1, i_2, ..., i_k$ are sampled according to the measure P_I and the costs $c_1^{\theta}, c_2^{\theta}, ..., c_k^{\theta}$ of the best solutions found in an execution of the θ configuration at a time t are the realisation of stochastic variables described by the unknown measures $P_C(c|\theta, i_1), P_C(c|\theta, i_2), ..., P_C(c|\theta, i_3)$.
Based on these assumptions, one can conclude that a sequence of estimators $\widehat{\mu}_1(\theta), \widehat{\mu}_2(\theta), ...$ can be constructed where $\widehat{\mu}_1(\theta) = c_1^{\theta}$ is simply the cost obtained by running a θ configuration at a given time t on an instance i_1 and the generic term $\widehat{\mu}_k(\theta)$ is given by:

$$\widehat{\mu}_k(\theta) = \frac{(k-1)\widehat{\mu}_{k-1}(\theta) + c_k^{\theta}}{k} \tag{9}$$

where c_k^{θ} is the cost obtained by running the configuration θ at a time t on the k^{th} instance i_k according to the measure P_I. In other words, $\widehat{\mu}_k(\theta)$ is an empirical mean of the observation vector:

$$c^k(\theta) = (c_1^{\theta}, c_2^{\theta}, ..., c_k^{\theta})$$

that can be acquired by adding the term c_k^{θ} to the vector:

$$c^{k-1}(\theta) = (c_1^{\theta}, c_2^{\theta}, ..., c_{k-1}^{\theta})$$

whose average is the estimate $\widehat{\mu}_{k-1}(\theta)$.

The variance of this sequence of estimates decreases by $1/k$, and therefore the estimate of the performance of the candidate θ gets sharper and sharper as k increases and converges to the true expectation $\mu(\theta)$.
Given the possibility of constructing the estimates, $\mu_1(\theta), \mu_2(\theta), ...$ for each candidate $\theta \in \Theta$, the racing algorithm incrementally builds in parallel such sequences for all set candidates in Θ and, as soon as sufficient evidence is obtained that criterion $\mu(\theta^{'})$ for a given candidate $\theta^{'}$ is better than the criterion $\mu(\tilde{\theta})$ of some other candidate $\tilde{\theta}$, and $\theta^{'}$ is discarded for additional evaluations.
A racing algorithm therefore generates a sequence of nested sets of candidate configurations:

$$\Theta_0 \supseteq \Theta_1 \supseteq \Theta_2 \supseteq ...,$$

starting from $\Theta_0 = \Theta$, and the next step, taken from set Θ_{k-1} to Θ_k, is obtained discarding some suboptimal configurations based on available information at step k.
At step k, when the racing candidate set is Θ_{k-1}, a new instance i_k is selected out. Each candidate $\theta \in \Theta_{k-1}$ is tested in i_k and the observed cost c_k^{θ} is appended in the vector $c^{k-1}(\theta)$ to forming the difference vectors $c^k(\theta)$, one for each $\theta \in \Theta_{k-1}$.
The step k is finished by defining Θ_k and discarding from Θ_{k-1} the sub-optimal sets using Friedman statistical test that compares the vectors $c_k(\theta)$ for all $\theta \in \Theta_{k-1}$.
The Algorithm 1 exemplifies the operation of a generic racing algorithm.

Algorithm 1: Generic racing function

Data: $M, Test$
Result: $\overline{\theta}$
begin
  $numExp \leftarrow 0$; ▷ Number of experiments;
  $numInst \leftarrow 0$ ▷ Number of instances;
  $C \leftarrow$ ALLOCATEMATRIX$(maxInstances, \Theta)$;
  $survivors \leftarrow \Theta$;
  while $numExp + |survivors| >$
  $M\&numInstances + 1 > maxInst$ **do**
    $i \leftarrow$ CHOOSEINST() ▷ Random selection of instances ;
    $numInst \leftarrow numInst + 1$;
    forall the $\theta \in survivors$ **do**
      $s \leftarrow$ RUNEXP(θ, i) ▷ run *target-metaheuristic*;
      $numInst \leftarrow numInst + 1$;
      $C[numIns, \theta] \leftarrow$ EVALUATE(s);
    $survivors \leftarrow$
    DELETECANDIDATES$(survivors, C, Test)$
  $\overline{\theta} \leftarrow$ BEST$(survivors, C)$
  return $\overline{\theta}$; ▷ best configuration

In short, the racing phase is the CVR's core. The quest for the best structure of the model is accelerated by the disposal of inferior candidates as soon as statistical evidence of its poor quality is collected. In fact, the measurement of $\mu(\theta)$ relative to the generic candidate θ, can be performed incrementally: the average of K errors, each one with respect to its k examples in the dataset.
This amount can be approximated by the average $\mu_k(\theta)$ which is the optimal parameter setting for each step of the cross-validation. Each k-fold cross-validation test generates an ideal parameter configuration, however the goal is to find the parameter configuration that performs better over all unknown instances. In this way, CVR uses a statistical method based on the estimator $\mu = \int cdP_C(c|\theta, i)dP_I(i)$, $\mu = \int cdP_C(c|\theta, i)dP_I(i)$ that allows to find the optimal parameter configuration, with statistical relevance, that has a shorter computational time among the parameters chosen from all instances.

3.4 Statistical analysis

If the groups of instances generated in the preprocessing step are representative, that is, each group is a representative sample of the total base of instances, the following steps should return k equally effective and statistically significant settings for solving the problem. However, since there is no way to guarantee the representativeness of the instance base (there may be groups of larger or harder instances, i.e. heterogeneous distribution), there is a possibility in the cross-validation process that some groups present more quality configurations than others, so a way is necessary to verify these cases and treat them if they happen.
The statistical test used to guarantee the significance of the CVR result is *Kruskal-Wallis* method (or *non-parametric Anova* - nP-ANOVA). Unlike Kruskal-Wallis's analysis of variance (Fisher's ANOVA, parametric test), nP-ANOVA does not require the assumptions of normality of the variable or homogeneity of variances among treatments. It is characterised as a distribution free test, that is, the theoretical population distribution of the data need not be estimated by means or sample variances for its correct application. When significant differences are detected among treatments by the nP-ANOVA, multiple double comparisons are usually made involving all pairs of treatments. At the end, a configuration optimised for the whole set of instances and able to perform well in new instances is obtained.

4. APPLICATION PROBLEM AND TARGET METAHEURISTIC

Permutation Flow Shop Problem (PFSP) is a part of production schedules that decides the order in which n jobs are processed on m machines, subject to all the jobs are processed in all the machines in the same sequence aiming to minimise some performance measure as the duration of the programming (makespan) or total flow time. PFSP plays a vital role in both automated manufacturing industries and NP-hard class problem [19],[15],[25]. In this work, PFSP is solved by finding the minimum makespan that is the total time that elapses from the first and the last tasks processed.
To accomplish this optimisation problem, a hybrid metaheuristic constructed from Biased Random-Key Genetic Algoritm (BRKGA) [10],[12] and Evolutionary Clustering Search (ECS) [22], has been recently proposed [11]. The Biased Random Key Evolutionary Clustering Search (BRKeCS) is designed to perform a clustering process for detecting promising search areas in a coded search space instead of the original problem one [6].
Applications involving BRKGA and CS are first designed to deal with the Minimisation of Tool Switches Problem (MTSP)[6]. MTSP is a combinatorial optimisation problem that seeks a sequence of jobs that minimises the number of tool switches required for a given processing[31].
BRKGA is an evolutionary meta-heuristic that encodes a candidate solution as a vector of random keys, i.e., a vector with real-valued components randomly generated in the interval $[0, 1]$. In combinatorial optimisation, the algorithm seeks in the solution space indirectly, by exploring the continuous unit hypercube and mapping continuous solutions to discrete ones using a encoder/decoder scheme [12].

Clustering Search (CS) is a generic way of combining search metaheuristics with clustering in order to detect promising regions so that these regions are subsequently exploited by problem-specific heuristics [8].
CS attempts to locate promising search areas by framing them by dynamic clusters that are represented by their respective *center*. Clusters are created, activated and eventually removed depending on the search dynamics performed by the support metaheuristic [22, 6, 21, 17]. In Evolutionary Clustering Search (ECS), the support meta-heuristic is an evolutionary algorithm, working as continuous generator of candidate solutions[8].
In this work, BRKGA is employed to generate candidate solutions for the clustering process, since it encodes a solution as a vector of random keys and produces a feasible solution through the decoder. BRKGA makes possible to simplify some components of the CS framework, requiring implement only the decoder and local search heuristic [6, 11].
The local search is an essential part to effectiveness of BRKeCS, since the promising areas of search space should be explored as soon as they are discovered. BRKeCS uses a 2-*opt*-based enumerative heuristic for promising areas exploitation [7, 8]. *Decoder* is designed to guarantee that, given as input a vector of random keys, it produces a feasible discrete solution of the combinatorial optimisation problem [12].
Despite the number of applications involving CS based algorithms [21, 17], a certain difficulty rests on the need for specific procedures for distance metric, assimilation operator and local search, beyond native meta-heuristic operators. Besides, a lot of performance parameters need to be tuned as well[22].
The following parameters are needed to be tuned with respective ranges: number of clusters *NumCl* (1,20), population size P (1,1000), mutation rate p_m (0.0,1.0), elite rate p_e (0.0,1.0), maximum number of individuals to make a cluster promising λ (0.0,1.0), maximum number of local search r_{Max} (1,10), and native local search parameter, sometimes expressed as $width$ (1,5) and $depth$ (1,5) in enumerative methods [8],[11].

5. COMPUTATIONAL RESULTS

For tests, a benchmark composed of two different set of instances has been used: the first one, proposed by Taillard [26], commonly found in literature. Taillard's instances are composed of 12 groups, ranging in size from 20 to 500 tasks and 5 to 20 machines, whereby each group consists of 10 instances with the same size. The second set, composed of randomly realistic instances, is generated with increased difficulty degree obtained by employing a task and machine correlation[32].
For tuning method, the *Iterated Racing for Automatic Algorithm Configuration* (Irace package [18]) has been employed for both CVR's built-in racing method as well as a competitive method for comparison purposes in its standalone version, without cross-validation. In both cases, Irace package was roughly configured in the following way: $\{maxExperiments = 1000, maxTime = 20000, budgetEstimation = 0.02, mu = 5, seed = NA, softRestart = 1, confidence = 0.95, elitist = 1\}$. Most of these configuration parameters are well-known for practitioners in algorithm tuning. The parameter mu is related to the number of configurations evaluated at each iteration.

5.1 BRKeCS without CVR tuning

Initially, a test of solution quality has been led by running BRKeCS 5 times, using the same number of objective function calls as other approaches found in literature: MOACSA [33], CR(MC) [4], ACO and HAMC1 algorithms [16, 23, 24]. Table 5.1 shows the percentage difference of algorithms with respect to PFSP instances 20x20. As can be observed, BRKeCS even without applying CVR tuning has a competitive performance compared with those approaches.

Table 1. Gap obtained by 4 optimisation approaches in Taillard's instances 20x20

Instances	HAMC1	CR(MC)	MOACSA	ACO	GA	BRKeCS
Ta021	2.51	7.41	0	5.44	0.45	0.13
Ta022	11.96	3.42	2.56	5.43	0	0*
Ta023	0.46	0	4.31	6.06	7.13	0.08
Ta024	11.41	0.29	0.30	6.70	0	0.04
Ta025	0	5.16	2.19	7.90	0.21	0.21

* In this instance has been found a new lower bound.

For the biggest instances, BRKeCS has showed satisfactory performance when compared with the other metaheuristics, as ACO [16]. Table 5.1 introduces more results of problem instances considered of high and medium difficulty.

Table 2. Best solution obtained by ACOC and BRKeCS in Taillard's instances with high and medium difficulty

Instances	jobs	machines	Best sol.	ACOC	BRKeCS
Ta041	50	10	3025	4036	3109
Ta051	50	20	3875	7080	3923
Ta061	100	5	5493	6245	5493
Ta071	100	10	5779	7563	5782
Ta081	100	20	6282	13270	6378

5.2 Minimum training instance subset

Considering that CVR needs to execute k race tuning to model the parameter distribution and infer the best settings, it is valid to gauge the minimum set of instances that must be included in the tuning process in order to avoid loss of quality in the generalisation step. For setup purposes, the experiments described bellow were performed using $k = 5$ folds and the total number of instances per fold was calculated according to the size of the training subset.
In this work, the process of discovering the minimum instance subset consists of incrementally adding instances of each group in the tuning process and analysing their impact over the BRKeCS performance when applied to the hardest problem instances (group 12, i.e., 500-task group) that were left out of the training process.
Tests has been performed through 6 tuning experiments, whereby the first one has used best parameters found by CVR for instances of the same test group, including instances of group 12 (*CVR-specific*). In other words, the residual error acquired is computed over the same instances used in training. In theory, the BRKeCS's best results are expected for this experiment since the performance parameters are found specifically for that group, making no use of generalisation. The problem in this strategy is the computational cost required to tuning BRKeCS specifically for all instance groups, including larger ones (groups 11 and 12).
The following 4 experiments (2_{nd}, 3_{rd}, 4_{th} and 5_{th} experiments) have used best parameters found by CVR for only a part of the 120 problem instances (*CVR-percentage*). In other words, based on training over a certain percentage of representative instances, BRKeCS was tuning and, later, tested on the instances of group 12 (generalisation group), from which the average of residual error was calculated.
For *CVR-percentage*, the first training subset was chosen by adding 5 instances, at random, of each group, except the generalisation

group 12, totalling 55 representative instances. Observe that considering the total of 120 instances, 5×11 corresponds approximately to 46% of training set.
The second training subset was built adding 5 instances, at random, of the less expensive groups (groups 1 to 10), totalling 50 representative instances (42% of training set). The third training subset was formed by taking only 3 instances, at random, of the same less expensive groups (groups 1 to 10), totalling 30 representative instances (only 25% of training set).
The fourth training set was further reduced so as to contain only 1 instance, chosen at random, of each group of less expensive instances, totalling only 10 instances (8% of training set). In the 6_{th} experiment, the BRKeCS parameters have been tuned using literature, considering the type of problem to be solved [6].
Similar 6 experiments, using same percentages, have been performed by applying BRKeCS over a new group of PFSP instances: the realistic ones, built by mixed correlation process [32]. Once again *CVR-specific*, *CVR-percentage* and literature tuning are employed in the experiment and later the BRKeCS's residual error averages are obtained as well.
The average of residual error acquired over the generalisation group (group 12) during the generalisation test is showed in Figure 5.2 for both PFSP instances: Taillard (blue line) and realistic (red line) instances. Preliminary evaluations has demonstrated a CVR predisposition to keep the residual error lower than 0.05 when used above 25% of the training instances. These residual errors are less than ones obtained just using literature parameters. It is worth emphasising that such good residual errors were obtained between 30 and 50 training instances, taken from a total of 120 ones.
As can be seen in Figure 5.2, the smallest residual errors were obtained by tuning specifically to each group of instances (*CVR-specific*), which is equivalent to training on all available ones, with no concern for time-of-experiment issues. As the training dataset is reduced, the residual error tends to increase, but depending on the percentage of instances used to compose the dataset, it is still more advantageous than using only the expertise found in the literature. It can be observed that better results than the literature can be obtained by performing training with at least 30% of the instances, for both, Taillard and Realistic cases.

5.3 Comparing CVR and Irace

Considering that CVR uses a built-in race tuning method, Irace package in this work, it is valid to compare the BRKeCS's performance tuned by both methods. In particular, it is interesting to assess the generalisation capacity of both methods: CVR and Irace. Recalling that the generalisation capability allows to use only a part of the problem instances in the training process, which is useful to avoid the need for training with large instances. Therefore, in this experiment, once again only a percentage of Taillard's instances are used in tuning process and the largest ones (500-tasks instances - group 12) were left out of training to be used in the generalisation phase.
Figure 3 shows the BRKeCS's residual error behaviour for 46%, 42%, 25% and 8% of training database, when tuned by CVR and Irace.
As one can observe, CVR makes BRKeCS to perform more robustly by finding parameters with greater generalisation capacity. The performance of BRKeCS trained by CVR on only 8% of the dataset (training instances $= 10$ and residual error $= 0.06074$) is comparable to its performance when trained by Irace using almost half of the dataset (training instances $= 55$ and residual error $= 0.06246$). Observe that, in addition, the performance degradation of Irace-trained metaheuristic is much more pronounced than using CVR.
Despite the apparent CVR superiority in this experiment, observe that Irace was not designed to generalise and needs to be trained with the complete dataset. Since 500-task (group 12) instances did not participate of the training, Irace does not know anything about them. Another point to be considered is that applying CVR over 8% of instances should not be as stable as Irace with 46%, that is, there is not enough instances for CVR to generalise with 95% confidence every time.

5.4 Folds and instances per fold

For tuning by CVR, it is necessary to define two parameters: number of folds, (k), and number of instances per fold, (i_k). The quality of the training and consequently the generalisation capacity must be influenced by these parameters. All previous experiments were carried out with $k = 5$ folds and varying i_k depending the total number of required training instances.
In this section, additional experiments are described aiming to evaluate the impact of different CVR parameter settings over BRKeCS tuning. Once again, instances of group 12 were left out of the training and used for validation sake.
Considering that the number of instances used in training is critical for the generalisation process, two very extreme situations were tested using 25% of training instances (30 instances of groups 1 to 10): 3 folds and 10 instances per fold (case 1) , and 10 folds and 3 instances per fold (case 2). The underlying question to be answered is: which is more advantageous, for the same number of instances, to have more folds or more instances in each fold?
The residual error obtained by BRKeCS in 50 trials on the instances of group 12 after CVR tuning has been 0.05463 and 0.07416 for, respectively, case 1 ($k = 3 \times i_k = 10$) and case 2 ($k = 10 \times i_k = 3$). As can be observed, 3×10-tuned BRKeCS has reached a smaller residual error, over largest instances (generalisation group). Moreover, note that 0.05463 of residual error obtained with case 1 is slightly high than 0.049325, previously achieved for the same number of training instances, 30, and depicted on figures 5.2 and 3, in which were used $k = 5 \times i_k = 6$.
There is therefore a relationship between k and i_k that induces a tuning with better robustness from minimal amounts of folds and instances per folds. Analysing the behaviour of the method in the extreme points, one can affirm that increasing k allows reducing dependence on the representativeness of instances in each training fold since this can eliminate the discrepant tuning models. Increasing i_k, in turn, should reduce the risk of over fitting, that is, over-tuned parameters for a small number of instances.
Hence the extremes $k = 1$ and $i_k = 1$ are highly detrimental to the good performance for CVR. The former is equivalent to apply only the racing method over many instances with no cross-validation. The later is similar to applying cross-validation to completely discrepant models.
In the experiment in question, it seems less harmful, in a presence of a small number of instances (only 25%), to set up CVR in order to form fewer folds with more instances instead of more folds with fewer instances. It is possible to conclude that the number of instances per fold has a greater weight than the number of folds for the generalisation capacity.

6. CONCLUSION

This paper revisits the Cross-Validated Racing (CVR): a tuning technique that employs cross-validation to evaluate the perfor-

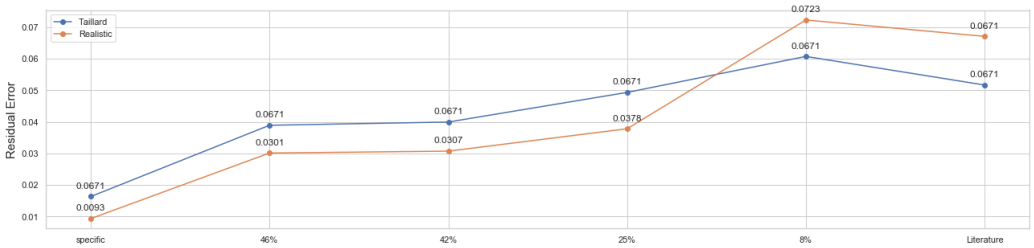

Fig. 2. BRKeCS's residual error average obtained in the generalisation test: at the extreme points are the residual errors referring to CVR-specific and parameters found in the literature. The larger the training set, the better the quality of the solutions found by BRKeCS.

mance of different metaheuristic parameter settings in order to find a better overall fit with the shortest execution time from a machine learning perspective.

CVR is applied to tune the recently proposed Biased Random Key Evolutionary Clustering Search in order to achieve competitive results for solving instances of Permutation Flow Shop Problem. Computational experiments have been carried out considering the expertise found in literature as well as racing method, using Taillard's and realistic instances.

The CVR generalisation capacity has showed a predisposition to keep the residual error low when used between between 25% and 42% of training instances. When compared with Irace package, CVR makes BRKeCS to perform more robustly by finding parameters with greater generalisation capacity. In the experiments performed, it has been required around 6 times fewer instances of training to obtain the same results over test instances. The generalisation process allows spending less training time, specially whether instances outside of training are the largest ones.

A natural gap lies in experimenting CVR over other optimisation algorithms and target problems, especially in applications that may contribute to new frontiers of state-of-the-art in combinatorial optimisation. In this sense, one aspect that deserves special interest concerns the ability of the CVR to select algorithms, not just algorithm parameters.

7. ACKNOWLEDGMENTS

This research is partially supported by FAPEMA, BRAZIL (Proc. UNIVERSAL 01294/16).

8. REFERENCES

[1] Eduardo Batista de Moraes A. Barbosa, Edson Luiz Frana A Senne, and Messias Borges A. Silva. Improving the performance of metaheuristics: An approach combining response surface methodology and racing algorithms. *International Journal of Engineering Mathematics*, 2015.

[2] Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research*, 54(1):99–114, 2006.

[3] Mauro Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer Publishing Company, Incorporated, 1st ed. 2005. 2nd printing edition, 2009.

[4] Rajendran C. Heuristics for scheduling in flowshop with multiple objectives. *European Journal of Operational Research*, 82:540555, 1995.

[5] Leslie Prez Cceres and Thomas Sttzle. Exploring variable neighborhood search for automatic algorithm configuration. *Electronic Notes in Discrete Mathematics*, 58:167 – 174, 2017. 4th International Conference on Variable Neighborhood Search.

[6] A.A. Chaves, L.A.N. Lorena, E.L.F. Senne, and M.G.C. Resende. Hybrid method with cs and brkga applied to the minimization of tool switches problem. *Computers & Operations Research*, 67:174 – 183, 2016.

[7] G.A Croes. : A method for solving traveling salesman problems. *Operations Research*, 6:791–812, 1958.

[8] A.C.M de Oliveira and L.A.N. Lorena. Hybrid evolutionary algorithms and clustering search. In Ajith Abraham, Crina Grosan, and Hisao Ishibuchi, editors, *Hybrid Evolutionary Algorithms*, volume 75 of *Studies in Computational Intelligence*, pages 77–99. Springer Berlin / Heidelberg, 2007.

[9] Katharina Eggensperger, Marius Lindauer, and Frank Hutter. Pitfalls and best practices in algorithm configuration. *CoRR*, abs/1705.06058, 2017.

[10] M. Ericsson, M.G.C. Resende, and P.M. Pardalos. A genetic algorithm for the weight setting problem in ospf routing. *Journal of Combinatorial Optimization*, 6(3):299–333, Sep 2002.

[11] Thiago Henrique Lemos Fonseca and Alexandre Cesar Muniz de Oliveira. Tuning of clustering search based metaheuristic by cross-validated racing approach. In Ignacio Rojas, Gonzalo Joya, and Andreu Catala, editors, *Advances in Computational Intelligence*, pages 62–72, Cham, 2017. Springer International Publishing.

[12] José Fernando Gonçalves and Mauricio G. C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, Oct 2011.

[13] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[14] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[15] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[16] Tarik Lamoudan, Fatima Elkhoukhi, Jaouad Boukachour, et al. Flow shop scheduling problem with transportation times, two-robots and inputs/outputs with limited capacity. *International Journal of Intelligent Computing Research (IJICR)*, 2(1/2/3/):244–253, 2011.

[17] Y. Liu, Y.and Ouyang, H. Sheng, and X. Zhang. Artificial bee and differential evolution improved by clustering search on continuous domain optimization. *Signal Image Technology and Internet Based Systems*, 2008.

[18] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The packageirace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[19] E. Mokotoff. Minimizing the makespan and total flow time on the permutation flow shop scheduling problem. In J. Blazewicz, M. Drozdowski, G. Kendall, and B. McCollum, editors, *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009), 10-12 Aug 2009, Dublin, Ireland*, pages 479–506, 2009. Paper.

[20] Volker Nannen and Agoston Endre Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 183–190. ACM, 2006.

[21] A.C.M Oliveira and T. S. Costa. Artificial bee and differential evolution improved by clustering search on continuous domain optimization. *Soft Computing*, pages 1–12, 2014.

[22] Alexandre C. M. Oliveira and Luiz A. N. Lorena. *Advances in Artificial Intelligence – SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil,September 29-Ocotber 1, 2004. Proceedings*, chapter Detecting Promising Areas by Evolutionary Clustering Search, pages 385–394. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[23] Ahmad Rabanimotlagh. An efficient ant colony optimization algorithm for multiobjective flow shop scheduling problem. *world academy of science, Engineering and Technology*, 75(54):127–133, 2011.

[24] Selvakuar S. J. Ravindran D., Noorul Haq A. and Sivaraman R. Flow shop scheduling with multiple objective of minimizing makespan and total flow time. *International Journal of Advanced Manufacturing Technology*, 25:10071012, 2005.

[25] S Reza Hejazi and Soroush Saghafian. Flowshop-scheduling problems with makespan criterion: A review. 43:2895–2929, 07 2005.

[26] ric Taillard. Scheduling instances, 2013.

[27] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method (Wiley Series in Probability and Statistics)*. 2 edition.

[28] Said Salhi. *Heuristic Search Book: The Emerging Science of Problem Solving*. Springer, 2017.

[29] Kenneth Sörensen and Fred W. Glover. *Metaheuristics*, pages 960–970. Springer US, Boston, MA, 2013.

[30] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, New Jersey, USA, 2009.

[31] Christopher S. Tang and Eric V. Denardo. Models arising from a flexible manufacturing machine, part i: Minimization of the number of tool switches. *Operations Research*, 36(5):767–777, 1988.

[32] Jean-Paul Watson, Laura Barbulescu, L. Darrell Whitley, and Adele E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14(2):98–123, 2002.

[33] B Yagmahan and M. M. Yenisey. A multi-objective ant colony system algorithm for flow shop scheduling problem. *Expert Systems with Applications*, 37:1361–1368, 2010.

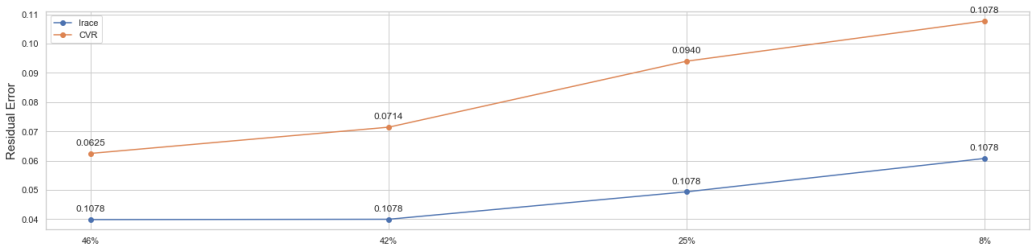

Fig. 3. Behaviour of CRV and Irace for gradual decrease of instances for training: BRKeCS performs better on the test instances with CVR parameters even with a reduced training dataset (8% - 10 instances).