Information Sciences 619 (2023) 153–171

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An investigation of F-Race training strategies for cross domain optimisation with memetic algorithms

Düriye Betül Gümüş [a,b,*], Ender Özcan [b], Jason Atkin [b], John H. Drake [c]

[a] Nottingham Business School, Nottingham Trent University, UK
[b] Computational Optimisation and Learning (COL) Lab, School of Computer Science, University of Nottingham, UK
[c] School of Computing and Mathematical Sciences, University of Leicester, UK

ARTICLE INFO

Article history:
Received 21 December 2021
Received in revised form 27 October 2022
Accepted 4 November 2022
Available online 10 November 2022

Keywords:
Parameter tuning
Cross-domain search
Combinatorial optimisation
Metaheuristics

ABSTRACT

Parameter tuning is a challenging and time-consuming task, crucial to obtaining improved metaheuristic performance. There is growing interest in cross-domain search methods, which consider a range of optimisation problems rather than being specialised for a single domain. Metaheuristics and hyper-heuristics are typically used as high-level cross-domain search methods, utilising problem-specific low-level heuristics for each problem domain to modify a solution. Such methods have a number of parameters to control their behaviour, whose initial settings can influence their search behaviour significantly. Previous methods in the literature either fix these parameters based on previous experience, or set them specifically for particular problem instances. There is a lack of extensive research investigating the tuning of these parameters systematically. In this paper, F-Race is deployed as an automated cross-domain parameter tuning approach. The parameters of a steady-state memetic algorithm and the low-level heuristics used by this algorithm are tuned across nine single-objective problem domains, using different training strategies and budgets to investigate whether F-Race is capable of effectively tuning parameters for cross-domain search. The empirical results show that the proposed methods manage to find good parameter settings, outperforming many methods from the literature, with different configurations identified as the best depending upon the training approach used.

1. Introduction

In the context of the single-objective problems addressed in the present paper, combinatorial optimisation can be defined as the process of attempting to find the best solution among all possible solutions for a problem, based on some objective value defining the quality of a solution [1]. Many real-world combinatorial optimisation problems are very complex and their search spaces too large to be exhaustively explored. Metaheuristics are commonly used to solve such complex problems, aiming to find a near optimal solution within a reasonable amount of time. A metaheuristic is defined by [2] as:

> "a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimisation algorithms. A problem-specific implementation of a heuristic optimisation algorithm according to the guidelines expressed in a metaheuristic framework is also referred to as a metaheuristic.".

* Corresponding author.
E-mail addresses: betul.gumus@ntu.ac.uk (D.B. Gümüş), ender.ozcan@nottingham.ac.uk (E. Özcan), jason.atkin@nottingham.ac.uk (J. Atkin), john.drake@leicester.ac.uk (J.H. Drake).

https://doi.org/10.1016/j.ins.2022.11.008
0020-0255/

According to this definition, the term 'metaheuristic' has been used to define two different concepts. The first one is the high-level framework that is a set of strategies used in the development of optimisation algorithms [3]. The metaheuristic framework is problem domain independent and can be applied across different problem domains. For example, iterated local search has been applied to a range of optimisation problems, including the quadratic assignment problem [4] and bin packing problems [5] among others. The second usage of metaheuristic denotes the specific implementation of a heuristic optimisation algorithm within a general framework. A metaheuristic implementation is problem domain specific. For example, the implementation of iterated local search for vehicle routing can not be used for solving the graph coloring problem. Metaheuristics (i.e., their implementations) are custom tailored to a specific problem domain, so modifying a metaheuristic to solve another problem domain requires significant expert intervention.

Almost all heuristic optimisation methods come with parameters that need to be set, with overall performance highly sensitive to the values of these parameters [6]. Determining the best parameter setting of a search method, i.e. parameter tuning, is a challenging task and crucial for improved metaheuristic performance [7]. There are a number of different common approaches to parameter tuning. Some researchers tune the metaheuristic implemented for a particular domain for each problem instance considered. Others sample a set of training problem instances from a domain, tune the metaheuristic, and apply it with the best parameter settings to a set of 'unseen' problem instances from that domain. To this extent, most parameter tuning studies in the scientific literature focus on a single domain and set the algorithm parameters specifically for that domain, sometimes even for a single problem instance. Additionally, a wide range of automated parameter tuning methods have been deployed in the literature, such as F-Race [8], REVAC [9], irace [10] and CRS-Tuning [11].

An emerging research area is the design of general-purpose problem-independent search methodologies which perform well over a range of different problems, known as 'cross-domain search' methods. A key goal is to increase the level of generality of search methodologies, so that they are applicable to various problems without modification, while still delivering good overall performance [12]. The definition of the term 'hyper-heuristic' was proposed by [13] as "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems". Hyper-heuristics have been succesfully deployed to solve a wide range of optimisation problems such as project scheduling [14,15], the traveling salesman problem [16] and search-based software testing [17]. The *HyFlex* framework was developed to support the design and comparison of cross-domain methods, and was utilised in the first Cross-domain Heuristic Search Challenge (CHeSC2011) [18]. HyFlex is a framework which contains all of the problem-specific components of a number of optimisation problems, including low-level heuristics, solution representation and initialisation routines, and an evaluation function. The high-level search method implemented within HyFlex can be a metaheuristic or a hyper-heuristic, using the available low-level heuristics within the framework to modify a solution during the search.

We classify the parameters of meta/hyper-heuristics implemented in HyFlex as either 'algorithm parameters' or 'heuristic parameters'. Algorithm parameters control the behaviour of the higher level search strategy, while heuristic parameters determine the extent of the changes that the corresponding low-level heuristics make to a solution. Previous studies employ a number of different ways to manage these parameters. For the heuristic parameters, a considerable number of approaches use arbitrary 'default' values, while some simply randomly choose values for these parameters. The general practice is to adapt the parameters to a new instance or problem domain. There has been very limited research effort dedicated to tuning those parameters effectively for cross-domain heuristic search. It is widely recognised that parameter tuning is crucial for improved metaheuristic performance when solving a single problem. However, the value and importance of cross-domain parameter tuning are yet to be extensively investigated. To the best of our knowledge, there are only a few studies which utilised parameter tuning for cross-domain approaches, and only to a limited extent. For example, [19] tuned their algorithm parameters with F-Race, but no parameter tuning experiments were performed for heuristic parameters. In this case, no detail was provided regarding the tuning process except the resultant tuned values. A CHeSC2011 competitor, NAHH [20], utilised the irace method for parameter tuning. However, as the parameter tuning experiments were performed for each problem domain independently, their approach cannot be considered as truly 'cross-domain' in the same sense as we propose in this paper. Although algorithm parameters were tuned by REVAC by [21,9], the lower level heuristic parameters were controlled during the search rather than tuned.

In our previous study [22], a steady-state memetic algorithm (SSMA) was used to solve instances from nine problem domains using HyFlex, with both the algorithm and heuristic parameters of the SSMA tuned via a design of experiments approach (the Taguchi method). Here, an automated parameter tuning method, F-Race [8], is deployed as a cross-domain parameter tuning approach. In order to investigate the suitability of F-Race for cross-domain parameter tuning, a number of training strategies considering different instances and run times are tested and compared. The performance of different SSMAs using the parameters found by F-Race with different training strategies and the Taguchi method from [22] are compared to the SSMA of [23] and the competing algorithms from the original CHeSC competition.

The structure of this paper is as follows: Section 2 provides a background covering HyFlex, including the selection hyper-heuristics developed with this framework and their corresponding parameters. Section 3 describes the methodologies that are used in this paper. The experimental design and computational results are presented in Section 4 and Section 5 respectively. Section 6 offers a conclusion and potential directions for future work.

2. Cross-domain heuristic search and the HyFlex framework

In cross-domain heuristic search, the aim is to design generally applicable search methodologies that perform well not only across instances of a single problem, but also across different problem domains [24]. The HyFlex framework was

introduced to support and stimulate the development and comparison of more general purpose search methods, able to generalise well over a variety of problem domains and instances. Originally developed to support an international competition (CHeSC2011), this framework provides problem-specific algorithm components so that algorithm designers can focus on developing general-purpose heuristic optimisation algorithms without requiring problem domain expertise [18]. The first version of the framework provided four problem domains (boolean satisfiability (SAT), one dimensional bin packing (BP), personnel scheduling (PS) and permutation flowshop (PFS)), with two additional problem domains made available shortly after CHeSC2011 (traveling salesman problem (TSP) and vehicle routing problem (VRP)). Subsequently, [25] implemented three further problem domains in HyFlex (0–1 Knapsack (0–1 KP), Max-Cut (CUT), and Quadratic Assignment (QAP)), taking the total number of problem domains available within this framework to nine.

There are four types of low-level heuristics (move operators) in HyFlex: mutation, ruin-recreate, crossover and local search. For the mutation, ruin-recreate and local search heuristics, there are parameters. The *intensity of mutation (IoM)* parameter is used in mutation and ruin-recreate heuristics to determine the extent of changes that the corresponding operator will make to the input solution. The *depth of search (DoS)* parameter determines the extent to which local search heuristics are applied. These parameters are used to control the behaviour of a low-level heuristic and take values in the interval [0,1]. The effect of a parameter value is problem and heuristic dependant, with the problem domain designer required to give an appropriate meaning to these parameters [26]. For example, the depth of search parameter in the SAT domain controls the number of iterations that the local search heuristic will complete, whereas in the PS domain it is used to determine the search time for the local search operator. The intensity of mutation parameter in PFS controls the number of randomly selected elements in the permutation to shuffle, whereas in the BP domain it decides the number of bins to remove pieces from.

Although the underlying details of the behaviour controlled by these parameters do not need to be known by the high-level algorithm designer, these parameters require effective parameter tuning or control to ensure high-quality algorithm performance. In this paper, we use F-Race with a number of different training strategies to tune the intensity of mutation and depth of search parameters, along with two algorithm parameters, providing a performance comparison to a number of existing methods from the literature.

2.1. Selection hyper-heuristics using HyFlex and their parameters

In this section, the methodologies and the parameter tuning and control strategies of selection hyper-heuristics using HyFlex are discussed. There are a growing number of studies in this area, each with their own methods and associated parameters. We refer to the parameters included in the design of the high level hyper-heuristics as 'algorithm parameters'. Since these hyper-heuristics were implemented in HyFlex, they also use the HyFlex parameters (IoM and DoS) for low-level heuristics. For a recent survey on selection hyper-heuristics including those developed using HyFlex, we refer the interested reader to [27].

Most of the early work using the HyFlex framework, including many of the competitors to CHeSC2011, did not perform any parameter tuning or control. These methods either made a random choice or used arbitrary 'default' values of 0.2 for the IoM and DoS parameters (e.g. [28]). Although some controlled heuristic parameters during the search (e.g. [29]), there was only one competitor (NAHH) to utilise a systematic parameter tuning method for these parameters [20]. However, as the parameters were tuned for each problem separately, this cannot be considered as cross-domain parameter tuning in the sense that we use in this paper. Despite the additional effort required to tune these parameters, NAHH was still outperformed in the competition by methods which did no parameter tuning at all (e.g. VNS-TW [28] and ML [30]). More recent work has demonstrated the value of parameter tuning for success over this benchmark (e.g. [22]), highlighting both the importance of parameter tuning and the variation in impact it has for different methods.

AdapHH [29] was the winner of the CHeSC2011 competition. It utilises an adaptive heuristic selection method and adaptive threshold move acceptance method. This algorithm determines the parameter settings of heuristics (IoM and DoS) adaptively via a reinforcement learning scheme. There are also a large number of algorithm parameters whose values were determined after some preliminary experiments, with no special tuning method applied. VNS-TW [28] and ML [30], the second and third placed entrants to the CHeSC2011 competition, are fundamentally based on two well-known search strategies from the literature: variable neighborhood search and iterated local search respectively. Although both of these methods performed no systematic or automated parameter tuning, using the default value of 0.2 for the heuristic parameters, they both showed extremely strong performance over a wide range of benchmarks among the field of twenty entrants to the competition.

NAHH [20] consists of several schemata that were tuned separately for each problem domain. The heuristic parameters, IoM and DoS, were defined for all schemata and tuned using irace [31]. If the application of a local search heuristic takes more than 10 s, the DoS parameter is set to the default value. As mentioned previously, the parameter tuning experiments using irace were carried out independently for each of the problem domains, rather than in a cross-domain manner. [32] modified and improved the traditional choice function of [33], proposing a modified choice function heuristic selection method for cross-domain search. The choice function scores heuristics based on a combination of three measures and applies the heuristic with the highest score at each step. Each measure has a weight to control the intensification and diversification

during the search; thus, choosing the right parameter values for these weights is crucial. In their study, an adaptive mechanism inspired by reinforcement learning was proposed to control these parameter values. The modified choice function delivered a great improvement over the original choice function for the HyFlex benchmark. The results show the importance of parameter tuning since the classic choice function ranked 20^{th} among the CHeSC2011 competitors, while the modified choice function achieved rank 12^{th}.

[34] used a data science technique called tensor analysis in order to detect the latent relationships between the low-level heuristics and the hyper-heuristic. This method ranked second against the CHeSC2011 competitors, fixing the low-level heuristic parameters to the default value of 0.2. [35] presented a method to automatically design the high-level strategy of a hyper-heuristic with the Q-learning reinforcement learning technique. Although this method performed well, ranking second against the CHeSC2011 competitors, no information was provided for the settings of the IoM and DoS parameters used in this work.

Since the CHeSC2011 competition, a number of papers have presented results demonstrating improved performance over AdapHH. Kheiri and Keedwell [36] used a method based on hidden Markov models (HMM), called the sequence-based selection hyper-heuristic (SSHH). Their method maintains a matrix of probabilities, determining the likelihood of selecting a particular parameter value for IoM and DoS from a discretised set of possible values. [37] presented a fair-share iterated local search hyper-heuristic with a conservative restart condition. This method performed well despite using the default value of 0.2 for the heuristic parameters. Adubi et al. [38] combined some of the ideas from these two papers, proposing an ILS-based method that also used a probability matrix to control parameter values. [39] proposed an iterated multi-stage selection hyper-heuristic which considers the idea that not all low-level heuristics are necessarily useful for a particular problem domain. The parameter values of the low-level heuristics are set to 0 initially, then the relevant parameter setting of a selected heuristic is updated to a random value within [0,1] if the move does not improve the candidate solution, otherwise the parameter setting remains the same. Zhao et al. [40] introduced a multi-stage hyper-heuristic, which incorporates both single-point and population-based search mechanisms. In this work, the IoM and DoS parameters in HyFlex are set to 0.4 and 0.3 respectively, based on experimental experience.

3. Related methodologies

In this work, we will focus on the performance of parameter tuning methods for a steady state memetic algorithm (SSMA) [23], using instances from a range of HyFlex problem domains. Although our main focus is on different training strategies for the F-Race automated parameter tuning method, we will also compare to previous parameter tuning work based on Taguchi design of experiments. The background of SSMA, F-Race and Taguchi design of experiments are discussed in the subsequent subsections.

3.1. Steady-state Memetic Algorithm (SSMA)

Originally presented in the context of HyFlex by [23], the steady-state memetic algorithm (SSMA) is a metaheuristic approach, combining the ideas of evolutionary computation and local search. Operating over a population of solutions, SSMA iteratively attempts to improve the quality of the population via successive application of crossover, mutation and local search heuristics. SSMA utilises low-level heuristics from all four categories within HyFlex (mutation, ruin-recreate, crossover and local search). In addition to the mutation heuristics, ruin-recreate heuristics are also used for the purpose of 'mutation' within SSMA. The pseudocode of this approach is given in Algorithm 1.

Algorithm 1: Pseudocode of the SSMA

```
Set the parameter values for PopSize, TourSize, IoM and DoS
Generate an initial population consisting of PopSize random individuals
Apply a random local search heuristic to each individual
while termination criterion is not satisfied do
  Parent1 ← Select-Parent(population, TourSize)
  Parent2 ← Select-Parent(population, TourSize)
  XO_ID  = Random(1, MAX_CROSSOVER)
  Child   ← ApplyCrossover (XO_ID, Parent1, Parent2)
  MU_ID  = Random(1, MAX_MUTATION)
  Child   ← ApplyMutation (MU_ID, IoM, Child)
  LS_ID  = Random(1, MAX_LOCALSEARCH)
  Child   ← ApplyLocalSearch(LS_ID, DoS, Child)
  Replace the worst individual in the population with Child
end while
```

Firstly, a population of solutions is created using the HyFlex initialisation routine provided in each domain with the desired number of individuals specified by the value of the *population size (PopSize)* parameter. As a part of the initialisation process, each generated individual is improved by applying a randomly selected local search operator. Then the evolutionary cycle starts. At each step, two *parents* are selected from the population using *tournament selection*. This method chooses a number of individuals of tournament size (*TourSize*) at random, the individual with the best fitness wins the tournament and is selected as a parent. In the case of a tie between individuals in a tournament, the first individual is assigned as a parent. A randomly chosen crossover operator is then applied to the two parents in order to create an offspring solution (Child). Although there are many crossover operators which create two offspring in the scientific literature, within HyFlex only one offspring is generated. The offspring then undergoes mutation and local search (hill-climbing) processes successively. At the end of the evolutionary cycle, the resultant solution replaces the worst individual in the current population. This evolutionary process continues until the termination criterion is satisfied.

[23] presented results over the six HyFlex problem domains available at that time, compared to another memetic algorithm variant and the existing hyper-heuristics from the literature. Although some success was demonstrated in a small number of problem domains, particularly PFS and TSP, the memetic algorithms were typically outperformed by single-point based selection hyper-heuristics. [22] extended the original SSMA work of [23], demonstrating that effective tuning of this method via Taguchi design of experiments was able to deliver significant performance gains for a large majority of the instances tested.

3.2. Taguchi design of experiments method

The Taguchi method is a well-known design of experiments technique to tune parameters with respect to a fractional factorial design. Taguchi proposed a special set of orthogonal arrays for experiments that consist of a subset of combinations of different parameters set to different values. These standard orthogonal arrays minimise the number of experiments required to understand the impact different parameters have on performance, based on the assumption that independent performance and pairwise interaction between different parameters are most important. In an orthogonal array, the columns correspond to the parameters (sometimes referred to as factors), the column entries represent the values of each parameter, and the rows show the parameter combinations to test [41]. Taguchi orthogonal arrays are balanced to ensure that all values of all parameters are considered equally. They are balanced in two ways: (1) there are an equal number of settings of each parameter in a column, (2) any two columns are balanced (pairwise orthogonal) so that there are an equal number of possible parameter combinations [42]. As a concrete example, given four parameters ($P1$ - $P4$) that are each able to take one of three values (0, 0.5, 1). Full enumeration of all combinations of parameters would require $3^4 = 81$ experiments to be completed. However, the use of an orthogonal array that is balanced as described above allows this to be reduced to only 9 experiments, as shown in Table 1.

In the combinatorial optimisation literature, [43] used the Taguchi method to tune genetic algorithm parameters, such as population size, crossover rate, mutation rate and stopping condition, to solve a job shop scheduling problem. [44] also considered a job shop scheduling problem variant, tuning the parameters of a differential evolution algorithm. [45] considered the fixed charge transportation problem, using Taguchi orthogonal arrays to tune the parameters of three metaheuristics, including well-known methods such as a genetic algorithm and simulated annealing. Different orthogonal arrays were used for each metaheuristic based on their required number of parameters and parameter values.

In our previous study [22], we tuned the parameters of an SSMA with the Taguchi method using a limited number of instances from several problem domains. The best parameter settings found were observed to generalise well to unseen instances. Unlike other studies in the literature, the parameters were tuned for cross-domain search over multiple problem domains rather than just for a single problem domain.

3.3. F-Race

F-Race [8] is an automated parameter tuning method for finding the best initial parameter settings for metaheuristics. An automated parameter tuning method finds one or more configurations (parameter settings for all parameters) which perform well on the given problem. This is an iterative process, at each iteration the candidate configurations are evaluated on a set of training instances and the evaluations are returned as input to the next iteration [46]. This iterative process continues until the configuration budget is reached. At the end of the search, the best configurations are returned by the method. F-Race follows a racing procedure using sequential statistical testing to find a configuration for metaheuristics that performs well. The idea behind F-Race is to discard statistically significantly worse performing configurations as soon as sufficient evidence is found, focusing on more promising configurations. F-Race finds a good configuration from a finite set of parameter combinations which are collected using full factorial design. In the case of continuous parameters, these need to be discretised, as F-Race requires discrete levels for each parameter to form a full factorial design.

The process starts by evaluating all possible configurations on r instances, where r represents the minimum number of instances considered before any elimination in order to gather enough information to make an informed decision. Friedman two-way analysis of variance by rank, also known as the Friedman test, is applied to assess whether there is any statistically significant differences between configurations. The null hypothesis of this test is ‘‘there are no differences between the con-

Table 1
Example of experiments required using Taguchi design of experiments with four parameters able to take three values (Taguchi L9 array)

Experiment Number	*P*1	*P*2	*P*3	*P*4
1	0	0	0	0
2	0	0.5	0.5	0.5
3	0	1	1	1
4	0.5	0	0.5	1
5	0.5	0.5	1	0
6	0.5	1	0	0.5
7	1	0	1	0.5
8	1	0.5	0	1
9	1	1	0.5	0

figurations". If the null hypothesis is rejected, pairwise comparisons are performed between the best ranked candidate configuration and each other configuration. All poor performing candidate configurations that are statistically significantly different to the best configuration are eliminated, with the remaining configurations passing to the next iteration. The process continues considering the next instance.

To explain this approach formally, assume that the race reaches iteration k with n remaining candidate configurations. The observed objective costs are stored in a $k * n$ array. According to the Friedman test, there are k blocks where each block represents the cost of each configuration on instance i_l at iteration k. In the Friedman test, instead of using the objective function cost values directly rankings are used. The costs are replaced with ranks within each block and the average rank is used in case of ties. R_{lj} corresponds to the rank of configuration c_j in block l and R_j is the sum of ranks for c_j. The Friedman test statistic (T) is shown in Eq. 1:

$$T = \frac{(n-1)\sum_{j=1}^{n}(R_j - (k(n+1)/2))^2}{\sum_{l=1}^{k}\sum_{j=1}^{n}R_{lj}^2 - (kn(n+1)^2)/4} \tag{1}$$

If the observed value of T is greater than the (1-α) quantile of the X^2 distribution, then we reject the null hypothesis and conclude that at least one configuration performs better than at least one other. In this case, pairwise comparisons between the best ranked parameter configurations from the Friedman test and each of the other remaining configurations are conducted. In F-Race, the following test is used as Friedman post hoc test:

$$\frac{|R_j - R_h|}{\sqrt{\frac{2k(1-\frac{T}{k(n-1)})(\sum_{l=1}^{k}\sum_{j=1}^{n}R_{lj}^2 - \frac{kn(n+1)^2}{4})}{(k-1)(n-1)}}} > t_{1-\alpha/2} \tag{2}$$

If the result of Eq. 2 is greater than the 1-$\alpha/2$ quantile of the Student's t-distribution, then we are confident to eliminate the worse performing configurations. All poor performing candidate configurations with a statistically significant difference are discarded, and the remaining configurations are retained for the next stage.

In the literature, F-Race has been used for offline parameter tuning when solving different types of combinatorial optimisation problems. The parameters of a Max–Min-Ant-System metaheuristic were tuned using F-Race on a set of instances of the traveling salesman problem by [8]. [47] used F-Race to find the best parameter configurations for iterated local search and simulated annealing applied to a timetabling problem. [48] tuned the relevant parameters of four algorithms using F-Race on instances of the vehicle routing problem. The common feature of these studies is that they all tuned the parameters of metaheuristics solving a single combinatorial optimisation problem. In this paper, we investigate the suitability and the application of F-Race for cross-domain search, tuning both the algorithmic and heuristic parameters of a high-level search method applied to a number of different problem domains.

4. Experimental design and setup

In this paper, we investigate the suitability of F-Race for cross-domain parameter tuning. This method has been used in many studies previously. However, its performance for cross-domain parameter tuning, in which the optimisation problems being solved are characteristically different, has not yet been tested. Previous work has shown that parameter tuning via the Taguchi method can enhance the performance of an SSMA in the context of cross-domain search significantly [22]. In this work, we test different strategies for the application of F-Race, to investigate how sensitive the method is to the instances being used for training and the order they appear, as well as the run time available for tuning. The performance of SSMAs

using the parameters found by F-Race, the Taguchi method [22] and an existing SSMA from the literature [23] are compared along with the competing algorithms from the original CHeSC competition.

We will investigate the tuning of two algorithm parameters and two heuristic parameters of SSMA. The algorithm parameters are the population size (PopSize) and tournament size for tournament selection (TourSize). The heuristic parameters are intensity of mutation and depth of search (IoM and DoS) in HyFlex. The parameter setting options (values or levels) used during the tuning experiments for IoM and DoS are 0.2, 0.4, 0.6, 0.8 and 1.0. The potential values for PopSize are 5, 10, 20, 40 and 80. For the TourSize parameter, the possible values are 2, 3, 4 and 5.

4.1. Training strategies for cross-domain parameter tuning with F-Race

In order to investigate the suitability of F-Race for cross-domain parameter tuning and the impact of implicit design decisions made when deploying it, a number of training strategies are tested and compared. These strategies are based on training using a different number of instances across a different number of domains, and different orderings of instances to be provided to F-Race. Of the four strategies tested, the first three use eight training instances from the four original 'public' problem domains of the CHeSC2011 competition (two each from SAT, BP, PS, PFS). The final strategy uses twelve instances from all six problem domains (additionally two each from TSP and VRP) of CHeSC2011. These training instances are the same instances which were used in our previous studies [49,22] in order to be able to make a direct comparison. The strategies used to test F-Race are as follows:

- **Default:** Add one instance from one of the four 'public' problem domains at each iteration, in the 'default' HyFlex domain ordering (SAT, BP, PS, PFS). This strategy acts as a control, enabling a better comparison to the CHeSC competition entrants, as only these domains were available to these methods in advance of the competition.
- **Grouped:** Add four instances as a group, one from each of the four problem domains at each iteration. This strategy aims to investigate whether grouping multiple instances from different domains can benefit the training process.
- **Reverse:** Add one instance from one of the four problem domains at each iteration, with the domains considered in a reverse order to the *Default* strategy. This strategy will demonstrate the impact of modifying the order in which problem domains are considered in the training process.
- **Extra:** Add one instance from one of the six CHeSC2011 problem domains (SAT, BP, PS, PFS, TSP, VRP) at each iteration. This strategy will show whether adding additional information in the form of other problem domains will impact the performance of the training method.

In the experiments, F-Race starts with 500 parameter configurations (combinations) which are created by full factorial design, and eliminates statistically significantly worse performing configurations at each iteration. Before the first elimination, all training instances are evaluated, with one run performed for each configuration on each instance. For example, if the number of training instances is eight, then the number of initial iterations without elimination of candidate configurations (r) is also eight. Once every configuration has been evaluated, F-Race performs the Friedman test. If there is no evidence of a statistically significant difference, then an instance in the queue with a different seed is added to the test. If the null hypothesis is rejected and there is evidence of statistically significant difference, the configurations which are statistically significantly worse are eliminated. F-Race continues with the remaining candidate configurations considering another instance. Here, 'another instance' refers to solving the instance with a different random seed. The configuration budget of F-Race is determined as 31 iterations for each of the training instances. This means that if the number of training instances is eight, then the total budget for F-Race will be 31 * 8 = 248 iterations. 31 runs is the standard used for comparison in the literature for this benchmark. When the configuration budget is reached, F-Race reports the remaining configurations and their mean ranks over all the instances. The one with the best rank is chosen as the best configuration.

4.2. Parameter tuning with F-Race using different computational budgets

Methods in the literature performing cross-domain search using the HyFlex framework typically operate with a nominal run time budget of 10 min. As the full factorial design gives 500 possible parameter configurations, using the full time budget of ten minutes would require 500 * 10 = 5000 min to execute. Without applying any tests and reducing the number of candidate configurations, for eight training instances, F-Race would perform eight iterations, requiring 5000 * 8 = 40000 min computational time before any elimination takes place. As a point of reference, the total time used by [22] for the Taguchi method was 62000 min, and was reduced to 9300 min by [49]. Given that it could take F-Race a significant number of repetitions before an elite set of non-dominated configurations are identified, the run time required is potentially extremely large. Hence, there is a need to try to reduce the computational effort required to obtain useful parameter settings. [49] demonstrated that using the Taguchi method, the same best configuration could be found by reducing the run time for execution of each configuration to 1 min. Here, we will also reduce the run time budget for F-Race significantly, presenting experiments using a run time budget of 1 and 2 min per configuration to investigate how the results change with different levels of computational effort. We will perform these experiments for each of the four training strategies of F-Race described above, analysing the behaviour and comparing the performance of different strategies with different budgets. To facilitate comparisons across different hardware when using HyFlex, a benchmarking tool was provided by the original competition

organisers. According to the tool, our machine is allowed a budget of 41.5 and 83 nominal seconds run time for the execution of each instance when training, equivalent to 1 min and 2 min respectively on the competition machines. For testing, 415 s on our machine is equivalent to 10 nominal 'CHeSC' minutes.

5. Experimental results

This section presents the results and analysis of our experiments using F-Race and the HyFlex framework for cross-domain search. Section 5.1 and 5.2 present the experiments for F-Race with 1 min and 2 min budgets respectively, discussing the best configurations found before comparing to the existing methods from the literature [23,22]. Section 5.3 compares the best configurations found in the previous subsections using 1 and 2 min budgets to one another, before Section 5.4 compares them to the selection hyper-heuristics from the literature submitted to the original CHeSC competition.

5.1. F-Race with 1 min run time

The first set of experiments uses F-Race with a 1 min time budget per instance. The list of the remaining configurations at the end of a run for each of the four training strategies are given in Table 2 with their ranks provided in parentheses. The top three configurations for each strategy are highlighted bold. Starting from the original 500 configurations, the progress of the total remaining configurations over time for each of these strategies is provided in Fig. 1. Among the remaining parameter configurations, the best candidate and the total time that F-Race used to find this result are as follows:

- **Default:** Fourteen configurations remained when the configuration budget was reached. Conf410 has the best mean rank, thus it is considered the best configuration. The total time required for F-Race to find this result is 12338 min.
- **Grouped:** Twelve configurations remained when the configuration budget was reached. Conf410 has the best mean rank, thus it is considered the best configuration. The total time required for F-Race to find this result is 13296 min.
- **Reverse:** Nine configurations remained when the configuration budget was reached. Conf410 has the best mean rank, thus it is considered the best configuration. The total time required for F-Race to find this result is 9359 min.
- **Extra:** Eighteen configurations remained when the configuration budget was reached. Conf410 has the best mean rank, thus it is considered the best configuration. The total time required for F-Race to find this result is 17690 min.

The results show that all of the strategies found the same best configuration: Conf410. The parameter values of Conf410 are as follows: population size 80, tournament size 2, intensity of mutation 0.4 and depth of search 1.0. The order of instances or adding instances as a group did not have a significant effect on the final results with this run time budget. Almost all of the remaining configurations have a population size of 80. In addition, the DoS value for all of the remaining configurations is either 1.0 or 0.8, with 1.0 occurring in 43 out of the 53 remaining configurations across all four strategies. For the TourSize and IoM parameters, no dominant value was found. As can be seen from the progress plots of the number of remaining configurations in Fig. 1, more than half of the configurations are eliminated very quickly. After a few eliminations, it becomes more difficult to find a statistically significant difference between the remaining configurations. The *Extra* strategy, which uses a larger number of instances for training (twelve instances from six domains, as opposed to eight from four for the other three strategies), eliminated the largest number of configurations at the first elimination step (after the first r iterations) compared to the other strategies. Despite this, more configurations remained at the end, indicating that the increased level of information provided by the additional instances and problem domains made it more difficult to distinguish between the performance of parameter configurations.

5.1.1. Performance comparison of SSMA-Ozcan2013 [23] and SSMA tuned by F-Race using 1 min run time

The base SSMA that we are tuning with F-Race is the same algorithm which was proposed by [23], referred to herein as SSMA-Ozcan2013. The parameters of SSMA-Ozcan2013 were not systematically tuned, offering poor results when compared to the CHeSC2011 participants. In this section, SSMA-Ozcan2013 and the best configuration SSMA found by F-Race with a 1 min per configuration budget (denoted as SSMA-Conf410) are compared. SSMA-Conf410 is evaluated on the same thirty instances from six problem domains as [23], with 31 runs performed for each instance, and a time limit of 10 nominal minutes as defined by the CHeSC competition. These thirty instances include the eight instances used for training in the *Default*, *Grouped* and *Reverse* strategies, and the twelve instances used for training by the *Extra* strategy.

The performance comparison between SSMA-Conf410 and SSMA-Ozcan2013 using a one-tail Wilcoxon signed ranked test is reported in Table 3, along with the mean, median and best results found by each approach over 31 runs for each of the thirty instances. The results of the statistical tests are given in the 'vs.' column of the table. '$\geqslant$' indicates that the result on the left is better than the result on the right on average, with '$>$' denoting that this difference is statistically significant within a 95% confidence interval. Similarly, '$<$' and '$\leqslant$' denote that the result on the right is better on average with and without a statistically significant difference. Later comparisons using Wilcoxon signed-rank test calculations also include cases where the sum of positive ranks and the sum of negative ranks are exactly the same, indicated by a '='.

SSMA-Conf410 outperforms SSMA-Ozcan2013 with a statistically significant difference on 17 instances, with the opposite true for 7 instances. There is no statistically significant difference between the two algorithms in the other 6 instances.

Table 2
Remaining configurations for each strategy with 1 min run time. The top three configurations for each strategy are highlighted bold, with their ranks provided in parentheses.

Config No.	PopSize	TourSize	IoM	DoS	Default	Grouped	Reverse	Extra
Conf305	40	2	0.2	1				(13)
Conf404	80	2	0.2	0.8	(13)	(11)		(18)
Conf405	80	2	0.2	1	(6)	(6)	(5)	(8)
Conf409	80	2	0.4	0.8	(12)			(17)
Conf410	80	2	0.4	1	**(1)**	**(1)**	**(1)**	**(1)**
Conf414	80	2	0.6	0.8				(15)
Conf415	80	2	0.6	1	(8)	(8)		
Conf420	80	2	0.8	1	(11)			(16)
Conf425	80	2	1	1	(10)		(8)	(10)
Conf430	80	3	0.2	1	**(2)**	**(2)**		**(3)**
Conf435	80	3	0.4	1	(5)	**(3)**	**(2)**	(5)
Conf440	80	3	0.6	1	**(3)**	(4)	**(3)**	**(2)**
Conf445	80	3	0.8	1			(6)	
Conf450	80	3	1	1	(9)	(9)	(7)	(6)
Conf454	80	4	0.2	0.8	(14)	(12)		(14)
Conf455	80	4	0.2	1	(4)	(5)	(4)	(4)
Conf460	80	4	0.4	1	(7)	(7)		(7)
Conf464	80	4	0.6	0.8				(11)
Conf465	80	4	0.6	1			(9)	(12)
Conf470	80	4	0.8	1		(10)		(9)

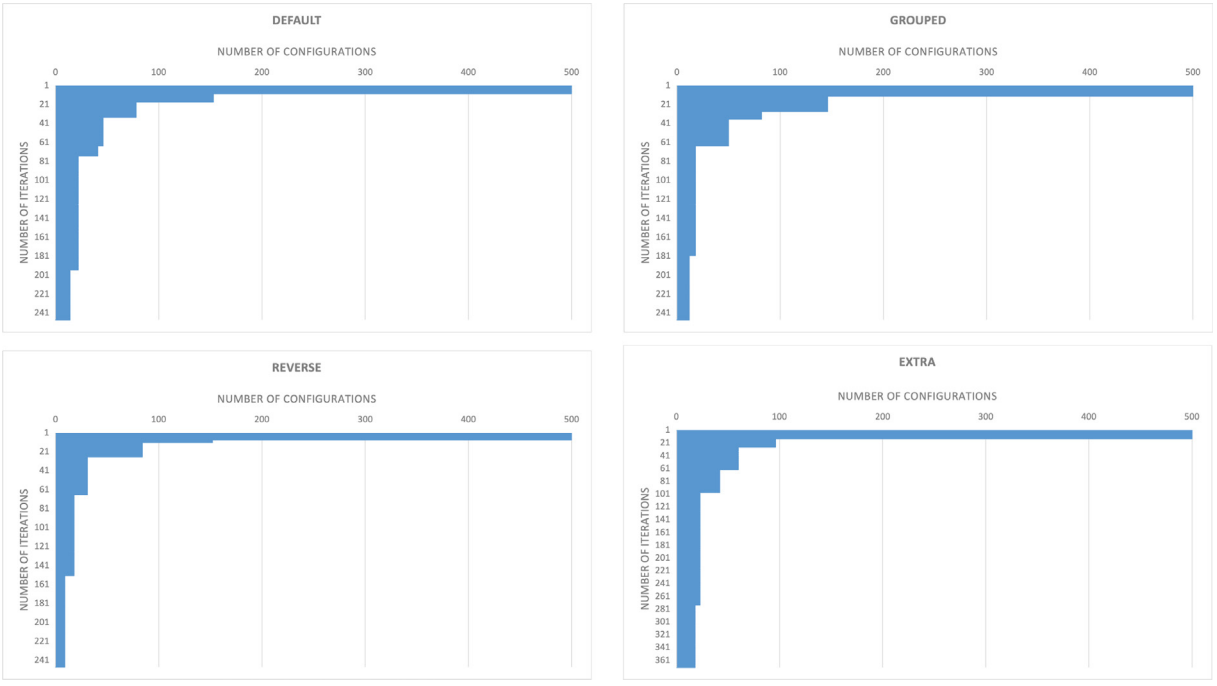

Fig. 1. Progress of the number of remaining configurations over iterations for each strategy with 1 min run time.

SSMA-Conf410 is clearly more successful on problem instances in the SAT, BP and PS domains. The results show that overall, the configuration found by F-Race outperforms SSMA-Ozcan2013, providing a parameter combination which performs well for cross-domain search with a relatively low computational budget for the training runs.

5.1.2. Comparison of SSMAs tuned by Taguchi and F-Race using 1 min run time

This section compares the performance of two tuning methods for cross domain search. The best parameter setting obtained by [22] using Taguchi was 0.2 for IoM, 1.0 for DoS, 5 for PopSize and 5 for TourSize, and is referred to as SSMA-Taguchi in the subsequent sections. It was evaluated over problem instances taken from nine different problem domains, both the original six domains from the CHeSC competition and the additional three domains introduced by [25]. To be able to investigate and compare the cross-domain performances of these two methods, SSMA-Conf410 is used to solve all 45

Table 3
Performance comparison of SSMA-Conf410 and SSMA-Ozcan2013 based on mean, median and best fitness values obtained from 31 runs for each instance of CHeSC2011.

		SSMA-Conf410				SSMA-Ozcan2013		
PD	ID	mean	median	best	vs.	mean	median	best
SAT	1	**9.903**	**9**	**5**	>	21.161	22	8
	2	**33.545**	**40**	**12**	>	52.484	53	37
	3	**16.387**	**11**	**4**	>	35	37	10
	4	**16.581**	**17**	**8**	>	27.742	27	16
	5	**13.774**	**14**	**11**	>	19.194	18	14
BP	1	**0.037**	**0.036**	**0.030**	>	0.083	0.082	0.075
	2	**0.007**	**0.008**	**0.007**	>	0.015	0.013	0.012
	3	**0.008**	**0.008**	**0.006**	>	0.022	0.022	0.019
	4	**0.109**	**0.109**	**0.108**	>	0.111	0.111	0.110
	5	**0.014**	**0.014**	**0.012**	>	0.043	0.041	0.036
PS	1	**31.290**	**32**	**27**	>	51.129	50	37
	2	**10396.387**	**10338**	**9991**	>	72015.613	70477	52056
	3	**3275.968**	**3272**	**3215**	>	13203.710	11859	5581
	4	**1816.226**	**1813**	**1557**	>	2942.419	2655	1820
	5	**357.936**	**360**	**321**	>	435.645	431	385
PFS	1	6264.742	6264	6241	<	**6257.806**	**6258**	**6231**
	2	26907.548	26908	26867	<	**26884.935**	**26884**	**26813**
	3	**6349.903**	**6357**	6319	⩾	6351.841	6363	**6318**
	4	11458.258	11458	11419	<	**11441.806**	**11441**	**11410**
	5	26705.452	26707	26649	⩽	**26699.226**	**26703**	**26626**
TSP	1	**48203.808**	**48194.920**	**48194.920**	⩾	48227.747	**48194.920**	**48194.920**
	2	**21153257.9**	**21159375.5**	21050272.3	⩾	21155458.2	21160876	**20969186**
	3	6827.100	6826.495	6810.696	⩽	**6825.522**	**6825.663**	**6800.708**
	4	68546.171	68639.410	67991.239	<	**68123.369**	**68059.971**	**6800.708**
	5	**53711.126**	**53702.434**	53192.830	⩾	53810.138	53748.537	**52685.992**
VRP	1	93723.112	93371.914	90116.035	<	**71768.053**	**71480.081**	**67820.589**
	2	**13293.735**	**13364.073**	**12308.190**	>	14324.522	14411.658	13358.612
	3	209689.948	209215.368	196823.398	<	**176206.081**	**177131.584**	**167704.513**
	4	**20727.215**	**20658.282**	**20653.158**	>	21647.018	21675.195	20678.097
	5	172492.221	172285.478	166707.184	<	**152642.040**	**152829.839**	**149032.552**

instances from the same nine HyFlex problem domains. As both SSMA-Taguchi and F-Race (with the *Default*, *Grouped* and *Reverse* strategies) were trained on the same instances from four of the problem domains, and then tested on the same instances from nine of the problem domains, we can compare these directly. Table 4 provides a comparison of the performance of SSMA-Taguchi and SSMA-Conf410 over 31 runs of each instance.

Based on these results, we see that SSMA-Taguchi outperforms SSMA-Conf410 with a statistically significant difference on 25 of the 45 instances tested. SSMA-Conf410 outperforms SSMA-Taguchi with a statistically significant difference for 14 instances, while there are 6 instances where no statistically significant difference is found. SSMA-Taguchi performs well on all instances of the PS and PFS domains; on the other hand, SSMA-Conf410 performs well on all BP instances. SSMA-Taguchi also performs better for most of the instances of the SAT and TSP problem domains. For the remaining domains, it is more difficult to draw general conclusions about the better approach, since there are instances for both of the configurations where they perform best. Overall, we conclude that SSMA-Taguchi outperforms SSMA-Conf410 over these nine problem domains. Although it is difficult to compare the overall computational effort spent training between these two approaches, it is worth highlighting that although the same set of training instances were used, each configuration was tested with a longer run time of 10 min. It might be that the 1 min computational budget is insufficient for F-Race to effectively identify a good parameter set for the longer testing runs.

5.2. F-Race with 2 min run time

In this subsection we extend the run time available when training F-Race to 2 min per problem instance, presenting results for the same four training strategies as before. The list of configurations remaining at the end of each application of F-Race for the four training strategies are given in Table 5, with the top three configurations (where applicable) for each strategy highlighted bold. The relative ranking of each configuration is provided in parenthesis. The progress of the number of remaining configurations over iterations for each of the strategies is provided in Fig. 2. The results of these four strategies are as follows:

- **Default:** Four configurations remained when the budget was exhausted. Conf424 has the best ranking among the remaining configurations. The total time that F-Race used to find this result is 43838 min.

Table 4
Performance comparison of SSMA-Conf410 and SSMA-Taguchi based on mean, median and best fitness values obtained from 31 runs for each instance of CHeSC2011 and five instances each from three additional HyFlex problem domains.

		SSMA-Conf410				SSMA-Taguchi		
PD	ID	mean	median	best	vs.	mean	median	best
SAT	1	9.903	**9**	5	⩽	**9.484**	**9**	**3**
	2	33.545	40	12	⩽	**30.065**	**36**	9
	3	16.387	11	4	⩽	**15.387**	**10**	**3**
	4	16.581	17	**8**	<	**13.613**	**13**	9
	5	13.774	14	11	<	**12.742**	**13**	**9**
BP	1	**0.037**	**0.036**	**0.030**	>	0.063	0.063	0.058
	2	**0.007**	**0.008**	**0.007**	>	0.011	0.012	0.008
	3	**0.008**	**0.008**	**0.006**	>	0.034	0.034	0.029
	4	**0.109**	**0.109**	**0.108**	>	0.11	0.11	0.11
	5	**0.014**	**0.014**	**0.012**	>	0.061	0.06	0.054
PS	1	31.29	32	27	<	**21.548**	**21**	**16**
	2	10396.387	10338	9991	<	**9705.129**	**9677**	**9477**
	3	3275.968	3272	3215	<	**3211.452**	**3219**	**3146**
	4	1816.226	1813	1557	<	**1596.871**	**1589**	**1344**
	5	357.936	360	321	<	**318.065**	**315**	**290**
PFS	1	6264.742	6264	6241	<	**6249.581**	**6251**	**6219**
	2	26907.548	26908	26867	<	**26812.452**	**26811**	**26754**
	3	6349.903	6357	6319	<	**6336.387**	**6333**	**6303**
	4	11458.258	11458	11419	<	**11376.613**	**11375**	**11333**
	5	26705.452	26707	26649	<	**26632.548**	**26640**	**26515**
TSP	1	**48203.81**	**48194.92**	**48194.92**	⩾	48221.583	**48194.92**	**48194.92**
	2	21153257.9	21159375.5	21050272.3	<	**20912006.4**	**20885233**	**20789117**
	3	6827.1	6826.495	6810.696	<	**6811.145**	**6811.518**	**6799.111**
	4	68546.171	68639.41	67991.239	<	**67029.810**	**67043.255**	**66518.735**
	5	53711.126	53702.434	53192.83	<	**53503.576**	**53457.422**	**52247.568**
VRP	1	93723.112	93371.914	90116.035	<	71946.305	**70776.497**	**65967.938**
	2	**13293.735**	**13364.073**	**12308.190**	>	13826.295	13384.024	13328.791
	3	209689.948	209215.368	196823.398	<	**147512.686**	**148001.434**	**143921.208**
	4	**20727.215**	**20658.282**	**20653.158**	>	21275.493	21648.051	20654.219
	5	172492.221	172285.478	166707.184	<	**147372.783**	**147228.056**	**145266.409**
KP	1	**−1250624.194**	**−1250591**	**−1255807**	>	-1218268.419	-1218057	-1238755
	2	-431338.419	-431338	-431347	<	**−431357.581**	**−431357**	**−431363**
	3	**−4303821.484**	**−4305327**	**−4321742**	>	-4259007.355	-4259569	-4272767
	4	**−1577174.645**	**−1577175**	**−1577175**	>	-1574342.419	-1572999	-1577175
	5	-1467361.903	-1467361	-1467381	<	**−1467381.355**	**−1467362**	**−1467429**
CUT	1	-273265650.7	-273353749	-274962313	<	**−273724297.5**	**−274024670**	**−276334160**
	2	**−3024.452**	**−3022**	-3037	>	-3018.323	-3018	-3040
	3	**−13128.742**	**−13129**	-13201	⩾	-13127.226	-13128	**−13206**
	4	-9802.161	-9805	-9860	<	**−9962.935**	**−9965**	**−10010**
	5	-2794.452	-2790	-2844	<	**−2819.613**	**−2822**	**−2860**
QAP	1	**154141.419**	**154136**	153992	>	154500.194	154472	**153978**
	2	**149884.968**	**149880**	149756	>	150315.677	150330	**149738**
	3	**1186951649.4**	**1186634168**	1185996137	⩽	1190423963	1189780533	1185996137
	4	44871905.7	44871022	44858486	<	**44852272.8**	**44853276**	**44826672**
	5	**273328**	**273332**	**273140**	>	273691.677	273726	273220

- **Grouped:** Only two configurations remained when the budget was exhausted. The IoM and DoS values of these configurations were the same, while the other two parameters had different settings. Among these two configurations, Conf424 has the better rank. The total time that F-Race used to find this result is 35392 min.
- **Reverse:** Sixteen configurations remained when the budget was exhausted. Conf30 has the best ranking among the remaining configurations. The total time that F-Race used to find this result is 54354 min.
- **Extra:** Only two configurations remained when the budget was exhausted. Among these two configurations, Conf424 has the better rank. The total time that F-Race used to find this result is 60138 min.

Three of the strategies found Conf424 to be the best setting while one strategy found a different configuration (Conf30). The parameter values of Conf424 are 80 for population size, 2 for tournament size, 1.0 for intensity of mutation and 0.8 for depth of search. Conf30 has the following parameter settings: population size of 5, tournament size of 3, intensity of mutation of 0.2 and depth of search of 1.0. The parameter values of Conf30 are very similar to the parameter values found by the Taguchi method, only the TourSize parameter value is different, while both of these configurations have the same PopSize, IoM and DoS parameter settings.

As can be seen in Table 5, more configurations remained at the end of the search when using the *Reverse* strategy, indicating that this strategy finds fewer significant differences between configurations. Fig. 2 shows that the *Reverse* strategy

Table 5
Remaining configurations for each strategy with 2 min run time. The top three configurations for each strategy are highlighted bold, with their ranks provided in parentheses.

Config No.	PopSize	TourSize	IoM	DoS	Default	Grouped	Reverse	Extra
Conf3	5	2	0.2	0.6	(4)			
Conf5	5	2	0.2	1	**(2)**			
Conf27	5	3	0.2	0.4			(11)	
Conf28	5	3	0.2	0.6			(5)	
Conf30	5	3	0.2	1			**(1)**	
Conf33	5	3	0.4	0.6			(8)	
Conf55	5	4	0.2	1			(7)	
Conf58	5	4	0.4	0.6			(9)	
Conf60	5	4	0.4	1			(4)	**(2)**
Conf399	40	5	1	0.8		**(2)**		
Conf409	80	2	0.4	0.8			(14)	
Conf410	80	2	0.4	1			**(2)**	
Conf415	80	2	0.6	1	**(3)**		(9)	
Conf420	80	2	0.8	1			**(3)**	
Conf424	80	2	1	0.8	**(1)**	**(1)**		**(1)**
Conf425	80	2	1	1			(6)	
Conf430	80	3	0.2	1			(10)	
Conf435	80	3	0.4	1			(15)	
Conf440	80	3	0.6	1			(12)	
Conf445	80	3	0.8	1			(13)	

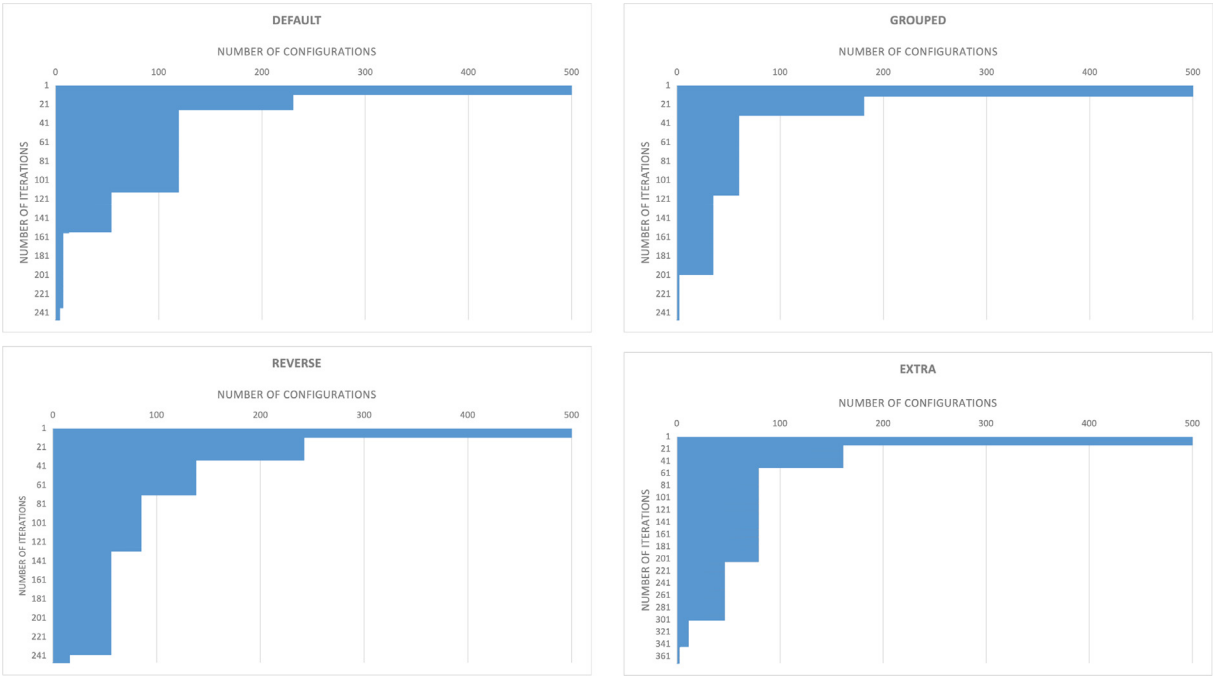

Fig. 2. Progress of the number of remaining configurations over iterations for each strategy with 2 min run time.

eliminated fewer configurations at the first elimination stage and retained more configurations during the process, resulting in a much longer time to execute compared to the *Default* and *Grouped* strategies. This analysis reveals that the order that instances are added to the training set has an effect on the final result of F-Race. Even though F-Race used the same instances in training for the *Default* and *Reverse* strategies, the remaining configurations are quite different. The best configuration found by the *Default* strategy is not in the remaining configurations found by the *Reverse* strategy at all.

In light of this, we have run additional experiments using a *Random* ordering of problem domains at each cycle. This can be seen as an alternative to reduce the effort of setting and testing different orders. For the 2 min experiments, the best configuration found is Conf425[1]. We note that a very similar configuration, Conf424, was found previously by three of the other

[1] For 1 min experiments, there is no difference in the best configuration found. The *Random* strategy finds the same configuration (Conf410) as the four existing training strategies.

strategies. The only difference between Conf424 and 425 is their DoS values. DoS is 0.8 for Conf424 while DoS is 1.0 for Conf425. Regarding this particular parameter, our previous work [49] found that DoS did not tend to have a significant effect on performance, so would not expect much variation from the results provided between these two configurations.

The effect of increasing the number of training instances does not seem to have a large impact on the final configurations found, the *Extra* strategy finds the same best configuration as the *Default* strategy. Compared to the results in Table 2 from a 1 min run time, there is a much greater diversity to the parameter settings found in the remaining configurations. Whereas almost every configuration found by all four training strategies had a PopSize of 80 and a high value for DoS, we observe a much greater variety of values for these parameters when allowing 2 min run time per configuration. Providing a longer run time allows the strength of a wider variety of configurations to emerge, suggesting that in the case of the shorter run times the configurations found may have been somewhat overfitted.

5.2.1. Performance comparison of SSMA-Ozcan2013 and SSMA tuned by F-Race using 2 min run time

The SSMAs with the two best configurations found by F-Race with 2 min run time are compared to the SSMA of [23] (SSMA-Ozcan2013), to assess whether F-Race managed to find better parameter combinations with a larger computational budget per configuration. Table 6 provides a direct comparison of the performance of SSMA-Ozcan2013 to SSMA-Conf424 and SSMA-Conf30, over the same thirty instances from six HyFlex problem domains as before, with 10 nominal minutes run time for each of the 31 runs of each instance.

The results of the paired one-tail Wilcoxon signed ranked tests indicate that both of the SSMA configurations found by F-Race outperform the original SSMA. The success of SSMA-Conf30 is more obvious, as in 28 out of 30 instances, SSMA-Conf30 performs better than SSMA-Ozcan2013 and in 26 of these cases, the difference is statistically significant. Comparing SSMA-Conf424 and the SSMA of [23], we can see that SSMA-Conf424 performs better in 18 instances, with a statistically significant difference in 17 of these cases. This demonstrates that F-Race has successfully found more effective parameter combinations for cross-domain search with a 2 min computational budget per configuration.

5.2.2. Performance comparison of the SSMAs tuned by Taguchi and F-Race using 2 min run time

Table 7 summarises the results and gives a performance comparison of the SSMAs found by F-Race with 2 min run time (SSMA-Conf424 and SSMA-Conf30) and SSMA-Taguchi, over 31 runs of each instance. Pairwise comparisons by Wilcoxon signed rank test were performed separately between SSMA-Taguchi and SSMA-Conf424, then SSMA-Taguchi and SSMA-Conf30.

SSMA-Taguchi performs better than SSMA-Conf424 in 29 out of 45 instances, in 25 of those cases the difference is statistically significant. SSMA-Conf424 performs better in 16 instances, while in 15 of those cases there are statistically significant differences. Overall, SSMA-Taguchi outperforms SSMA-Conf424. The performance comparison between SSMA-Taguchi and SSMA-Conf30 shows that for most of the instances, there is no statistically significant difference between the results. There are 27 instances in which both of the SSMAs perform similarly and there is no statistically significant difference between them. SSMA-Taguchi outperforms SSMA-Conf30 with a statistically significant difference in 7 out of 45 instances. On the other hand, SSMA-Conf30 performs significantly better than SSMA-Taguchi for 11 instances. Hence, we conclude that SSMA-Conf30 has a better performance across the 45 instances.

5.3. Performance comparison of the SSMAs tuned by F-Race using 1 min and 2 min run times

In the previous subsections, SSMA was tuned using two different run times with F-Race. In this section we compare the results of F-Race using 1 min and 2 min run times directly, to see if allowing a longer time for each run of each configuration has positive effect on performance. With 1 min run time, all four training strategies found SSMA-Conf410 as the best configuration. With 2 min run time, three of the strategies found SSMA-Conf424 as the best configuration, with SSMA-Conf30 found to be the best by the *Reverse* strategy. The SSMA algorithms with these three parameter configurations are tested on all 45 instances from the nine HyFlex problem domains, with 10 nominal minutes run time, in line with the standard termination criterion from the literature. Table 8 shows the results comparing Conf410 to Conf30 and Conf424 based on one-tail Wilcoxon signed ranked tests over 45 instances from all 9 HyFlex domains.

From these results, we see that SSMA-Conf30 performs significantly better than SSMA-Conf410 in 23 out of 45 instances. For the remaining instances, SSMA-Conf410 performs statistically significantly better in 12 cases, while there is no statistically significant difference in the other 10 instances. Comparing SSMA-Conf410 and SSMA-Conf424, we see that SSMA-Conf410 performs significantly better in 29 instances. SSMA-Conf424 performs significantly better than SSMA-Conf410 in only 3 instances. There are 13 cases in which the differences are not statistically significant between the two configurations. However, based on the mean results SSMA-Conf410 performs better on average in 8 out of these 13 instances.

These results show that the run time for each instance available when training is not the only criteria for finding a better configuration using F-Race. One of the tuned SSMA algorithms using 2 min run time (SSMA-Conf30) outperforms the tuned SSMA using 1 min run time (SSMA-Conf410). On the other hand, the other settings obtained by F-Race using 2 min run time (SSMA-Conf424) found worse overall results compared to the 1 min run time results. Providing additional run time to F-Race did not necessarily result a better solution in all cases. However, in the case of the *Reverse* strategy, the parameter configuration obtained with 2 min run times offers significantly improved performance. The capability of F-Race to find the best

Table 6
Performance comparison of SSMA-Ozcan2013, SSMA-Conf424 and SSMA-Conf30 based on mean, median and best fitness values obtained from 31 runs for each instance of CHeSC2011.

		SSMA-Conf424				SSMA-Ozcan2013				SSMA-Conf30		
PD	ID	mean	median	best	vs.	mean	median	best	vs.	mean	median	best
SAT	1	11.194	11	6	$>$	21.161	22	8	$<$	10.194	10	4
	2	37.065	44	10	$>$	52.484	53	37	$<$	31.129	34	8
	3	19.677	16	3	$>$	35.000	37	10	$<$	12.419	8	2
	4	16.645	17	10	$>$	27.742	27	16	$<$	12.613	12	7
	5	13.484	13	11	$>$	19.194	18	14	$<$	11.968	12	8
BP	1	0.042	0.043	0.023	$>$	0.083	0.082	0.075	$<$	0.057	0.056	0.051
	2	0.007	0.007	0.007	$>$	0.015	0.013	0.012	$<$	0.011	0.012	0.008
	3	0.010	0.010	0.007	$>$	0.022	0.022	0.019	$>$	0.029	0.029	0.023
	4	0.109	0.109	0.108	$>$	0.111	0.111	0.110	$<$	0.110	0.110	0.109
	5	0.018	0.018	0.015	$>$	0.043	0.041	0.036	$>$	0.051	0.050	0.040
PS	1	34.290	34	26	$>$	51.129	50	37	$<$	20.161	20	14
	2	10599.871	10503	10175	$>$	72015.613	70477	52056	$<$	9757.806	9758	9445
	3	3298.613	3289	3206	$>$	13203.710	11859	5581	$<$	3202.452	3181	3138
	4	1961.419	1974	1733	$>$	2942.419	2655	1820	$<$	1597.968	1570	1398
	5	363.323	365	335	$>$	435.645	431	385	$<$	321.452	320	290
PFS	1	6297.935	6299	6281	$<$	6257.806	6258	6231	$<$	6246.065	6244	6222
	2	26915.968	26914	26867	$<$	26884.935	26884	26813	$<$	26812.355	26809	26751
	3	6361.226	6366	6330	$<$	6351.871	6363	6318	$<$	6343.000	6350	6307
	4	11473.935	11476	11440	$<$	11441.806	11441	11410	$<$	11384.323	11387	11337
	5	26713.516	26712	26645	$<$	26699.226	26703	26626	$<$	26617.065	26617	26546
TSP	1	48198.977	48194.920	48194.920	$\geqslant$	48227.747	48194.9201	48194.9201	$\leqslant$	48215.658	48194.920	48194.920
	2	21172689.133	21178851.066	21050272.271	$\leqslant$	21155458.166	21160875.64	20969185.56	$<$	21011844.551	20980299.318	20855229.864
	3	6859.966	6860.228	6838.551	$<$	6825.552	6825.66255	6800.708271	$<$	6814.353	6813.875	6796.592
	4	68879.050	68887.686	68370.321	$<$	68123.369	68059.97098	67423.65527	$<$	67144.671	67245.273	66385.259
	5	54198.922	54164.596	53374.297	$<$	53810.138	53748.53746	52685.99238	$\leqslant$	53729.805	53639.446	52860.753
VRP	1	107293.333	107022.611	101111.735	$<$	71768.053	71480.081	67820.589	$<$	60508.492	60454.929	58778.751
	2	13366.053	13363.840	12313.472	$>$	14324.522	14411.658	13358.612	$<$	13790.548	13421.401	13315.537
	3	252604.413	251856.117	240231.929	$<$	176206.081	177131.584	167704.513	$<$	147668.620	148180.147	145149.551
	4	21088.581	20677.297	20655.207	$>$	21647.018	21675.195	20678.097	$<$	21144.088	20679.833	20653.760
	5	187091.686	187863.745	181752.128	$<$	152642.040	152829.839	149032.552	$<$	147114.519	147105.888	145792.856

Table 7
Performance comparison of SSMA-Taguchi, SSMA-Conf424 and SSMA-Conf30 based on mean, median and best fitness values obtained from 31 runs for each instance of CHeSC2011 and five instances each from three additional HyFlex domains.

		SSMA-Conf424				SSMA-Taguchi				SSMA-Conf30		
PD	ID	mean	median	best	vs.	mean	median	best	vs.	mean	median	best
SAT	1	11.194	11	6	$<$	9.484	9	3	$\geqslant$	10.194	10	4
	2	37.065	44	10	$<$	30.065	36	9	$\geqslant$	31.129	34	8
	3	19.677	16	3	$\leqslant$	15.387	10	3	$\leqslant$	12.419	8	2
	4	16.645	17	10	$<$	13.613	13	9	$\leqslant$	12.613	12	7
	5	13.484	13	11	$\leqslant$	12.742	13	9	$\leqslant$	11.968	12	8
BP	1	0.042	0.043	0.023	$>$	0.063	0.063	0.058	$<$	0.057	0.056	0.051
	2	0.007	0.007	0.007	$>$	0.011	0.012	0.008	$\leqslant$	0.011	0.012	0.008
	3	0.010	0.010	0.007	$>$	0.034	0.034	0.029	$<$	0.029	0.029	0.023
	4	0.109	0.109	0.108	$>$	0.110	0.110	0.110	$<$	0.110	0.110	0.109
	5	0.018	0.018	0.015	$>$	0.061	0.060	0.054	$<$	0.051	0.050	0.040
PS	1	34.290	34	26	$<$	21.548	21	16	$<$	20.161	20	14
	2	10599.871	10503	10175	$<$	9705.129	9677	9477	$>$	9757.806	9758	9445
	3	3298.613	3289	3206	$<$	3211.452	3219	3146	$\leqslant$	3202.452	3181	3138
	4	1961.419	1974	1733	$<$	1596.871	1589	1344	$=$	1597.968	1570	1398
	5	363.323	365	335	$<$	318.065	315	290	$\geqslant$	321.452	320	290
PFS	1	6297.935	6299	6281	$<$	6249.581	6251	6219	$\leqslant$	6246.065	6244	6222
	2	26915.968	26914	26867	$<$	26812.452	26811	26754	$\geqslant$	26812.355	26809	26751
	3	6361.226	6366	6330	$<$	6336.387	6333	6303	$\geqslant$	6343.000	6350	6307
	4	11473.935	11476	11440	$<$	11376.613	11375	11333	$>$	11384.323	11387	11337
	5	26713.516	26712	26645	$<$	26632.548	26640	26515	$<$	26617.065	26617	26546
TSP	1	48198.977	48194.920	48194.920	$>$	48221.583	48194.920	48194.920	$\leqslant$	48215.658	48194.920	48194.920
	2	21172689.133	21178851.066	21050272.271	$<$	20912006.397	20885233.010	20789116.983	$>$	21011844.551	20980299.318	20855229.864
	3	6859.966	6860.228	6838.551	$<$	6811.145	6811.518	6799.111	$\geqslant$	6814.353	6813.875	6796.592
	4	68879.050	68887.686	68370.321	$<$	67029.810	67043.255	66518.735	$>$	67144.671	67245.273	66385.259
	5	54198.922	54164.596	53374.297	$<$	53503.576	53457.422	52247.568	$>$	53729.805	53639.446	52860.753
VRP	1	107293.333	107022.611	101111.735	$<$	71946.305	70776.497	65967.938	$<$	60508.492	60454.929	58778.751
	2	13366.053	13363.840	12313.472	$>$	13826.295	13384.024	13328.791	$\leqslant$	13790.548	13421.401	13315.537
	3	252604.413	251856.117	240231.929	$<$	147512.686	148001.434	143921.208	$\geqslant$	147668.620	148180.147	145149.551
	4	21088.581	20677.297	20655.207	$\geqslant$	21275.493	21648.051	20654.219	$\leqslant$	21144.088	20679.833	20653.760
	5	187091.686	187863.745	181752.128	$<$	147372.783	147228.056	145266.409	$\leqslant$	147114.519	147105.888	145792.856
KP	1	-1249594.677	-1249575	-1253595	$>$	-1218268.419	-1218057	-1238755	$<$	-1247959.548	-1256601	-1262931
	2	-431312.355	-431312	-431323	$<$	-431357.581	-431357	-431363	$\geqslant$	-431356.871	-431356	-431362
	3	-4297035.516	-4297892	-4317606	$>$	-4259007.355	-4259569	-4272767	$<$	-4321918.419	-4336862	-4366624
	4	-1577150.323	-1577160	-1577168	$>$	-1574342.419	-1572999	-1577175	$\geqslant$	-1574070.387	-1572999	-1577175
	5	-1467355.935	-1467353	-1467378	$<$	-1467381.355	-1467362	-1467429	$\geqslant$	-1467371.226	-1467361	-1467437
CUT	1	-272265830.323	-272308054	-274362731	$<$	-273724297.548	-274024670	-276334160	$\geqslant$	-273575291.484	-273752723	-276546279
	2	-3021.097	-3020	-3034	$>$	-3018.323	-3018	-3040	$\geqslant$	-3017.710	-3018	-3030
	3	-13143.516	-13140	-13193	$>$	-13127.226	-13128	-13206	$\leqslant$	-13129.871	-13134	-13211
	4	-9826.290	-9841	-9902	$<$	-9962.935	-9965	-10010	$>$	-9950.645	-9952	-9994
	5	-2813.355	-2814	-2852	$\leqslant$	-2819.613	-2822	-2860	$>$	-2810.194	-2812	-2850
QAP	1	154324.645	154338	154038	$>$	154500.194	154472	153978	$\leqslant$	154373.419	154374	153898
	2	150069.935	150070	149886	$>$	150315.677	150330	149738	$<$	150143.935	150152	149732
	3	1188695259.710	1188572215	1186607150	$\leqslant$	1190423963.065	1189780533	1185996137	$\geqslant$	1188431602.871	1188636623	1185996137
	4	44871350.194	44872460	44837300	$<$	44852272.839	44853276	44826672	$\leqslant$	44850121.548	44850516	44813956
	5	273585.355	273572	273344	$>$	273691.677	273726	273220	$<$	273574.968	273526	273258

Table 8
Performance comparison of SSMA-Conf410 to SSMA-Conf30 and SSMA-Conf424 based on one-tail Wilcoxon signed ranked tests for 9 HyFlex domains.

	Conf410 vs Conf30					Conf410 vs Conf424				
Domain	>	⩾	=	⩽	<	>	⩾	=	⩽	<
SAT	0	1	0	2	2	0	4	0	1	0
BP	5	0	0	0	0	4	0	0	1	0
PS	0	0	0	0	5	4	1	0	0	0
PFS	0	0	0	1	4	4	1	0	0	0
TSP	0	1	0	1	3	4	0	0	1	0
VRP	2	0	0	0	3	4	1	0	0	0
KP	1	0	0	1	3	4	1	0	0	0
CUT	1	0	0	2	2	2	0	0	0	3
QAP	3	0	1	0	1	3	0	1	1	0
Total	12	2	1	7	23	29	8	1	4	3

parameter settings for cross-domain search depends on both the run time and the order in which instances are presented for training.

5.4. Performance comparison of suggested configurations to the CHeSC2011 competitors

In this section, SSMAs using the suggested parameter values from different parameter tuning methods and strategies are compared to the selection hyper-heuristics from the literature that competed in CHeSC2011, over the thirty instances from six problem domains used in the competition. We note here that there was some overlap between the training and testing sets used for CHeSC2011, with twelve of the thirty instances in the test set available to all competitors for training in advance of the competition. The SSMAs are denoted as follows:

- SSMA-Ozcan2013: The SSMA of [23] which was not systematically tuned using a parameter tuning method. The default values in HyFlex were used for the low-level heuristic parameters (0.2).
- SSMA-Taguchi: The SSMA of [22], where the parameters were tuned using the Taguchi method.
- SSMA-Conf410: The SSMA which uses the parameter settings found by F-Race with all four training strategies when the run time is 1 min.
- SSMA-Conf424: The SSMA with the parameter values found using F-Race with 2 min run time by three of the training strategies; *Default*, *Grouped* and *Extra*.
- SSMA-Conf30: The SSMA using the parameter values found by F-Race with the *Reverse* strategy and 2 min run time.

We utilise the 'Formula 1' scoring system from the original CHeSC2011 competition for comparison [50]. When using this system, the median objective values of the algorithm from 31 runs for each instance are collected, and the top eight algorithms awarded scores. Points of 10, 8, 6, 5, 4, 3, 2 and 1 respectively are given to each of the top approaches for each instance. If there is a tie, it is broken by taking the mean scores of the tied algorithms. At the end, the scores awarded to each instance are added up to calculate the final score of the algorithm. The results of the competing hyper-heuristics to CHeSC2011 have subsequently served as a benchmark for performance of future cross-domain approaches.

Table 9 provides a relative ranking of the five SSMAs and CHeSC2011 competitors.

From Table 9, we see that SSMA-Conf30 clearly performs the best of the five SMMAs, ranking 4^{th} overall among the methods compared. It is followed by SSMA-Taguchi which was ranked 5^{th}. SSMA-Conf410 and SSMA-Conf424 are ranked 14^{th} and 20^{th} respectively. SSMA-Ozcan2013 offers the poorest performance compared to other SSMAs, with a rank of 22^{nd}. Interestingly some configurations seem to perform better in some problem domains than others. Whereas SSMA-Conf410 and SSMA-Conf424 are performing well in BP, they score no points for VRP and PS, showing that they are not within the top eight methods for any instances in these domains. These two configurations have very large values for PopSize (80), suggesting that this parameter value may be important for good performance in this domain. On the other hand, SSMA-Conf30 has a much lower PopSize (5), performing poorly in BP, but offering more balanced performance across the problem domains tested. Interestingly, of the four problem domains that this configuration is performing well in, two of them (TSP and VRP) were not included in the instances that it was trained on. This suggests that the training method used is generalising well to unseen problem domains and instances.

Considering that all of the SSMAs have the same source code, it is obvious that the effect of parameter tuning methods is crucial to overall performance. Although a relatively small number of parameters are considered, there is a significant difference between the performance offered by methods tuned using different approaches. Despite this, there are still three methods from the competition that outperform even the strongest SSMA configuration, and a number of methods from the literature that claim to outperform all of these approaches [37,39,38,40]. This highlights the fact that performance cannot be improved indefinitely by parameter tuning alone. Although parameter tuning can help push a method towards the top end of its performance envelope, there is a natural limit to the improvement that can be obtained, depending on the under-

Table 9
Performance comparison of SSMAs found by different strategies (Default (D), Grouped (G), Reverse (R), Extra (E), 1 min, 2 min) and CHeSC2011 competitors across six HyFlex problem domains based on Formula1 scores.

Rank	Algorithm	SAT	BP	PS	PFS	TSP	VRP	Total
1	AdapHH	34.75	45	7	33	34.5	9	163.25
2	ML	14.5	9	25.5	36.5	8	20	113.5
3	VNS-TW	34.25	2	31	30.5	4.5	3	105.25
4	**SSMA-Conf30 [R; 2 min]**	**0**	**0**	**28**	**24.5**	**16.6**	**24**	**93.1**
5	**SSMA-Taguchi**	**0**	**0**	**28**	**23**	**28.6**	**11**	**90.6**
6	PHUNTER	10.5	3	8	3.5	18.5	30	73.5
7	NAHH	14	19	0	19.5	10	5	67.5
8	HAHA	32.75	0	20	0	0	12	64.75
9	EPH	0	5	5	13.5	27.5	12	63
10	ISEA	6	28	13	0	8	3	58
11	KSATS-HH	24	8	4	0	0	19	55
12	HAEA	0.5	2	0	3	5	22	32.5
13	ACO-HH	0	19	0	5	7	0	31
14	**SSMA-Conf410 [D,G,R,E; 1 min]**	**0**	**17**	**0**	**0**	**7.6**	**0**	**24.6**
15	GenHive	0	11	3	3	2	4	23
16	AVEG-Nep	12	0	0	0	0	9	21
17	XCJ	5.5	11	0	0	0	4	20.5
18	SA-ILS	0.75	0	12.5	0	0	3	16.25
19	GISS	0.75	0	10	0	0	5	15.75
20	**SSMA-Conf424 [D,G,E; 2 min]**	**0**	**7**	**0**	**0**	**6.6**	**0**	**13.6**
21	DynILS	0	9	0	0	3	0	12
22	**SSMA [23]**	**0**	**0**	**0**	**0**	**6.6**	**0**	**6.6**
23	MCHH-S	4.75	0	0	0	0	0	4.75
24	SelfSearch	0	0	0	0	1	0	1
25	Ant-Q	0	0	0	0	0	0	0

lying structure of the method. In particular, SSMA seems to struggle in the SAT problem domain irrespective of the parameter configuration used. Moreover, there may also be significant room for improvement in the performance of other state-of-the-art methods in the literature, by conducting systematic parameter tuning such as that completed in this work.

6. Conclusion

In this paper, the cross-domain performance of an automated parameter tuning technique (F-Race) was investigated with a number of different strategies for training. These strategies differ in the number of instances used for training, the order in which the instances are presented to F-Race, and the time budget allowed for each run of a candidate configuration to solve each training instance. The high-level algorithm parameters of a steady-state memetic algorithm, and the low-level heuristic parameters for mutation and local search in a cross-domain environment (HyFlex), were tuned using F-Race with these strategies.

Parameter tuning with F-Race clearly offers improved performance, with all strategies outperforming the base method using arbitrarily chosen parameters over a cross-domain benchmark. However, different strategies provide different results, so an appropriate strategy must be chosen carefully. With a shorter time budget, the results show that all four training strategies identified the same best configuration of parameters. When the time for each run was increased, different training strategies found different configurations to be the best. Interestingly, but perhaps not surprisingly, the order in which training instances are presented to the tuning method is also important. Making a decision too early, using initial observations based on instances that are not representative of the test set, will lead to poor performance.

It is difficult to declare a clear winner between the F-Race method using different training strategies presented here and the Taguchi-based approach from the literature. The best configuration found by F-Race outperforms Taguchi, but F-Race is sensitive to the training budget and the order in which training instances are added. The Taguchi design of experiments approach to parameter tuning involves a fractional factorial design, and hence it can overlook some good parameter configurations, although the performance does seem to be more robust. Effectively this represents a trade-off between the sampling rate for training and the overall execution time, an issue which we will investigate further in our future work.

Typically, more than half of the possible configurations are eliminated in the first step, after each configuration has been evaluated once on each instance. Although this is done to reduce the overall computational overhead required for training, it is possible that some good configurations are lost early on. Improved performance might be possible by dedicating more time to the early phases, gathering more information from which to base the early elimination decisions by increasing the number of iterations that are completed before the first elimination decision is taken. Future work will also consider mixing tuning strategies, merging approaches based on full enumeration such as those presented here with the Taguchi method. The orthogonal array yielded by the Taguchi approach could be used to reduce the overall sample size, although again this

represents a trade-off as some good configurations could be lost. Indeed, the best configuration found here (SSMA-Conf30) is not in the Taguchi orthogonal array.

Another line of research worth pursuing is increasing the range of parameters tuned to include algorithm configuration decisions. Our current work only considers explicit algorithm parameters such as population size, however the scope could be extended to more implicit algorithm design decisions such as the parent selection mechanism or replacement strategy used in the memetic algorithm. The SSMA approach studied here has relatively few parameters to tune. However, methods such as Taguchi and F-Race don't necessarily scale well to a large number of parameters. F-Race in particular requires a full factorial design, which is difficult and time consuming to deploy to methods with many parameters. Investigating whether or not these methods are still effective as the number of parameters increases will provide insight into the wider applicability of these approaches.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] H.H. Hoos, T. Stützle, Stochastic local search: Foundations and applications, Elsevier, 2004.
[2] K. Sörensen, F.W. Glover, Metaheuristics, in: Encyclopedia of Operations Research and Management Science, Springer, 2013, pp. 960–970.
[3] K. Sörensen, M. Sevaux, F. Glover, A history of metaheuristics, Handbook of Heuristics (2018) 1–18.
[4] T. Stützle, Iterated local search for the quadratic assignment problem, European Journal of Operational Research 174 (3) (2006) 1519–1539.
[5] R. Masson, T. Vidal, J. Michallet, P.H.V. Penna, V. Petrucci, A. Subramanian, H. Dubedout, An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems, Expert Systems with Applications 40 (13) (2013) 5266–5275.
[6] I. Boussaïd, J. Lepagnot, P. Siarry, A survey on optimization metaheuristics, Information Sciences 237 (2013) 82–117.
[7] S.K. Smit, A.E. Eiben, Comparing parameter tuning methods for evolutionary algorithms, in: 2009 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2009, pp. 399–406.
[8] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, A racing algorithm for configuring metaheuristics, in: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, 11–18, 2002.
[9] N.R. Sabar, G. Kendall, Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems, Information Sciences 314 (2015) 225–239.
[10] M. El Yafrani, B. Ahiod, Efficiently solving the traveling thief problem using hill climbing and simulated annealing, Information Sciences 432 (2018) 231–244.
[11] N. Veček, M. Mernik, B. Filipič, M. Črepinšek, Parameter tuning with Chess Rating System (CRS-Tuning) for meta-heuristic algorithms, Information Sciences 372 (2016) 446–469.
[12] E.K. Burke, T. Curtois, G. Kendall, M. Hyde, G. Ochoa, J.A. Vazquez-Rodriguez, Towards the decathlon challenge of search heuristics, in: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, ACM, 2205–2208, 2009a.
[13] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: A survey of the state of the art, Journal of the Operational Research Society 64 (12) (2013) 1695–1724.
[14] G. Koulinas, L. Kotsikas, K. Anagnostopoulos, A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem, Information Sciences 277 (2014) 680–693.
[15] S. Asta, D. Karapetyan, A. Kheiri, E. Özcan, A.J. Parkes, Combining Monte-Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem, Information Sciences 373 (2016) 476–498.
[16] V. Pandiri, A. Singh, A hyper-heuristic based artificial bee colony algorithm for k-interconnected multi-depot multi-traveling salesman problem, Information Sciences 463 (2018) 261–281.
[17] K.Z. Zamli, F. Din, G. Kendall, B.S. Ahmed, An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t-way test suite generation, Information Sciences 399 (2017) 121–153.
[18] G. Ochoa, M. Hyde, T. Curtois, J.A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A.J. Parkes, S. Petrovic, et al, Hyflex: A benchmark framework for cross-domain heuristic search, in: European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, 2012, pp. 136–147.
[19] L. Di Gaspero, T. Urli, A reinforcement learning approach for the cross-domain heuristic search challenge, in: Proceedings of the 9th Metaheuristics International Conference (MIC 2011), Udine, Italy, 2011.
[20] F. Mascia, T. Stützle, A non-adaptive stochastic local search algorithm for the chesc 2011 competition, in: Learning and Intelligent Optimization, Springer, 2012, pp. 101–114.
[21] N.R. Sabar, M. Ayob, G. Kendall, R. Qu, Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems, IEEE Transactions on Evolutionary Computation 19 (3) (2014) 309–325.
[22] D.B. Gümüş, E. Özcan, J. Atkin, An investigation of tuning a memetic algorithm for cross-domain search, in: 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 135–142.
[23] E. Özcan, S. Asta, C. Altintas, Memetic Algorithms for Cross-domain Heuristic Search, in: Computational Intelligence (UKCI), 2013 13th UK Workshop on, IEEE, 175–182, 2013.
[24] E.K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J.A. Vázquez-Rodríguez, M. Gendreau, Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms, in: 2010 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2010, pp. 1–8.
[25] S. Adriaensen, G. Ochoa, A. Nowé, A benchmark set extension and comparative study for the hyflex framework, in: 2015 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 784–791.
[26] E.K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J.A. Vázquez-Rodríguez, Hyflex: A flexible framework for the design and analysis of hyper-heuristics, in: Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland, 2009, pp. 790–797.
[27] J.H. Drake, A. Kheiri, E. Özcan, E.K. Burke, Recent advances in selection hyper-heuristics, European Journal of Operational Research 285 (2) (2020) 405–428.
[28] P.-C. Hsiao, T.-C. Chiang, L.-C. Fu, A variable neighborhood search-based hyperheuristic for cross-domain optimization problems in CHeSC 2011 competition, in: Fifty-Third Conference of OR Society (OR53), Nottingham, UK, 2011.
[29] M. Mısır, K. Verbeeck, P. De Causmaecker, G.V. Berghe, An intelligent hyper-heuristic framework for chesc 2011, in: Learning and Intelligent Optimization, Springer, 2012, pp. 461–466.
[30] M. Larose, A hyper-heuristic for the chesc 2011, CHeSC2011 Competition.

[31] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.
[32] J.H. Drake, E. Özcan, E.K. Burke, An improved choice function heuristic selection for cross domain heuristic search, in: International Conference on Parallel Problem Solving from Nature, Springer, 2012, pp. 307–316.
[33] P. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach to scheduling a sales summit, in: International Conference on the Practice and Theory of Automated Timetabling, Springer, 2000, pp. 176–190.
[34] S. Asta, E. Özcan, A tensor-based selection hyper-heuristic for cross-domain heuristic search, Information Sciences 299 (2015) 412–432.
[35] S.S. Choong, L.-P. Wong, C.P. Lim, Automatic design of hyper-heuristic based on reinforcement learning, Information Sciences 436 (2018) 89–107.
[36] A. Kheiri, E. Keedwell, A sequence-based selection hyper-heuristic utilising a hidden Markov model, in: Proceedings of the 2015 annual conference on genetic and evolutionary computation, 417–424, 2015.
[37] S. Adriaensen, T. Brys, A. Nowé, Fair-share ILS: a simple state-of-the-art iterated local search hyperheuristic, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM, 1303–1310, 2014.
[38] S.A. Adubi, O.O. Oladipupo, O.O. Olugbara, Configuring the Perturbation Operations of an Iterated Local Search Algorithm for Cross-domain Search: A Probabilistic Learning Approach, in: 2021 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2021, pp. 1372–1379.
[39] A. Kheiri, E. Özcan, An iterated multi-stage selection hyper-heuristic, European Journal of Operational Research 250 (1) (2016) 77–90.
[40] F. Zhao, S. Di, J. Cao, J. Tang, et al, A novel cooperative multi-stage hyper-heuristic for combination optimization problems, Complex System Modeling and Simulation 1 (2) (2021) 91–108.
[41] R.N. Kacker, E.S. Lagergren, J.J. Filliben, Taguchi's orthogonal arrays are classical designs of experiments, Journal of Research of the National Institute of Standards and Technology 96 (5) (1991) 577.
[42] R. Tyasnurita, E. Özcan, R. John, Learning heuristic selection using a time delay neural network for open vehicle routing, in: IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 1474–1481.
[43] J.U. Sun, A. Taguchi, Approach to Parameter Setting in a Genetic Algorithm for General Job Shop Scheduling Problem, Industrial Engineering and Management Systems 6 (2) (2007) 119–124.
[44] P. Dempster, P. Li, J.H. Drake, Solving the distributed two machine flow-shop scheduling problem using differential evolution, in: International Conference on Swarm Intelligence, Springer, 2017, pp. 449–457.
[45] K. Yousefi, A.J. Afshari, M. Hajiaghaei-Keshteli, Solving the fixed charge transportation problem by new heuristic approach, Journal of Optimization in Industrial Engineering 12 (1) (2019) 41–52.
[46] T. Stützle, M. López-Ibáñez, Automated Design of Metaheuristic Algorithms, in: Handbook of Metaheuristics, Springer, 2019, pp. 541–579.
[47] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L.M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, et al., A comparison of the performance of different metaheuristics on the timetabling problem, in: International Conference on the Practice and Theory of Automated Timetabling, Springer, 329–351, 2002.
[48] P. Pellegrini, D. Favaretto, E. Moretti, Multiple ant colony optimization for a rich vehicle routing problem: a case study, in: International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, Springer, 2007, pp. 627–634.
[49] D.B. Gümüş, E. Özcan, J. Atkin, An analysis of the Taguchi method for tuning a memetic algorithm with reduced computational time budget, in: International Symposium on Computer and Information Sciences, Springer, 2016, pp. 12–20.
[50] E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, B. McCollum, G. Ochoa, A.J. Parkes, S. Petrovic, The cross-domain heuristic search challenge–an international research competition, in: International Conference on Learning and Intelligent Optimization, Springer, 2011, pp. 631–634.