On the Automatic Design of Multi-Objective Particle Swarm Optimizers: Experimentation and Analysis

Antonio J. Nebro[1,2*], Manuel López-Ibáñez[3], José García-Nieto[1,2] and Carlos A. Coello Coello[4,5]

[1*]ITIS Software, University of Málaga, Ada Byron Research Building, Málaga, 29071, Málaga, Spain.
[2]Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S. de Ingeniería Informática, Málaga, 29071, Málaga, Spain.
[3]Alliance Manchester Business School, University of Manchester, Booth Street West, Manchester, M15 6PB, UK.
[4]Evolutionary Computation Group, CINVESTAV-IPN, Av. IPN No. 2508, Ciudad de México, 07360, México.
[5]School of Engineering and Sciences, Tecnologico de Monterrey, Eugenio Garza Sada 2501, Monterrey, Nuevo León 64849, México.

*Corresponding author(s). E-mail(s): ajnebro@uma.es;
Contributing authors: manuel.lopez-ibanez@manchester.ac.uk; jnieto@uma.es; carlos.coellocoello@cinvestav.mx;

Abstract

Research in multi-objective particle swarm optimizers (MOPSOs) progresses by proposing one new MOPSO at a time. In spite of the commonalities among different MOPSOs, it is often unclear which algorithmic components are crucial for explaining the performance of a particular MOPSO design. Moreover, it is expected that different designs may perform best on different problem families and identifying a best overall MOPSO is a challenging task. We tackle this challenge here by: (1) proposing AutoMOPSO, a flexible algorithmic template for designing MOPSOs with a design space that can instantiate thousands of potential MOPSOs; and (2) searching for good-performing MOPSO designs given a family of training problems by means of an automatic configuration tool (irace). We apply this automatic design methodology to generate a MOPSO that significantly outperforms two state-of-the-art MOPSOs on four well-known bi-objective problem families. We also identify the key design choices and parameters of the winning MOPSO by means of ablation. AutoMOPSO is publicly available as part of the jMetal framework.

Keywords: Automatic algorithm design; multi-objective optimization; particle swarm optimization; benchmarking

1 Introduction

The optimization of problems composed of two or more conflicting objectives or functions using metaheuristics became a very active field of research since the early 2000s, when popular algorithms such as NSGA-II (Deb et al., 2002) and SPEA2 (Zitzler et al., 2002) were introduced and the book of (Deb, 2001) and the first edition of the book (Coello Coello et al., 2007) were published. Since then, a number of research lines have been deeply studied, including many-objective optimization (Li et al., 2015), dealing with expensive objective functions (Knowles, 2006; Chugh et al., 2018), and dynamic problems (Jiang et al., 2022), just to mention a few. As a consequence, a large number of multi-objective metaheuristics have been proposed.

The fact that the objectives are conflicting in multi-objective problems means that improving one objective leads to a worsening of the others. As a consequence, the optimum of these problems is usually not a single solution, but a set of trade-off solutions collectively known as Pareto set, whose image in the objective space is called Pareto front. Given a multi-objective problem, the main goal of using metaheuristics is to find an accurate approximation of its Pareto front, where accuracy is measured in terms of convergence, distribution and spread of the found solutions with respect to the Pareto front.

The effectiveness of metaheuristics (both single and multi-objective) depends to a large extent on the correct setting of their algorithmic parameters. The approach usually adopted is to manually adjust the parameters by conducting pilot tests, which is a tedious and not rigorous task. If the final user (i.e., the decision maker) is not an expert in metaheuristics, such user will not have a minimum background to attempt the trial-and-error strategy, so she/he will probably opt for using a widely known technique, such as NSGA-II, configured with default settings taken from the paper where the algorithm was originally presented.

In this context, a line that is gaining momentum is the automatic algorithm configuration and, as an extension, the automatic design of metaheuristics. While in the first case the goal is, given a set of problems used as *training set*, to find a particular parameter setting that solve them in an efficient way, the second case is a step forward in which not only the control parameters are considered, but also the components of the algorithm that can be combined to design a new ad-hoc algorithmic variant automatically.

In this paper, we are interested in the auto-design of multi-objective particle swarm optimizers (MOPSOs), which constitute a family of nature-inspired metaheuristics that have shown to be very effective to solve multi-objective problems (Durillo et al., 2009). The inspiration of particle swarm optimization (PSO) algorithms, proposed by (Kennedy and Eberhart, 1995) is the choreography of bird flocks when seeking for food. The implementation of the algorithm adopts a population of particles, called swarm, which are initialized by default with random solutions (i.e., random positions in the design space), and whose behavior is affected by either their best local (i.e., within a certain neighborhood) or the best global particle (also known as the leader). Over the generations, particles adapt their beliefs to the most successful solutions found in their environment.

The approach we consider here is the automatic design of MOPSOs, following a methodology already applied to the design of multi-objective ant colony optimization (López-Ibáñez and Stützle, 2012) and multi-objective evolutionary algorithms (Bezerra et al., 2016, 2020). The main ingredients of this methodology are: (1) a flexible design space that allows instantiating MOPSO algorithms by configuring the parameters of some software framework and (2) the application of automatic configuration tools to search for designs given a set of (training) problem instances and a unary quality metric, such as hypervolume (Zitzler and Thiele, 1998). Therefore, one of our contributions is a flexible design space for MOPSOs implemented in the jMetal framework (Durillo and Nebro, 2011). Our overall goal is to illustrate the automatic design methodology in the case of MOPSOs for several well-known problem families.

This paper is an extension of (Doblas et al., 2022), where we presented an approach based on combining the jMetal framework with the irace package such that, given a design space of MOPSO components, we could find the best configurations for a set of test suites. The study carried out showed that the found *AutoMOPSOs* outperformed reference algorithms including SMPSO (Nebro et al., 2009) and OMOPSO (Reyes Sierra and Coello Coello, 2005), when applied to the solution of three problem families (for a total of 21 problems). Here, we increase the set of benchmark problems, reduce the budget of the evaluations to make them more challenging, and conduct a more in-depth analysis.

Concretely, our aim is to answer the following research questions (RQs):

RQ1. Is it possible to find designs of MOPSO algorithms using automatic design which are able to outperform state-of-the-art algorithms such as SMPSO and OMOPSO?
RQ2. Is the auto-design process able to obtain a configuration that yields to accurate results when a benchmark of realistic problems is added to the study?
RQ3. What are the design parameters that have the strongest effect on the performance of MOPSO algorithms given the design space considered here?
RQ4. Can a configuration obtained for a given test suite lead the algorithm to avoid overffiting and allow a generalization for many other problems?

These questions are addressed in the discussion section in the light of the results obtained after performing a series of experiments. In this regard, a new set of experiments have been conducted in the current study on an extended benchmark containing different problem families, which indeed incorporate an enriched set of algorithmic components and parameters. All this allows us to carry out an in-depth analysis of the designs found using ablation analysis (Fawcett and Hoos, 2016) with the goal of determining which parameters have more influence in the MOPSO variants under consideration.

This paper is organized according to the following structure of contents. The next section contains a review of previous related work in order to provide the proper context to our proposal. Section 3 describes the methods and tools employed, where the design and development decisions taken to generate the AutoMOPSO approach are explained. Extended experiments and comparisons are provided in Section 4. In Section 5, a series of discussions are included with the aim of answering the research questions previously indicated. Finally, our conclusions and some possible future steps in this research line are described in Section 6.

2 Related Work

Automatic algorithm configuration (Birattari, 2009) is the idea of using optimization methods for automatically setting the parameter values of an algorithm to maximize a quality metric, such as expected solution cost, over a given set of problem instances. To approach this idea, automatic configuration methods, such as irace (López-Ibáñez et al., 2016), are emerging as black-box optimization methods able to handle integer, real and categorical parameters, which may be also conditional on the values of other parameters. In addition, such methods must handle the stochasticity of the algorithm being tuned and intelligently use the computational budget given by selecting how many and which problem instances are used for evaluating the performance of a candidate configuration.

Automatic design extends this idea (López-Ibáñez and Stützle, 2012; Khudabukhsh et al., 2016; Stützle and López-Ibáñez, 2019) by considering flexible algorithmic frameworks from which different algorithmic designs can be instantiated by appropriately setting parameter values. In automatic design the challenge is the definition of such frameworks, which may be flexible enough to define a large space of potentially useful algorithmic designs, while at the same time reducing as much as possible redundancies between alternative designs (Stützle and López-Ibáñez, 2019).

If we focus on PSO, Frankestein's PSO (Montes de Oca et al.,, 2009) is a proposal to combine different components of single-objective PSOs, but without using an automatic configuration approach. The PSO-X framework (Camacho-Villalón et al, 2021) applies automatic design to PSOs, but it is also focused on single-objective optimization algorithms. Thus, many design decisions that are only relevant for multi-objective problems (such as the role of archived particles in the velocity update) are not taken into account in these proposals.

To the best of our knowledge, (de Lima and Pozo, 2017) is the only work that has studied automatic design of MOPSO algorithms. In that work, the design space is given as a context-free grammar which is tuned for the DTLZ (Deb-Thiele-Laumanns-Zitzler) problem family (Deb et al., 2005) by using both grammatical evolution and irace. The automatically-designed MOPSOs are compared with respect to SMPSO. Although their methodology is similar to ours, we focus here on a larger design space, 25 parameters, while 10 were taken in (de Lima and Pozo, 2017), and on the analysis of the algorithmic components that contribute the most to the observed performance. In addition, we consider a larger set of problem families. In our work her, we focus on bi-objective problems and we have chosen four problem families with the goal of understanding which problem families lead to more general MOPSO designs when used as training sets. In particular, we focus on bi-objective problem families: the Zitzler-Deb-Thiele (ZDT) (Zitzler et al., 2000), Deb-Thiele-Laumanns-Zitzler (DTLZ) (Deb et al., 2005), Walking-Fish-Group (WFG) (Huband et al., 2006) and the RE (Tanabe and Ishibuchi, 2020) problem families. We also compare the

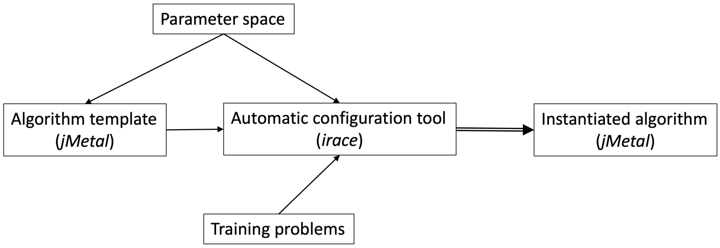

Fig. 1: Auto-design process scheme.

automatically-designed MOPSOs not only to SMPSO, but also to OMOPSO, since the latter sometimes outperforms the former.

Finally, as an additional contribution, we make available the resulting framework as part of jMetal, which will allow practitioners and researchers to apply automatic design and the analysis shown here to other problem scenarios.

3 Methods and Tools

The proposed process to carry out the automatic design of a metaheuristic is depicted in Fig. 1. The starting point is an algorithmic template defining the behavior of the baseline algorithm to be designed and a parameter space which defines its parameters and components, including their relationships, that can be combined in the template to produce a particular algorithm. Then, after selecting a number of optimization problems as the training set, an automatic configuration tool is executed to find a particular configuration that yields an instantiated algorithm able of optimize efficiently the problems in the training set. Depending on the diversity of the training problems and their representativity of a larger family of problems, the resulting algorithm may either be biased to them, leading to overfiting, or can be more generic and perform well for other similar problems.

When the template is tailored to a specific algorithm and the parameter space merely describes the usual (typically numerical) parameters of that algorithm, the above process corresponds to *automatic configuration*. An example is the work presented by (Nebro et al., 2019), where the specific algorithm is NSGA-II and the goal was to automatically configure its parameters, such as population size, mutation rate, etc. The concept of *automatic design* goes one step beyond and implies building a new meta-algorithm or a generic algorithmic template where some (high-level) parameters represent design choices that distinguish between different algorithms. Setting those high-level parameters appropriately allows instantiating specific algorithms. In addition, specific algorithms will have a number of low-level parameters that are sometimes shared among different potential designs, but are often disabled for specific high-level designs.

In this work, we start from a canonical MOPSO algorithm (described in Section 3.1) and, from it, we propose an algorithmic template, called AutoMOPSO, for the automatic design of multi-objective particle swarm optimizers, which is in turn detailed in Section 3.2. Some elements of this template can be chosen from a number of alternative options, including various numerical settings. Those configurable elements, together with their domains and various dependencies that exist among them, give rise to a parameter space (Section 3.3). By assigning values to the parameters of the AutoMOPSO template, we can instantiate and run a specific MOPSO.

We implemented AutoMOPSO in the jMetal framework, which is a library for multi-objective optimization with metaheuristics available in Java (Durillo and Nebro, 2011; Nebro et al., 2015). Given a string representing pairs of parameters and values, our implementation of AutoMOPSO creates the corresponding MOPSO algorithm. When running this MOPSO algorithm on a particular problem up to a given termination criterion, the result is the value of a unary indicator measuring the quality of the obtained Pareto front approximation.

To search in the parameter space of AutoMOPSO, the automatic configuration method adopted was irace (López-Ibáñez et al., 2016), which is an R package implementing an elitist iterated racing strategy for the automatic tuning of optimization algorithms. Given the parameter space and a set of training problems, irace calls the jMetal implementation of AutoMOPSO to generate specific MOPSO designs, then evaluates them on problems selected from the training set and captures the returned quality indicator values to compare alternative MOPSO designs.

3.1 Canonical MOPSO Algorithm

A generic PSO works by iteratively generating new particles' positions located in a given problem search space. The position $\mathbf{x}_i$ represents a set of Cartesian coordinates describing a point in solution space. Each of these new particles' positions are calculated at each iteration t as follows:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{1}$$

where $\mathbf{v}_i^{t+1}$ is the velocity vector of particle i. This velocity is updated using the next formula:

$$\mathbf{v}_i^{t+1} = \omega \cdot \mathbf{v}_i^t + U^t[0, \varphi_1] \cdot (\mathbf{p}_i^t - \mathbf{x}_i^t) + U^t[0, \varphi_2] \cdot (\mathbf{b}_i^t - \mathbf{x}_i^t) \tag{2}$$

where $\mathbf{p}_i^t$ is the personal best position that the particle i has ever stored, and $\mathbf{b}_i^t$ is the position found by the member of its neighborhood that has had the best performance so far (also known as leader or global best). Acceleration coefficients φ_1 and φ_2 control the relative effect of the personal best, and U^t is a diagonal matrix with elements distributed uniformly at random in the interval $[0, \varphi_i]$. Finally, ω is called the inertia weight and influences the trade-off between exploitation and exploration.

An alternative version of the velocity equation was proposed by Clerc and Kennedy (2002), where the constriction coefficient χ (which arises from the observation that the velocity of some particles keeps growing unless restrained in some way) is adopted instead of the inertia weight ω:

$$\mathbf{v}_i^{t+1} = \chi \cdot \left(\mathbf{v}_i^t + U^t[0, \varphi_1] \cdot (\mathbf{p}_i^t - \mathbf{x}_i^t) + U^t[0, \varphi_2] \cdot (\mathbf{b}_i^t - \mathbf{x}_i^t)\right) \tag{3}$$

where the constriction coefficient χ is calculated from the two acceleration coefficients φ_1 and φ_2 as:

$$\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \quad \text{where} \quad \varphi = \varphi_1 + \varphi_2 \quad \text{and} \quad \varphi > 4 \tag{4}$$

This constriction method results in convergence over time, and the amplitude of particles' oscillations decreases along the optimization process.

In order to adapt the canonical PSO to the multi-objective domain, we consider that the global best, instead of being a single particle, is implemented as a bounded archive containing the non-dominated solutions found so far. This means that a strategy for choosing the concrete global best particle must be adopted to apply the velocity update formula. We also assume a perturbation method that can be optionally applied, with the aim of improving some aspect of the search process (e.g., fostering diversification).

With these considerations, the pseudo-code defining the working of a generic MOPSO is presented in Listing 1. The first steps are to create the initial swarm and evaluating it (lines 1 and 2). Next, the velocity, local best and global best archive are initialized (lines 3-5) and then the main loop of the algorithm starts and is executed until a termination condition is met (line 6). In the body of the loop, the position and velocity of the particles are updated and they can be possibly modified by a perturbation function; after that, the swarm is re-evaluated and the local best and global best archives are updated. When the loop ends, the algorithm returns the global best archive as a result.

Listing 1: Pseudo-code of a generic MOPSO

```
1  create initial swarm
2  evaluate swam
3  initialize velocity
4  initialize local best
5  initialize global best archive
6  while termination condition is not met
7    update velocity
8    update position
9    perturbation
10   evaluate swarm
11   update local best
12   update global best archive
13 }
14
15 return global best archive
```

3.2 AutoMOPSO jMetal Algorithm Template

Our proposed AutoMOPSO template is presented in Listing 2. It depicts the actual jMetal code and we can observe that it closely mimics the pseudo-code in Listing 1. A key point in defining the template is that all the steps (e.g., create initial swarm, velocity initialization, etc.) are components that can be set at configuration time, thus leading to different MOPSO variants. jMetal includes a catalog of components, providing one or more specific implementations of each one. For example, the creation of the initial swarm assigns random values to the position of the particles by default, but other strategies can be adopted, as shown in the next section.

Listing 2: AutoMOPSO template in jMetal

```
swarm = createInitialSwarm.create(swarmSize);
swarm = evaluation.evaluate(swarm);
velocity = velocityInitialization.initialize(swarm);
localBest = localBestInitialization.initialize(swarm);
globalBestArchive = globalBestInitialization.initialize(swarm);

while (!termination.isMet()) {
```

```
    velocity = velocityUpdate.update(swarm, velocity, localBest,
        globalBestArchive, globalBestSelection,
        inertiaWeightStrategy);
    swarm = positionUpdate.update(swarm, velocity);
    swarm = perturbation.perturb(swarm);
    swarm = evaluation.evaluate(swarm);
    globalBestArchive = globalBestUpdate.update(swarm,
        globalBestArchive);
    localBest = localBestUpdate.update(swarm, localBest);
}

return globalBest;
```

It is worth commenting the velocity update component. This component updates the velocity $\mathbf{v}_i^{t+1}$ from $\mathbf{v}_i^t$ by applying equations such as Eq. 2 or Eq. 3. In our template, the strategies for selecting the global best and to adjust the inertia weight are also parameters that can configured.

About the perturbation, in our template we assume that particles can be modified by applying mutation operators (which are available in jMetal as variation operators of genetic algorithms) with a given frequency.

3.3 Design Space of AutoMOPSO

The configurable components of the AutoMOPSO template and their available choices define the design space of AutoMOPSO. These components may have their own parameters that also need to be configured; examples are the aforementioned velocity update or the probability of the mutations used as perturbations. All these parameters lead to the design space shown in Table 1. The table shows the domain of each parameter as a range of integers ($\mathbb{N}$) or real values ($\mathbb{R}$), in the case of numerical parameters, or as a set of values in the case of categorical parameters. Some (conditional) parameters only have an effect for certain values of other parameters; for example, `tournamentSize` is only active if `globalBestSelection` is set to the value *tournament*. In addition, we include at the end of the table a set of non-configurable components that currently have only a single default value in jMetal, but could be extended in the future. We briefly describe each component in the remainder of this section.

We assume that the maximum size of the archive storing the global best particles (named `leaderArchive` in the table) is 100 while the swarm size can take a value within the range $[10, 200]$. There are three possible types of `leaderArchive` depending of the density estimator used to remove particles when it is full: crowding distance (Deb et al., 2002), hypervolume contribution (Knowles and Corne, 2003; Beume et al., 2007), and spatial spread deviation (Santiago et al., 2019).

The swarm can be initialized (`swarmInitialization`) using three possible strategies, namely random, based on a latin hypercube sampling, and the scheme used in scatter search algorithms (Nebro et al., 2008). The velocity of the particles can be initialized (`velocityInitialization`) to 0.0 (*default*) or by selecting one of the schemes defined in the standard PSO 2007 and the standard PSO 2011 (Clerc, 2012).

The concept of perturbation is implemented as a mutation operator that is applied to the particles of the swarm with a frequency F between 1 an 10 (that is, the mutation is applied to particles at positions $F, 2F, \dots$). There are currently four mutation operators that can be chosen, each of them having its own parameter that controls its intensity, e.g., the distribution index for polynomial-based mutation. The probability of applying a mutation operator in evolutionary algorithms is typically set to

$1/n$, where n is the number of decision variables of the problem. In AutoMOPSO, the parameter `mutationProbabilityFactor` divided by n controls the effective mutation probability. The last configurable parameter regarding mutation is the repair strategy, which determines how to proceed when the result of mutating a variable leads to an out-of-bounds value. Concretely, the mutation choices are:

- *random*: the variable takes a uniform random value within the bounds.
- *bounds*: if the value is lower/higher than the lower/upper bound, the variable is assigned the upper/lower bound (this is called saturation in Caraffini et al. (2019)).
- *round*: if the value is lower/higher than the lower/upper bound, the variable is assigned the upper/lower bound (i.e., this could be named inverse-saturation or reverse-saturation to follow the nomenclature of Caraffini et al. (2019)).

There are four strategies for computing the inertia weight ω in Eq. 2 (`inertiaWeightStrategy`): *constant*, *random* within the range $[0.1, 1.0]$, and linearly increasing and linearly decreasing with minimum and maximum weight values in the ranges $[0.1, 0.5]$ and $[0.5, 1.0]$, respectively.

Table 1: Design space of AutoMOPSO. The leader archive has size 100. Non-configurable parameters currently have a single (default) implementation. Parameters and components in boldface are new additions with respect to (Doblas et al., 2022).

Parameter	Domain	
swarmSize	$[10, 200] \subset \mathbb{N}$	
leaderArchive	{ *crowdingDistance, hypervolume, spatialSpreadDeviation* }	
swarmInitialization	{ *random, latinHypercubeSampling, scatterSearch* }	
velocityInitialization	{ *default*, ***SPSO2007***, ***SPSO2011*** }	
mutation	{ *uniform, nonUniform, polynomial*, ***linkedPolynomial*** }	
mutationProbabilityFactor	$[0.0, 2.0] \subset \mathbb{R}$	
mutationRepairStrategy	{ *random, round, bounds* }	
uniformMutationPerturbation	$[0.0, 1.0] \subset \mathbb{R}$	**if** `mutation`=*uniform*
nonUniformMutationPerturbation	$[0.0, 1.0] \subset \mathbb{R}$	**if** `mutation`=*nonUniform*
polynomialMutationDistIndex	$[5.0, 400.0] \subset \mathbb{R}$	**if** `mutation`=*polynomial*
linkedPolynomialMutationDistIndex	$[5.0, 400.0] \subset \mathbb{R}$	**if** `mutation`=*linkedPolynomial*
mutationFrequency	$[1, 10] \subset \mathbb{N}$	
inertiaWeightStrategy	{ *constant, random, linearIncreasing, linearDecreasing* }	
weight	$[0.1, 1.0] \subset \mathbb{R}$	**if** `inertiaWeightStrategy`=*constant*
weightMin	$[0.1, 0.5] \subset \mathbb{R}$	**if** `inertiaWeightStrategy`≠*constant*
weightMax	$[0.5, 1.0] \subset \mathbb{R}$	**if** `inertiaWeightStrategy`≠*constant*
velocityUpdate	{ *defaultVelocityUpdate, constrainedVelocityUpdate* }	
c1Min	$[1.0, 2.0] \subset \mathbb{R}$	
c1Max	$[2.0, 3.0] \subset \mathbb{R}$	
c2Min	$[1.0, 2.0] \subset \mathbb{R}$	
c2Max	$[2.0, 3.0] \subset \mathbb{R}$	
globalBestSelection	{ ***tournament***, *random* }	
tournamentSize	$[2, 10] \subset \mathbb{N}$	**if** `globalBestSelection`=*tournament*
velocityChangeWhenLowerLimitIsReached	$[-1.0, 1.0] \subset \mathbb{R}$	
velocityChangeWhenUpperLimitIsReached	$[-1.0, 1.0] \subset \mathbb{R}$	
Non-configurable Parameter		
localBestInitialization	Default	
localBestUpdate	Default	
globalBestInitialization	Default	
globalBestUpdate	Default	
positionUpdate	Default	

As indicated in the previous section, there are two alternatives for updating the particles' velocity: *defaultVelocityUpdate* (Eq. 2) and *constrainedVelocityUpdate* (Eq. 3). The φ_1 and φ_2 coefficients (named c_1 and c_2) take values within the ranges $[1.0, 2.0]$ and $[2.0, 3.0]$, respectively. These ranges are set to consider most of possible values used in the literature for setting these parameters (Clerc, 2012). The `globalBestSelection` defines how the global best particle $\mathbf{b}_i^t$ is selected from the leader archive for updating the velocity; the two choices are *random* and N-ary *tournament*, where N is a parameter (`tournamentSize`) between 2 (i.e., binary tournament) and 10.

When the default non-configurable `positionUpdate` component applies Eq. 1, the resulting position can be out-of-bounds. In this case, the position is set to the bound value (in the same way as the *bounds* scheme in the `mutationRepairStrategy` and the velocity is modified by multiplying it by a factor V in the range $[-1.0, 1.0]$, resulting in a velocity change. A negative V value implies a change in the velocity direction, 0.0 means that the particle stops, a positive value less than 1.0 is equivalent to reduce the speed, and the velocity remains the same if the value of V is 1.0. We allow different values of V depending on whether the lower or upper bound is reached (parameters `velocityChangeWhenLowerLimitIsReached` and `velocityChangeWhenUpperLimitIsReached` in Table 1 that are originally considered in SMPSO (Nebro et al., 2009)).

About the rest of non-configurable components, the default policies for initializing and updating the local best are that each particle is its local best at the beginning and the local best is updated if the current particle dominates it. The `globalBestInitialization` and `globalBestUpdate` just insert the particles of the swarm into the leaders archive.

3.4 Finding AutoMOPSO Configurations with irace

In this section, we give some details about how irace is used to find AutoMOPSO configurations. As shown by Fig. 1, irace require as inputs three elements: the parameter space, the algorithm template (i.e., AutoMOPSO), and the training problems. jMetal automatically generates the parameter space description required by irace that contains the same information included in Table 1. This description is stored in a file which is already included in the jMetal project.

Irace starts by sampling uniformly at random a number of valid configurations from the design space and calling an executable program (AutoMOPSO in our case) with them on a small number of training problems. Each execution must return a value that measures the quality of the configuration on that particular problem instance. The main of goal of irace is to find the configuration minimizing that expected value of that measure. Concretely, in this work we have used the hypervolume quality indicator (Zitzler and Thiele, 1998), which takes into account both the convergence, distribution and spread of the Pareto front approximations found by the algorithms. The higher hypervolume value, the better is the front in terms of those properties; since irace assumes that the measure has to be minimized, we multiply the hypervolume by -1. In principle, irace may use any unary quality indicator, such as, additive epsilon, inverted generational distance, etc., and previous studies have investigated the combination of multiple indicators within irace (Bezerra et al., 2020).

Within irace, configurations are evaluated by means of statistical racing, which is a classical method for lazy selection under uncertainty. Within each race, configurations are evaluated on an increasing number of problem instances and poor performing configurations are removed as soon as possible. The goal of racing is to compare good configurations on a large number of (training) problem instances, while avoiding wasting evaluations on poor performing ones. After a race, the surviving configurations are used to update a probabilistic sampling model, which is then used to sample new configurations. The number of configurations sampled at each iteration is dynamically adjusted and depends on the number of parameters and the remaining computational budget. Surviving and new configurations are used together to start a new race. This process of sampling and racing is repeated until a given computational budget is consumed, typically a maximum number of evaluations. Full details about the internal algorithm used by irace are available in the original publication (López-Ibáñez et al., 2016).

We give now some implementation details of AutoMOPSO. The main requirements we considered when designing it was that it had to be flexible enough to allow to produce a particular MOPSO algorithm from any configuration generated by irace. From our experiences with jMetal, we were aware that the architecture presented in Nebro et al. (2015) lacked the desirable flexibility we needed, so we designed a component-based architecture (Nebro et al., 2019) that has been adopted to implement a template for generic MOPSO algorithms.[1] This architecture is complemented with a catalog of MOPSO components that can be combined to produce particular MOPSOs. Furthermore, jMetal has been extended with a package including support for defining any kind of configurable parameter and parsing them in an easy way. AutoMOPSO is the result of using this package in combination with the MOPSO template.

More detailed information about AutoMOPSO and irace can be found in the jMetal documentation.[2]

4 Experiments and Results

This section is devoted to explain the experiments conducted in this study, as well as the results obtained from them. It starts with the experimental setup and the different design options of AutoMOPSO. A comparative study and ablation analysis are then carried out to support the discussions and the final conclusions.

4.1 Experimental Setup

Once we have the algorithm template and the parameter space, keeping in mind the auto-design process in Fig. 1, the last required ingredient is to select the problems to be used as the training set. As mentioned earlier, we focus on bi-objective problems and, in particular, the four problem families, ZDT, DTLZ, WFG, and RE, which are briefly described next.

The ZDT test suite (Zitzler et al., 2000) is composed of six bi-objective problems (we do not consider the ZDT5 problem, as it is binary) that can scale in the number of decision variables, keeping the Pareto optimal front in the same location. Both the DTLZ (Deb et al., 2005) and WFG (Huband et al., 2006) test suites are scalable

[1] https://jmetal.readthedocs.io/en/latest/component.html
[2] https://jmetal.readthedocs.io/en/latest/autoconfiguration.html#automopso

Table 2: Main features of the benchmark problems. The number of variables (n) of the ZDT, DTLZ, and WFG problems are default values.

Problem	n	**Main properties**
ZDT1	30	Convex
ZDT2	30	Concave
ZDT3	30	Disconnected
ZDT4	10	Convex, multi-modal
ZDT5	10	Concave
DTLZ1	7	Linear, multi-modal
DTLZ2	12	Concave
DTLZ3	12	Concave, multi-modal
DTLZ4	12	Concave, biased
DTLZ5	12	Degenerate
DTLZ6	12	Degenerate
DTLZ7	22	Disconnected
WFG1	6	Mixed, biased
WFG2	6	Convex, disconnected, multi-modal, non-separable
WFG3	6	Linear, degenerate, non-separable
WFG4	6	Concave, multi-modal
WFG5	6	Concave, deceptive
WFG6	6	Concave, non-separable
WFG7	6	Concave, biased
WFG8	6	Concave, biased, non-separable
WFG9	6	Concave, biased, multi-modal, deceptive, non-separable
RE21	4	Convex
RE22	3	Mixed
RE23	4	Mixed, disconnected
RE24	3	Convex
RE25	2	Mixed, disconnected

in both decision and objective space, having seven and nine problems, respectively. The RE problem family (Tanabe and Ishibuchi, 2020) is the result of collecting real-worlds problems from the literature. RE contains 16 test instances, from which five have two objectives. A summary of the main features of the problems is included in Table 2.

The methodology applied consists in running irace once per problem family, using such problem family as the training set. As a result, we obtain four different MOPSO algorithms instantiated from AutoMOPSO (AutoMOPSO designs). Since irace adjusts dynamically the population size and the number of runs performed per problem instance, these values are not fixed *a priori*. We give to irace the SMPSO and OMOPSO configurations, which can be instantiated from our AutoMOPSO template (as shown in Table 3) as two initial configurations, while irace generates additional initial configurations by sampling uniformly from the parameter space (as explained in Section 3.4). Each run of irace is given a maximum budget of 100,000 runs of MOPSO algorithms instantiated from AutoMOPSO. Each run of a MOPSO algorithm stops after 10,000 solution evaluations, as recommended by Tanabe and Ishibuchi (2020),[3] and returns up to 100 non-dominated solutions (due to the bounded size of the `leaderArchive`). Within irace, the quality of a MOPSO run is evaluated according to the hypervolume value of the returned `leaderArchive`.

[3]We reduced the number of solution evaluations from the 25,000 used by Doblas et al. (2022) to keep all problems as challenging.

Table 3: Settings of the MOPSO algorithms. (CD: crowding distance, HV: hypervolume contribution, LHS: latin hypercube sampling, SS: scatter search). The subscripts z, d, w and re in AutoMOPSO stand, respectively, for the designs generated from the ZDT, DTLZ, WFG, and RE problems. A value of '–' means that the parameter is disabled.

Parameter	**SMPSO**	**OMOPSO**	**AutoMOPSO$_z$**	**AutoMOPSO$_d$**	**AutoMOPSO$_w$**	**AutoMOPSO$_{re}$**
swarmSize	100	100	39	194	53	111
leaderArchive	CD	CD	HV	HV	HV	HV
swarmInitialization	random	random	random	SS	LHS	random
velocityInitialization	default	default	SPSO2007	SPSO2007	SPSO2011	SPSO2011
mutation	polynomial	uniform	nonUniform	linkedPolynomial	polynomial	linkedPolynomial
mutationProbabilityFactor	1.0	1.0	1.65	0.05	0.04	0.60
mutationRepairStrategy	bounds	bounds	round	random	round	random
uniformMutationPert.	–	0.5	–	–	–	–
nonUniformMutationPert.	–	–	0.1002	–	–	–
polynomialMut.Dist.Index	20.0	–	–	–	105.8681	–
linkedPol.Mut.Dist.Index	–	–	–	151.3372	–	307.3590
mutationFrequency	6	3	10	6	9	6
inertiaWeightStrategy	constant	random	constant	linearDec.	linearInc.	constant
weight	0.1	–	0.12	–	–	0.11
weightMin	–	0.1	–	0.25	0.20	–
weightMax	–	0.5	–	0.96	0.59	–
velocityUpdate	constrained	default	default	constrained	default	default
c1Min	1.5	1.5	1.89	1.75	1.29	1.29
c1Max	2.5	2.0	2.25	2.22	2.56	2.78
c2Min	1.5	1.5	1.50	1.46	1.33	1.12
c2Max	2.5	2.0	2.90	2.07	2.34	2.73
globalBestSelection	tournament	tournament	tournament	tournament	tournament	tournament
tournamentSize	2	2	9	5	8	2
velocityChangeLowerLimit	-1.0	-1.0	0.02	0.86	-0.94	0.99
velocityChangeUpperLimit	-1.0	-1.0	-0.90	0.06	-0.82	0.75

To run irace we have used 64 cores of computers provided by the Supercomputing and Bioinnovation Center (SCBI) of the University of Malaga. The total computing of each irace execution has been roughly 8 hours. Although this time may be considered high, it should be borne in mind that this process only has to be done once, and trying to fine-tune the algorithms manually usually takes considerably longer.

4.2 Finding AutoMOPSO Designs

Table 3 includes the settings of SMPSO, OMOPSO and the four AutoMOPSO designs found by irace. We will use the term AutoMOPSO$_x$, where the subscript x can take the values z, d, w and re to refer to the variant corresponding to the ZDT, DTLZ, WFG, and RE problem families, respectively. A first observation at a glance shows that only two parameters are common in all the AutoMOPSO variants: the hypervolume-based leader archive and the tournament global-best selection. The hypervolume-based archive was also selected in the experiments reported by Doblas et al. (2022), but all the other AutoMOPSO parameters have different values now.

Besides the inclusion of new parameter values, such as linked polynomial-based mutation (Zille et al., 2016) or the SPSO2007 and SPSO2011 velocity initialization strategies, the fact that the stopping condition has been reduced from 25,000 function evaluations to 10,000 makes the two studies not directly comparable.

If we compare the AutoMOPSO designs with SMPSO, we see that only AutoMOPSO$_d$ adopts the constrained velocity update, which is the main feature of SMPSO, but differs in the rest of elements. Therefore, the velocity changes when getting to the lower and upper boundaries are both -1.0 in SMPSO, while in

AutoMOPSO$_d$ the values are 0.86 in the lower limit (the velocity remains almost unchanged) and 0.06 in the upper limit (the velocity is close to 0.0).

4.3 Comparative Study

Once we have found four AutoMOPSO variants from different training sets, the next step is to make a comparative study to assess their performances. We include NSGA-II in this study to use it as a reference, and it is configured with commonly used settings: population size of 100, simulated binary crossover SBX (probability $= 0.9$, distribution index $= 20$) and polynomial-based mutation (probability $= 1/n$, distribution index $= 20$).

For this comparison, we perform 25 independent runs with different random seeds of each algorithm per problem. The quality of the obtained Pareto front approximations is evaluated using two indicators: unary additive epsilon ($I_{\epsilon+}$) (Zitzler et al., 2003), which gives a measure of the approximation distance to a reference set, typically a best-known approximation to the Pareto front; and hypervolume (I_{HV}) (Zitzler and Thiele, 1998), which measures convergence, distribution and spread.

We report the median and interquartile range of the indicator values, as well as the results of applying the Wilcoxon rank sum test to check whether the differences are significant (with a confidence level of 95%) with respect to a reference algorithm.

Table 4 includes the $I_{\epsilon+}$ results. We include the variant AutoMOPSO$_z$ in the last column as the reference algorithm, as it is the best one according to Friedmans' statistical ranking (see Table 6). Cells with dark and light gray background highlight the best and second best values, respectively. The symbols $+$, $-$ and $\approx$ in the cells indicate that the differences with the reference algorithm are significantly better, worse, or there is no difference according to the Pairwise Wilcoxon rank sum test.

At a glance, we observe in Table 4 that the representative SMPSO and OMOPSO algorithms are outperformed by the AutoMOPSO variants, achieving the best result in only one problem (OMOPSO on WFG5); NSGA-II, the only evolutionary algorithm in the study, gets the lowest indicator values on instances WFG4 and RE23. If we focus on the variants AutoMOPSO$_z$, AutoMOPSO$_d$, AutoMOPSO$_w$, and AutoMOPSO$_{re}$ and the values obtained by them on the ZDT, DTLZ, WFG, and RE problems, we see that, in general, each of them performs best on the benchmark used as training set when generating them, although they do not achieve the best $I_{\epsilon+}$ values in all the problems. For example, AutoMOPSO$_z$ finds the Pareto front approximations with best $I_{\epsilon+}$ values on instances ZDT1, ZDT2, ZDT3, and ZDT6, but it is the worst performing algorithm on ZDT4. This indicates that irace was not able to find a compromise design for the five ZDT problems. A similar behavior is observed in the other AutoMOPSO variants.

The summary of the Wilcoxon rank sum test (last row of Table 4) confirms, as the Friedman statistical ranking does, that AutoMOPSO$_z$ is the solver achieving the best overall results. It gets the higher number of best results with significance in all the pairwise comparisons with respect to the rest of algorithms, being AutoMOPSO$_{re}$ the one having the highest number of ties (11 out of the total of 26 problems).

The findings with $I_{\epsilon+}$ are confirmed when analyzing the values of the I_{HV} quality indicator in Table 5 and Friedman's ranking in Table 7. We observe that some cells have a hypervolume value of 0.0, which means that the obtained Pareto front approximations are dominated by the reference point obtained from the reference Pareto fronts. Consequently, they do not contribute to the hypervolume. In this sense, the three instances where OMOPSO, AutoMOPSO$_z$, AutoMOPSO$_w$, and

Table 4: Median and interquartile range (IQR) of the results of the $I_{\epsilon+}$ quality indicator. Cells with dark and light gray background highlights, respectively, the best and second best indicator values. The algorithm in the last column is the reference algorithm, and the symbols +, − and ≈ indicate that the differences with the reference algorithm are significantly better, worse, or there is no difference according to the Wilcoxon rank sum test (confidence level: 95%).

	NSGAII	**SMPSO**	**OMOPSO**	**AutoMOPSO$_d$**	**AutoMOPSO$_w$**	**AutoMOPSO$_{re}$**	**AutoMOPSO$_z$**
ZDT1	2.58e-02(5.14e-03)−	1.26e-02(4.62e-02)−	1.08e-02(7.67e-03)−	5.56e-03(1.85e-04)−	3.69e-02(1.60e-01)−	3.77e-02(1.52e-01)−	5.22e-03(1.22e-04)
ZDT2	8.50e-02(7.28e-01)−	7.55e-03(2.36e-03)−	1.18e-02(6.52e-03)−	6.51e-01(9.94e-01)−	1.62e+00(6.17e-01)−	6.08e-03(3.33e-01)−	5.27e-03(2.67e-04)
ZDT3	2.19e-02(3.47e-03)−	1.39e-01(1.27e-01)−	7.00e-02(1.26e-01)−	4.18e-03(6.66e-04)−	1.39e-01(2.04e-01)−	2.13e-01(1.16e-01)−	2.92e-03(2.23e-04)
ZDT4	8.05e-01(3.14e-01)+	9.04e-03(1.84e-03)+	8.90e+00(5.56e+00)+	6.24e-03(6.41e-04)+	9.14e+00(5.58e+00)+	1.17e+01(5.85e+00)+	1.97e+01(1.19e+01)
ZDT6	3.61e-01(5.77e-02)−	7.08e-03(2.99e-03)−	6.50e-03(1.24e-03)−	6.39e-03(3.03e-04)−	5.87e-03(5.30e-04)−	5.82e-03(4.65e-04) ≈	5.61e-03(2.78e-04)
DTLZ1	8.11e-01(9.22e-01)+	8.77e-03(6.26e-03)+	3.24e+01(2.25e+01)+	8.06e-03(4.66e-03)+	2.54e+01(3.17e+01)+	3.80e+01(1.53e+01)+	5.46e+01(2.22e+01)
DTLZ2	1.26e-02(2.62e-03)−	6.56e-03(6.49e-04)+	6.41e-03(5.94e-04)+	4.99e-03(3.41e-04)+	7.60e-03(4.41e-03) ≈	6.44e-03(1.67e-03)+	7.50e-03(1.63e-03)
DTLZ3	1.14e+01(6.69e+00)+	7.07e-01(6.98e-01)+	8.17e+01(4.97e+01)+	2.10e-02(3.35e-02)+	6.24e+01(6.06e+01)+	9.42e+01(6.48e+01)+	1.69e+02(2.00e+01)
DTLZ4	1.20e-02(3.51e-03)−	6.86e-03(6.72e-04)+	7.51e-03(9.06e-04) ≈	4.96e-03(2.11e-04)+	1.56e-02(9.89e-01)−	7.94e-03(4.44e-03) ≈	8.77e-03(4.64e-03)
DTLZ5	1.19e-02(4.23e-03)−	5.83e-03(7.83e-04) ≈	5.92e-03(4.08e-04) ≈	4.59e-03(3.34e-04)+	9.11e-03(3.92e-03)−	6.35e-03(1.35e-03) ≈	6.80e-03(1.49e-03)
DTLZ6	1.25e+00(1.15e-01)−	9.80e-03(8.27e-01)−	6.30e-03(8.24e-01)−	4.64e-03(4.95e-04)−	2.00e+00(2.00e+00)−	4.57e-03(3.36e-04) ≈	4.49e-03(2.33e-04)
DTLZ7	1.61e-02(2.19e-03)−	6.51e-03(1.76e-03)−	8.66e-03(7.41e-01)−	4.37e-03(7.05e-04)−	1.24e-02(7.45e-01)−	6.14e-03(7.43e-01)−	3.96e-03(1.84e-04)
WFG1	6.64e-01(8.93e-02)−	4.67e-01(6.03e-03)−	4.59e-01(8.65e-03)−	4.65e-01(9.83e-03)−	4.52e-01(3.81e-02)−	4.42e-01(2.59e-02)−	3.65e-01(6.04e-02)
WFG2	1.82e-01(1.73e-01)−	1.75e-02(5.30e-03)−	6.01e-03(1.70e-03)−	1.31e-02(5.69e-03)−	2.83e-03(5.96e-04)+	5.28e-03(4.40e-03)−	3.88e-03(2.00e-03)
WFG3	1.50e-02(2.50e-03)−	1.14e-02(2.32e-03)−	6.94e-03(6.07e-04)−	8.28e-03(7.72e-04)−	5.53e-03(2.34e-04)+	5.92e-03(3.90e-04)−	5.73e-03(2.88e-04)
WFG4	1.22e-02(1.69e-03)+	2.84e-02(5.04e-03)−	2.24e-02(3.15e-03)−	2.73e-02(3.59e-03)−	1.58e-02(3.67e-03)+	1.84e-02(3.06e-03) ≈	1.96e-02(3.68e-03)
WFG5	3.34e-02(2.58e-03)−	2.82e-02(1.49e-03) ≈	2.79e-02(3.56e-04)+	2.84e-02(4.74e-04) ≈	2.83e-02(3.74e-04) ≈	2.84e-02(4.57e-04) ≈	2.83e-02(2.82e-04)
WFG6	1.73e-02(6.32e-03)−	9.52e-03(1.32e-03)−	6.07e-03(5.12e-04)−	7.51e-03(5.02e-03)−	4.57e-03(4.08e-04) ≈	4.82e-03(4.35e-04)−	4.68e-03(3.46e-04)
WFG7	1.30e-02(2.53e-03)−	9.32e-03(7.26e-04)−	6.29e-03(3.09e-04)−	6.45e-03(4.77e-04)−	4.71e-03(9.70e-05)+	4.91e-03(2.72e-04) ≈	4.86e-03(2.41e-04)
WFG8	2.45e-01(7.53e-02) ≈	2.06e-01(4.26e-02) ≈	2.46e-01(2.07e-03) ≈	1.91e-01(3.01e-02)+	2.46e-01(4.07e-03) ≈	2.45e-01(2.04e-03) ≈	2.46e-01(5.26e-02)
WFG9	1.63e-02(3.62e-03)−	1.29e-02(1.82e-03)−	1.09e-02(9.52e-04)−	1.17e-02(1.53e-03)−	8.26e-03(1.54e-03)+	9.43e-03(1.54e-03) ≈	9.68e-03(2.98e-03)
RE21	1.45e-02(3.29e-03)−	6.27e-03(4.49e-04)−	6.00e-03(2.39e-04)−	6.07e-03(3.51e-04)−	5.55e-03(1.96e-04) ≈	5.38e-03(2.64e-04) ≈	5.50e-03(1.90e-04)
RE22	1.58e-02(2.70e-03)−	7.89e-03(8.64e-04)−	7.00e-03(5.04e-04)−	8.68e-03(1.56e-03)−	6.60e-03(5.22e-04)−	6.16e-03(7.58e-04)+	6.28e-03(4.23e-04)
RE23	5.93e-03(2.59e-03)+	7.04e-03(3.86e-03)+	7.32e-03(9.54e-04)+	1.60e-02(4.59e-03)+	3.52e-02(1.57e-02)−	3.34e-02(1.67e-02)−	2.31e-02(1.01e-02)
RE24	1.10e-02(3.63e-03)−	5.64e-03(3.99e-04)−	5.34e-03(3.96e-04)−	2.56e-03(1.35e-04)−	2.39e-03(1.52e-04) ≈	2.41e-03(1.78e-04) ≈	2.39e-03(1.53e-04)
RE25	7.52e-09(1.13e-09)+	3.99e-09(3.30e-10)+	3.85e-09(3.31e-10)+	1.15e-08(1.00e-08)+	1.98e-07(7.70e-08) ≈	3.50e-09(3.60e-10)+	2.03e-07(6.24e-08)
+/≈/−	6/1/19	7/3/16	7/3/16	9/1/16	8/7/11	6/11/9	

Table 5: Median and interquartile range (IQR) of the results of the I_{HV} quality indicator. Cells with dark and light gray background highlights, respectively, the best and second best indicator values. The algorithm in the last column is the reference algorithm, and the symbols +, − and ≈ indicate that the differences with the reference algorithm are significantly better, worse, or there is no difference according to the Wilcoxon rank sum test (confidence level: 95%).

	NSGAII	SMPSO	OMOPSO	AutoMOPSO$_d$	AutoMOPSO$_w$	AutoMOPSO$_{re}$	AutoMOPSO$_z$
ZDT1	6.38e-01(5.97e-03)−	6.57e-01(4.88e-02)−	6.56e-01(1.04e-02)−	6.62e-01(1.74e-05)−	6.22e-01(2.07e-01)−	6.14e-01(2.05e-01)−	6.62e-01(9.64e-06)
ZDT2	2.92e-01(1.58e-01)−	3.28e-01(1.74e-03)−	3.22e-01(6.82e-03)−	4.21e-02(3.29e-01)−	0.00e+00(0.00e+00)−	3.29e-01(9.49e-02)−	3.29e-01(8.78e-06)
ZDT3	4.97e-01(2.82e-03)−	4.19e-01(1.74e-01)−	4.68e-01(1.23e-01)−	5.16e-01(2.70e-04)−	3.91e-01(2.24e-01)−	3.17e-01(1.24e-01)−	5.16e-01(7.38e-06)
ZDT4	1.39e-01(2.17e-01)+	6.60e-01(8.68e-04)+	0.00e+00(0.00e+00) ≈	6.62e-01(9.03e-05)+	0.00e+00(0.00e+00) ≈	0.00e+00(0.00e+00) ≈	0.00e+00(0.00e+00)
ZDT6	1.63e-01(3.33e-02)−	4.01e-01(4.09e-04)−	4.01e-01(9.29e-05)−	4.01e-01(4.86e-05)−	4.01e-01(6.74e-05)−	4.01e-01(1.70e-05) ≈	4.01e-01(6.75e-06)
DTLZ1	0.00e+00(1.38e-01)+	4.93e-01(2.36e-03)+	0.00e+00(0.00e+00) ≈	4.95e-01(7.47e-04)+	0.00e+00(0.00e+00) ≈	0.00e+00(0.00e+00) ≈	0.00e+00(0.00e+00)
DTLZ2	2.09e-01(3.86e-04)+	2.10e-01(3.29e-04)+	2.09e-01(4.47e-04)+	2.11e-01(7.86e-06)+	2.06e-01(6.71e-03) ≈	2.09e-01(2.56e-03)+	2.07e-01(2.04e-03)
DTLZ3	0.00e+00(0.00e+00) ≈	8.58e-02(1.23e-01)+	0.00e+00(0.00e+00) ≈	2.06e-01(1.50e-02)+	0.00e+00(0.00e+00) ≈	0.00e+00(0.00e+00) ≈	0.00e+00(0.00e+00)
DTLZ4	2.09e-01(6.30e-04)+	2.10e-01(3.37e-04)+	2.08e-01(1.12e-03)+	2.11e-01(1.49e-05)+	1.93e-01(2.00e-01)−	2.06e-01(8.00e-03) ≈	2.06e-01(7.56e-03)
DTLZ5	2.10e-01(3.51e-04) ≈	2.11e-01(3.42e-04)+	2.11e-01(5.52e-04)+	2.13e-01(1.69e-05)+	2.06e-01(7.00e-03)−	2.11e-01(1.94e-03) ≈	2.09e-01(2.83e-03)
DTLZ6	0.00e+00(0.00e+00)−	2.12e-01(2.12e-01)−	2.12e-01(2.12e-01)−	2.13e-01(2.51e-05)−	0.00e+00(0.00e+00)−	2.13e-01(1.33e-05)−	2.13e-01(1.05e-05)
DTLZ7	3.22e-01(3.58e-03)−	3.33e-01(1.05e-03)−	3.30e-01(1.14e-01)−	3.35e-01(6.96e-05)−	3.32e-01(2.77e-01)−	3.34e-01(1.16e-01)−	3.35e-01(6.36e-06)
WFG1	2.45e-01(9.29e-02)+	1.06e-01(3.42e-03)−	1.10e-01(9.60e-03)−	1.07e-01(5.97e-03)−	1.19e-01(3.38e-02)−	1.29e-01(4.77e-02)−	1.90e-01(9.84e-02)
WFG2	5.60e-01(2.36e-03)−	5.54e-01(3.88e-03)−	5.62e-01(5.84e-04)−	5.57e-01(3.15e-03)−	5.64e-01(4.73e-04)+	5.62e-01(1.35e-03)−	5.64e-01(1.62e-03)
WFG3	4.90e-01(1.26e-03)−	4.88e-01(1.31e-03)−	4.93e-01(3.31e-04)−	4.93e-01(6.01e-04)−	4.95e-01(2.25e-05)+	4.95e-01(9.57e-05)−	4.95e-01(7.62e-05)
WFG4	2.16e-01(6.85e-04)+	1.96e-01(1.55e-03)−	2.02e-01(1.93e-03)−	1.96e-01(2.51e-03)−	2.09e-01(4.25e-03)+	2.07e-01(2.63e-03) ≈	2.06e-01(4.13e-03)
WFG5	1.95e-01(2.78e-04)−	1.96e-01(1.07e-04)−	1.96e-01(1.29e-04)−	1.96e-01(5.10e-05)−	1.97e-01(2.32e-05) ≈	1.97e-01(5.65e-05) ≈	1.97e-01(2.98e-05)
WFG6	1.99e-01(8.30e-03)−	2.05e-01(1.33e-03)−	2.09e-01(2.73e-04)−	2.08e-01(1.80e-03)−	2.11e-01(8.06e-05)+	2.11e-01(3.04e-04)−	2.11e-01(8.62e-05)
WFG7	2.08e-01(5.78e-04)−	2.06e-01(1.14e-03)−	2.10e-01(3.00e-04)−	2.10e-01(4.97e-04)−	2.11e-01(1.36e-05)+	2.11e-01(3.65e-05)−	2.11e-01(3.12e-05)
WFG8	1.42e-01(1.64e-03)−	1.42e-01(3.36e-03)−	1.44e-01(1.35e-03)−	1.43e-01(2.41e-03)−	1.46e-01(2.13e-03)−	1.49e-01(1.28e-03)+	1.48e-01(1.58e-03)
WFG9	2.35e-01(2.85e-03)−	2.33e-01(6.61e-04)−	2.35e-01(5.43e-04)−	2.35e-01(5.16e-04)−	2.40e-01(2.00e-03)+	2.39e-01(1.13e-03) ≈	2.38e-01(2.23e-03)
RE21	6.71e-01(3.74e-04)−	6.74e-01(1.05e-04)−	6.74e-01(1.17e-04)−	6.74e-01(3.69e-05)−	6.74e-01(1.99e-05)−	6.75e-01(8.80e-06)+	6.75e-01(3.60e-05)
RE22	5.44e-01(1.53e-03)−	5.47e-01(2.36e-04)−	5.48e-01(2.01e-04)−	5.48e-01(9.57e-05)−	5.49e-01(7.80e-05) ≈	5.49e-01(5.93e-05)+	5.49e-01(7.14e-05)
RE23	9.52e-01(1.27e-04)+	9.53e-01(9.25e-05)+	9.53e-01(6.67e-05)+	9.52e-01(6.48e-04)+	9.47e-01(3.85e-03)−	9.48e-01(3.74e-03)−	9.50e-01(1.58e-03)
RE24	9.59e-01(3.49e-04)−	9.60e-01(2.22e-05)−	9.60e-01(4.36e-05)−	9.60e-01(5.17e-06)−	9.60e-01(3.45e-06) ≈	9.60e-01(1.82e-06)+	9.60e-01(6.48e-06)
RE25	8.71e-01(3.50e-11)+	8.71e-01(5.62e-11)+	8.71e-01(3.33e-11)+	8.71e-01(6.84e-10)+	8.71e-01(1.21e-08) ≈	8.71e-01(3.35e-11)+	8.71e-01(1.39e-08)
+/ ≈ /−	8/2/16	8/0/18	5/3/18	8/0/18	6/8/12	6/9/11	

AutoMOPSO$_{re}$ get a I_{HV} equal to 0.0 are ZDT4, DTLZ1, and DTLZ3, which are multimodal problems (see Table 2).

Table 6: Average Friedman's rankings with Holm's Adjusted p-values (0.05) of the compared algorithms. Symbol * indicates the control algorithm and the column at the left contains the overall ranking of positions with regards to $I_{\epsilon+}$.

Algorithm	Ranking	p-value	Holm	Hypothesis
AutoMOPSO$_z$*	3.231	-	-	-
AutoMOPSO$_d$	3.538	1.093E-11	0.05	Rejected
AutoMOPSO$_{re}$	3.731	2.458E-28	0.025	Rejected
OMOPSO	3.846	4.762E-42	0.017	Rejected
AutoMOPSO$_w$	4.077	6.863E-78	0.013	Rejected
SMPSO	4.423	9.809E-153	0.01	Rejected
NSGAII	5.154	0E0	0.008	Rejected

Table 7: Average Friedman's rankings with Holm's Adjusted p-values (0.05) of the compared algorithms. Symbol * indicates the control algorithm and the column at the left contains the overall ranking of positions with regards to I_{HV}.

Algorithm	Ranking	p-value	Holm	Hypothesis
AutoMOPSO$_z$*	3.019	-	-	-
AutoMOPSO$_{re}$	3.481	2.187E-24	0.05	Rejected
AutoMOPSO$_d$	3.769	1.366E-61	0.025	Rejected
OMOPSO	4.096	5.666E-125	0.017	Rejected
AutoMOPSO$_w$	4.288	8.321E-173	0.013	Rejected
SMPSO	4.615	4.53E-272	0.01	Rejected
NSGAII	4.731	0E0	0.008	Rejected

Friedman's test ranks the algorithms for each problem separately, so the lower the ranking, the better the algorithm performs. Then, together with Friedman's ranking, a Holm's post-hoc for multiple comparisons has been applied to check if there are significant differences between the resulting distributions. In this case, following the methodology proposed in (Derrac et al., 2011), Tables 6 and 7 show the statistical results for $I_{\epsilon+}$ and I_{HV} respectively, where symbol * indicates the best ranked technique according to Friedman (then used as the control algorithm) and the fourth column contains the Holm's p-value of each technique with regards to the control one. It is worth noting that AutoMOPSO$_z$ is used as the control algorithm for the two indicators, and all the remaining techniques reached ≤ 0.05 adjusted p-value, hence rejecting the null hypotheses of similar distributions (with regards to AutoMOPSO$_z$).

We have included two figures in Appendix A, Fig. 5 and Fig. 6, containing the boxplots of the $I_{\epsilon+}$ and I_{HV} quality indicator values, respectively. This way, the dispersion of the data, including outliers, can be observed in a visual way. Additionally, we report the tables resulting from applying the Kolmogorov-Smirnov test (Sheskin,

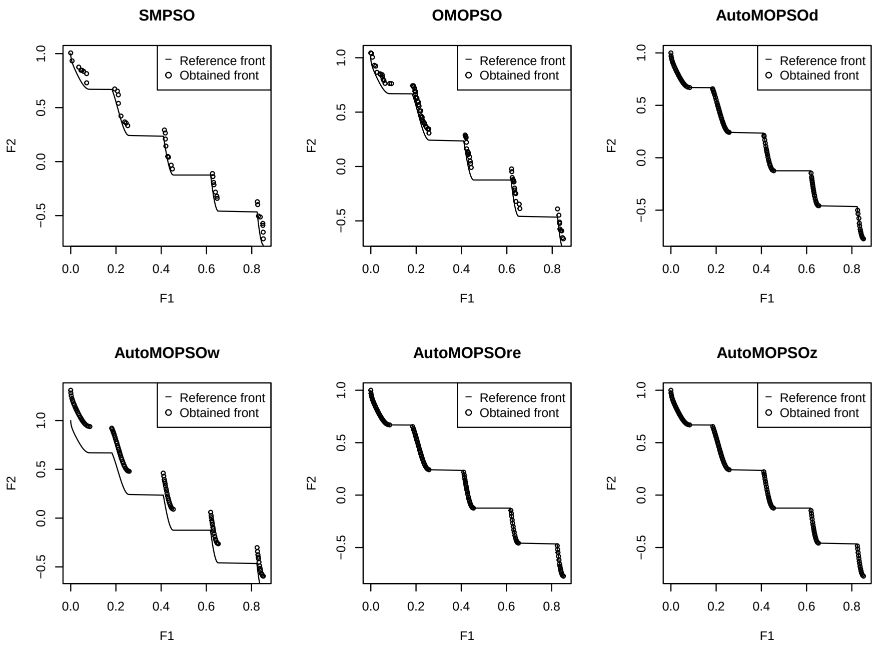

Fig. 2: Pareto front approximations obtained by the MOPSO algorithms corresponding to the median of the I_{HV} for problem ZDT3.

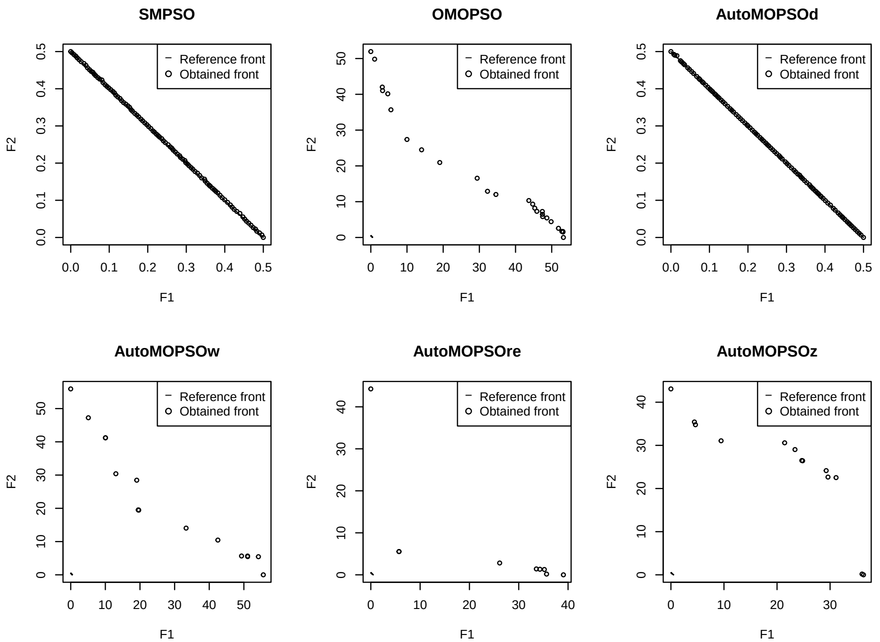

Fig. 3: Pareto front approximations obtained by the MOPSO algorithms corresponding to the median of the I_{HV} for problem DTLZ1.

2011), a non-parametric test to determine whether two samples are derived from different distributions, in Appendix B.

From a graphical viewpoint, Fig. 2 shows the fronts obtained by SMPSO, OMOPSO and the four AutoMOPSO variants for problem ZDT3, to exemplify our results. We observe that SMPSO, OMOPSO, and AutoMOPSO$_w$ are not able to find a front converging to the reference Pareto front in 10,000 evaluations which was the value set as our stopping condition. AutoMOPSO$_w$ has a lower $I_{\epsilon+}$ value than AutoMOPSO$_d$and AutoMOPSO$_{re}$ with statistical confidence, although the differences are not significant when looking at the plots.

We include in Fig. 3 the fronts obtained by the six MOPSO algorithms for problem DTLZ1, where we observe that only SMPSO and AutoMOPSO$_d$ are able to obtain fronts converging to the reference Pareto front on this problem. As it was commented above, SMPSO and AutoMOPSO$_d$ adopt a velocity update scheme based on velocity constriction, so this feature appears necessary to effectively solve some multimodal problems.

4.4 Ablation Analysis

We analyze AutoMOPSO$_z$ in more detail, which was the best overall configuration in the comparison shown above. For this purpose, we perform an ablation analysis (Fawcett and Hoos, 2016) as implemented by the irace package (López-Ibáñez et al., 2016). The analysis starts from a *source* algorithmic configuration and generates all possible configurations that change just one parameter to match the value in the *target* configuration. When the changed parameter is conditional on other parameters, all the parameters required to fulfill the condition are changed simultaneously. All these configurations are evaluated on a number of problem instances and the best one is kept. The next step repeats the process but starting from the best configuration selected in the previous step. The process stops when all parameters have the same value as in the target configuration. The parameter changed at each step, and the impact of its change on solution quality measures the contribution of each individual parameter to the differences between the source and target algorithm designs.

We performed ablation analysis using SMPSO as our source and AutoMOPSO$_z$ as our target (see Table 3) adopting two different sets of problem instances. The analysis on the left side of Fig. 4 evaluates every configuration on all the ZDT problems (5 repetitions per instance). Hypervolume values are transformed into ranks and lower rank values indicate better quality. Starting from the source (SMPSO), the most significant changes occur in the first 6 steps, when changing parameters `leaderArchive`, `velocityChangeLowerLimit`, `swarmSize`, `selectionTournamentSize`, `velocityChangeUpperLimit` and `velocityUpdate`. From the analysis of other AutoMOPSO configurations, the change in `leaderArchive` from *crowding distance* to *hypervolume* seems to be a critical improvement to the design of MOPSOs, as well as the use of a larger tournament size and a smaller swarm size than SMPSO.

The right plot of Fig. 4 repeats the analysis using all problem families (ZDT, DTLZ, WFG and RE, with 5 repetitions per instance), which explains the higher variance of the rank values. In this case, the most significant parameter changes are `leaderArchive`, `c1Max`, `selectionTournamentSize`, `velocityChangeLowerLimit`, and `swarmSize`. A further improvement occurs after setting `mutation` to *nonUniform*. Again, a hypervolume-based archive, a large tournament size and a smaller swarm size are key design settings. Remarkably, some of the settings that were clearly beneficial for the ZDT family, such as default `velocityUpdate` and

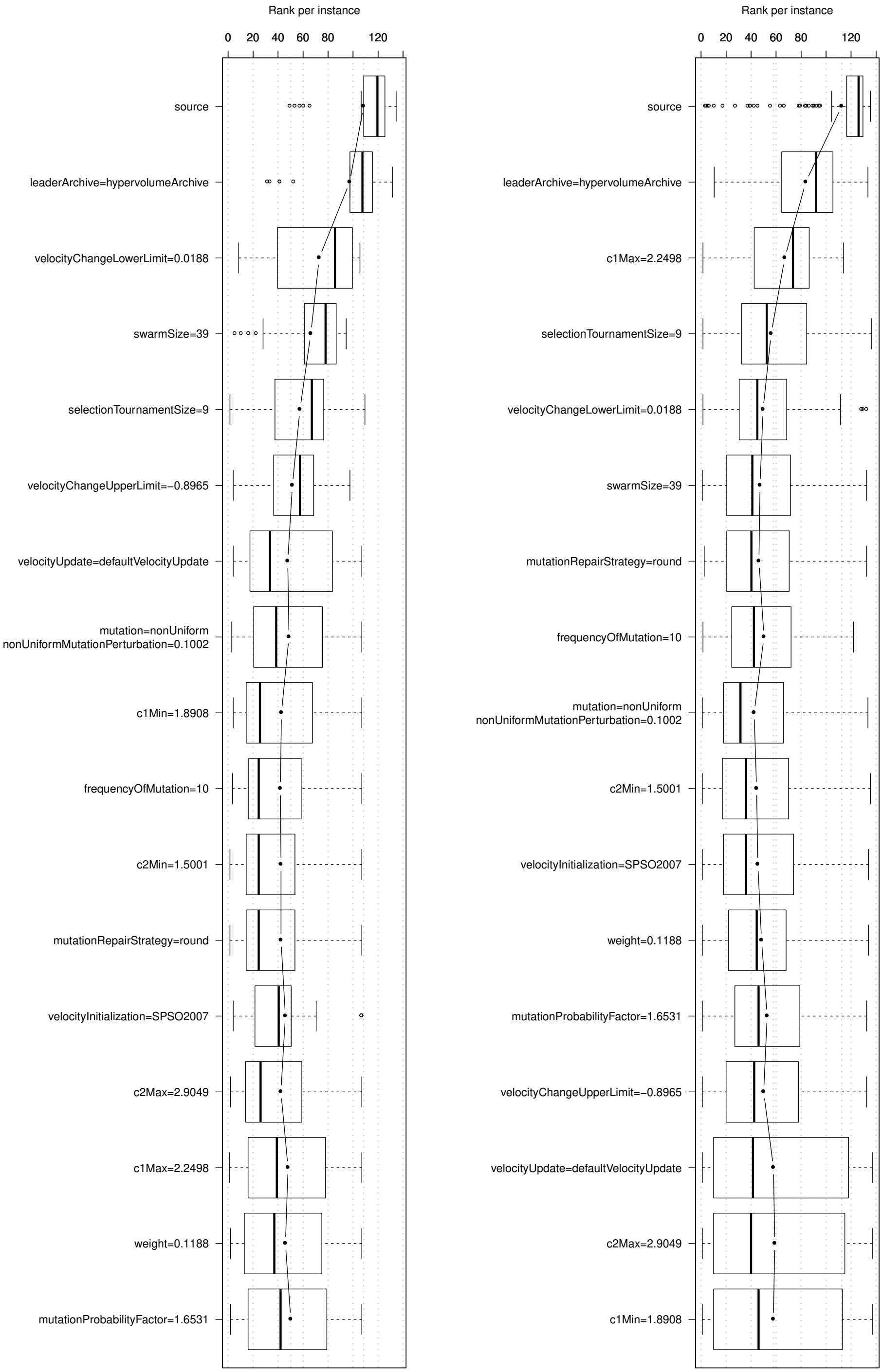

Fig. 4: Ablation analysis of AutoMOPSO$_z$ on the ZDT family (*left*) and on all problem families (*right*). The source is SMPSO.

`velocityChangeUpperLimit` equal to -0.8965, are actually harmful for other problem families. Our conclusion from this observation is that, although AutoMOPSO$_z$ is the best-performing configuration on all problem families, the ZDT family is still not representative of other problem families, which raises the question of how to select a representative problem family for automatic design of MOPSOs. To further analyze this issue, we have repeated our experimentation by using combinations of three of the four benchmark families as training set and letting the last one for validation. The results are included in Appendix C, and they show that AutoMOPSOz remains as the best competitive variant.

5 Discussion

Once the initial experiments have been carried out, this section directly addresses the research questions posed at the beginning of this study, which are answered as follows:

RQ1. Is it possible to find designs of MOPSO algorithms by automatic design that outperform state-of-the-art algorithms such as SMPSO and OMOPSO?
Although this question was already answered positively before (Doblas et al., 2022), here we confirm it by considering additional problems and an extended design space.
RQ2. Is the auto-design process able of obtaining a configuration yielding accurate results when a family of realistic problems are added to the study?
The new problem family RE has been added to the experimental study, which comprises real-world problems. For the sake of simplicity, we have restricted ourselves to bi-objective problems, yet they show structural features of convexity, disconnection and mixed versions. Similar observations are made for the problems selected, which confirm our previous study.
RQ3. What are the design parameters that have the strongest effect on the performance of MOPSO algorithms given the design space considered here?
An ablation analysis of AutoMOPSO$_z$ shows that the choice of leader archive, swarm size and tournament size are the most crucial parameters.
RQ4. Can a configuration obtained for a given test suite lead the algorithm to avoid overfitting and allow a generalization for many other problems?
The comparative study shows that when AutoMOPSO is specifically trained for a given problem family, it obtains the best performance for this family. Although in the scope of a general statistical comparison, it can be observed that AutoMOPSO trained with the ZDT test suite obtains a salient performance for all the problem families, that is, it shows higher generalization capabilities than the remaining models. Nevertheless, the ablation analysis also shows that some design choices of AutoMOPSO$_z$ work well for the ZDT family but harm its performance on other families, which indicates that (unsurprisingly) ZDT is not representative of other problem families and that (surprisingly) even better MOPSO designs are possible.

6 Conclusions

This paper proposes AutoMOPSO, an algorithmic template for the design of MOPSOs. State-of-the-art MOPSOs, such as SMPSO and OMOPSO, can be instantiated from the proposed AutoMOPSO template. Besides key numerical parameters, such as the swarm size, the categorical design choices available in AutoMOPSO give rise to thousands of potential MOPSO designs.

We use an automatic configuration tool (irace) to search the design space of AutoMOPSO given a problem family. In this way, we generate four different AutoMOPSO designs. While each AutoMOPSO design performs best on the problem family that it was designed for, not all of them perform well on the other problem families. When considering all problem families together, the AutoMOPSO design obtained from the ZDT family (AutoMOPSO$_z$) significantly outperforms all other AutoMOPSOs as well as SMPSO and OMOPSO.

The proposed AutoMOPSO was implemented in the jMetal framework and is publicly available for further study and extension by researchers, as well as for its application to other problem scenarios. Further extensions of AutoMOPSO with components taken from other MOPSOs from the literature as well as novel ones would likely improve the results reported here.

We believe that future MOPSO proposals should integrate their novel components into AutoMOPSO, instead of analyzing just a single "novel" MOPSO design, and let the automatic design approach decide how to combine and tune those components for various problem scenarios (Stützle and López-Ibáñez, 2019). In this way, the automatic design process can evaluate thousands of potential designs and an ablation analysis can show the actual impact of any novel components, as illustrated in this work. We argue that our proposed approach would lead to faster research progress, less duplication of efforts and a clearer picture of the key algorithmic components of MOPSOs. Applying AutoMOPSO to real-world problems and analyzing its performance on many-objective problems are also lines of further research.

Funding. This work has been partially funded by the Spanish Ministry of Science and Innovation via Grant PID2020-112540RB-C41 (AEI/FEDER, UE). Carlos A. Coello Coello gratefully acknowledges support from CONACyT grant no. 2016-01-1920 (Investigación en Fronteras de la Ciencia 2016).

Acknowledments. The authors thank the Supercomputing and Bioinnovation Center (SCBI) of the University of Malaga for their provision of computational resources and technical support

Data availability. The source code used in this paper is freely available as part of the jMetal project: https://github.com/jMetal/jMetal

Author's contributions. Conceptualization, AJN, MLI, JGN, CACC; Methodology, AJN, MLI, JGN, CACC; Software, AJN, MLI; Validation, AJN, MLI, JGN; Analysis, AJN, MLI, JGN; writing-original draft preparation, AJN, MLI, JGN; writing-review and editing, AJN, MLI, JGN, CACC. All authors have read and agreed to the published version of the manuscript.

Conflict of interest. Not applicable

Consent for publication. Not applicable

Ethics approval and consent to participate. Not applicable

References

Beume N, Naujoks B, Emmerich MTM (2007) SMS-EMOA: Multiobjective selection based on dominated hypervolume. European Journal of Operational Research 181(3):1653–1669. https://doi.org/10.1016/j.ejor.2006.08.008

Bezerra LCT, López-Ibáñez M, Stützle T (2016) Automatic component-wise design of multi-objective evolutionary algorithms. IEEE Transactions on Evolutionary Computation 20(3):403–417. https://doi.org/10.1109/TEVC.2015.2474158

Bezerra LCT, López-Ibáñez M, Stützle T (2020) Automatically designing state-of-the-art multi- and many-objective evolutionary algorithms. Evolutionary Computation 28(2):195–226. https://doi.org/10.1162/evco_a_00263

Birattari M (2009) Tuning Metaheuristics: A Machine Learning Perspective, Studies in Computational Intelligence, vol 197. Springer, Berlin/Heidelberg, https://doi.org/10.1007/978-3-642-00483-4

Camacho-Villalón CL, Stützle T, Dorigo M (2021) PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms. IEEE Transactions on Evolutionary Computation 26(3):402–416. https://doi.org/10.1109/TEVC.2021.3102863

Caraffini F, Kononova AV, Corne D (2019) Infeasibility and structural bias in differential evolution. Information Sciences 496:161–179. https://doi.org/10.1016/j.ins.2019.05.019

Chugh T, Jin Y, Miettinen K, Hakanen J, Sindhya K (2018) A surrogate-assisted reference vector guided evolutionary algorithm for computationally expensive many-objective optimization. IEEE Transactions on Evolutionary Computation 22(1):129–142

Clerc M (2012) Standard Particle Swarm Optimisation, URL https://hal.archives-ouvertes.fr/hal-00764996, hal-00764996

Clerc M, Kennedy J (2002) The particle swarm - explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6(1):58–73. https://doi.org/10.1109/4235.985692

Coello Coello CA, Lamont GB, Van Veldhuizen DA (2007) Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd edn. Springer, New York, NY, https://doi.org/10.1007/978-0-387-36797-2

Deb K (2001) Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, Chichester, UK

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2):182–197. https://doi.org/10.1109/4235.996017

Deb K, Thiele L, Laumanns M, Zitzler E (2005) Scalable test problems for evolutionary multiobjective optimization. In: Abraham A, Jain L, Goldberg R (eds)

Evolutionary Multiobjective Optimization. Advanced Information and Knowledge Processing, Springer, London, UK, p 105–145, https://doi.org/10.1007/1-84628-137-7_6

Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation 1(1):3–18. https://doi.org/https://doi.org/10.1016/j.swevo.2011.02.002

Doblas D, Nebro AJ, López-Ibáñez M, García-Nieto J, Coello Coello CA (2022) Automatic design of multi-objective particle swarm optimizers. In: Dorigo M, Hamann H, López-Ibáñez M, García-Nieto J, Engelbrecht A, Pinciroli C, Strobel V, Camacho-Villalón CL (eds) Swarm Intelligence, 13th International Conference, ANTS 2022, Lecture Notes in Computer Science, vol 13491. Springer, Cham, Switzerland, p 28–40, https://doi.org/10.1007/978-3-031-20176-9_3

Durillo JJ, Nebro AJ (2011) jMetal: A Java framework for multi-objective optimization. Advances in Engineering Software 42(10):760–771. https://doi.org/10.1016/j.advengsoft.2011.05.014

Durillo JJ, García-Nieto J, Nebro AJ, Coello Coello CA, Luna F, Alba E (2009) Multi-objective particle swarm optimizers: An experimental comparison. In: Ehrgott M, Fonseca CM, Gandibleux X, Hao JK, Sevaux M (eds) Evolutionary Multi-criterion Optimization, EMO 2009, Lecture Notes in Computer Science, vol 5467. Springer, Heidelberg, p 495–509

Fawcett C, Hoos HH (2016) Analysing differences between algorithm configurations through ablation. Journal of Heuristics 22(4):431–458

Huband S, Hingston P, Barone L, While L (2006) A review of multiobjective test problems and a scalable test problem toolkit. IEEE Transactions on Evolutionary Computation 10(5):477–506. https://doi.org/10.1109/TEVC.2005.861417

Jiang S, Zou J, Yang S, Yao X (2022) Evolutionary dynamic multi-objective optimisation: A survey. ACM Computing Surveys 55(4). https://doi.org/10.1145/3524495

Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of International Conference on Neural Networks (ICNN'95). IEEE Press, Piscataway, NJ, pp 1942–1948, https://doi.org/10.1109/ICNN.1995.488968

KhudaBukhsh AR, Xu L, Hoos HH, Leyton-Brown K (2016) SATenstein: Automatically building local search SAT Solvers from Components. Artificial Intelligence 232:20–42. https://doi.org/10.1016/j.artint.2015.11.002

Knowles JD (2006) ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. IEEE Transactions on Evolutionary Computation 10(1):50–66. https://doi.org/10.1109/TEVC.2005.851274

Knowles JD, Corne D (2003) Properties of an adaptive archiving algorithm for storing nondominated vectors. IEEE Transactions on Evolutionary Computation 7(2):100–116

Li B, Li J, Tang K, Yao X (2015) Many-objective evolutionary algorithms: A survey. ACM Computing Surveys 48(1):1–35. https://doi.org/10.1145/2792984

de Lima RHR, Pozo ATR (2017) A study on auto-configuration of multi-objective particle swarm optimization algorithm. In: Proceedings of the 2017 Congress on Evolutionary Computation (CEC 2017). IEEE Press, Piscataway, NJ, pp 718–725, https://doi.org/10.1109/CEC.2017.7969381

López-Ibáñez M, Stützle T (2012) The automatic design of multi-objective ant colony optimization algorithms. IEEE Transactions on Evolutionary Computation 16(6):861–875. https://doi.org/10.1109/TEVC.2011.2182651

López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Stützle T, Birattari M (2016) The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3:43–58. https://doi.org/10.1016/j.orp.2016.09.002

Nebro AJ, Luna F, Alba E, Dorronsoro B, Durillo JJ, Beham A (2008) AbYSS: Adapting scatter search to multiobjective optimization. IEEE Transactions on Evolutionary Computation 12(4):439–457. https://doi.org/10.1109/TEVC.2007.913109

Nebro AJ, Durillo JJ, García-Nieto J, Coello Coello CA, Luna F, Alba E (2009) SMPSO: A new PSO-based metaheuristic for multi-objective optimization. In: 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM), pp 66–73, https://doi.org/10.1109/MCDM.2009.4938830

Nebro AJ, Durillo JJ, Vergne M (2015) Redesigning the jMetal multi-objective optimization framework. In: Jiménez Laredo JL, Silva S, Esparcia-Alcázar AI (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO Companion 2015. ACM Press, New York, NY, p 1093–1100, https://doi.org/10.1145/2739482.2768462

Nebro AJ, López-Ibáñez M, Barba-González C, García-Nieto J (2019) Automatic configuration of NSGA-II with jMetal and irace. In: López-Ibáñez M, Auger A, Stützle T (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO Companion 2019. ACM Press, New York, NY, p 1374–1381, https://doi.org/10.1145/3319619.3326832

Montes de Oca MA, Stutzle T, Birattari M, Dorigo M (2009) Frankenstein's pso: A composite particle swarm optimization algorithm. IEEE Transactions on Evolutionary Computation 13(5):1120–1132

Reyes Sierra M, Coello Coello CA (2005) Improving PSO-based multi-objective optimization using crowding, mutation and ϵ-dominance. In: Coello Coello CA, Hernández Aguirre A, Zitzler E (eds) Evolutionary Multi-criterion Optimization, EMO 2005, Lecture Notes in Computer Science, vol 3410. Springer, Heidelberg, p 505–519

Santiago A, Dorronsoro B, Nebro AJ, Durillo JJ, Castillo O, Fraire HJ (2019) A novel multi-objective evolutionary algorithm with fuzzy logic based adaptive selection of operators: FAME. Information Sciences 471:233–251. https://doi.org/10.1016/j.ins.2018.09.005

Sheskin DJ (2011) Handbook of Parametric and Nonparametric Statistical Procedures, 5th edn. Chapman & Hall/CRC

Stützle T, López-Ibáñez M (2019) Automated design of metaheuristic algorithms. In: Gendreau M, Potvin JY (eds) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol 272. Springer, p 541–579, https://doi.org/10.1007/978-3-319-91086-4_17

Tanabe R, Ishibuchi H (2020) An easy-to-use real-world multi-objective optimization problem suite. Applied Soft Computing 89:106,078

Zille H, Ishibuchi H, Mostaghim S, Nojima Y (2016) Mutation operators based on variable grouping for multi-objective large-scale optimization. In: Chen X, Stafylopatis A (eds) Computational Intelligence (SSCI), 2016 IEEE Symposium Series on, pp 1–8, https://doi.org/10.1109/SSCI.2016.7850214

Zitzler E, Thiele L (1998) Multiobjective optimization using evolutionary algorithms - A comparative case study. In: Eiben AE, Bäck T, Schoenauer M, Schwefel HP (eds) Parallel Problem Solving from Nature – PPSN V, Lecture Notes in Computer Science, vol 1498. Springer, Heidelberg, p 292–301, https://doi.org/10.1007/BFb0056872

Zitzler E, Thiele L, Deb K (2000) Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation 8(2):173–195. https://doi.org/10.1162/106365600568202

Zitzler E, Laumanns M, Thiele L (2002) SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In: Giannakoglou KC, Tsahalis DT, Periaux J, Papaliliou KD, Fogarty T (eds) Evolutionary Methods for Design, Optimisation and Control. CIMNE, Barcelona, Spain, pp 95–100

Zitzler E, Thiele L, Laumanns M, Fonseca CM, Grunert da Fonseca V (2003) Performance assessment of multiobjective optimizers: an analysis and review. IEEE Transactions on Evolutionary Computation 7(2):117–132. https://doi.org/10.1109/TEVC.2003.810758

Appendix A: Statistical Plots

We include in this appendix the boxplots of the values of the $I_{\epsilon+}$ and I_{HV} quality indicators in Figs. 5 and 6, respectively.

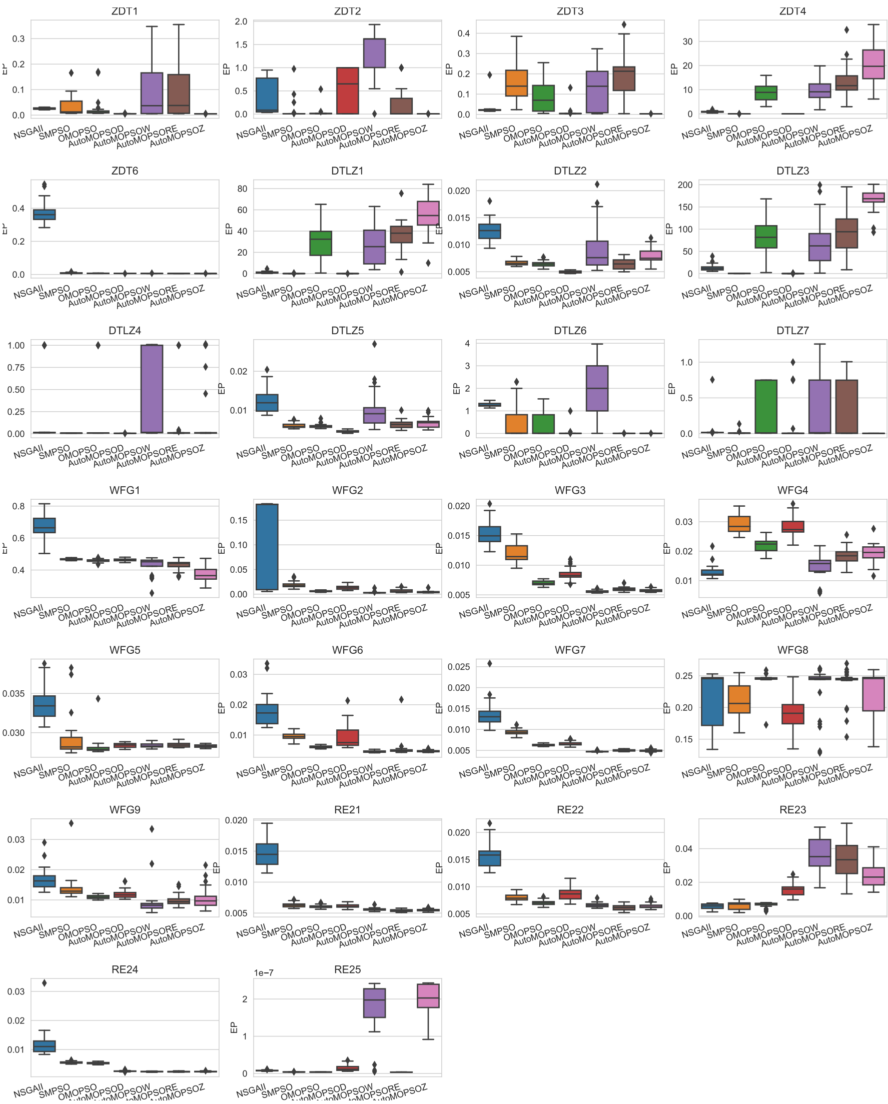

Fig. 5: Boxplots of the $I_{\epsilon+}$ indicator values

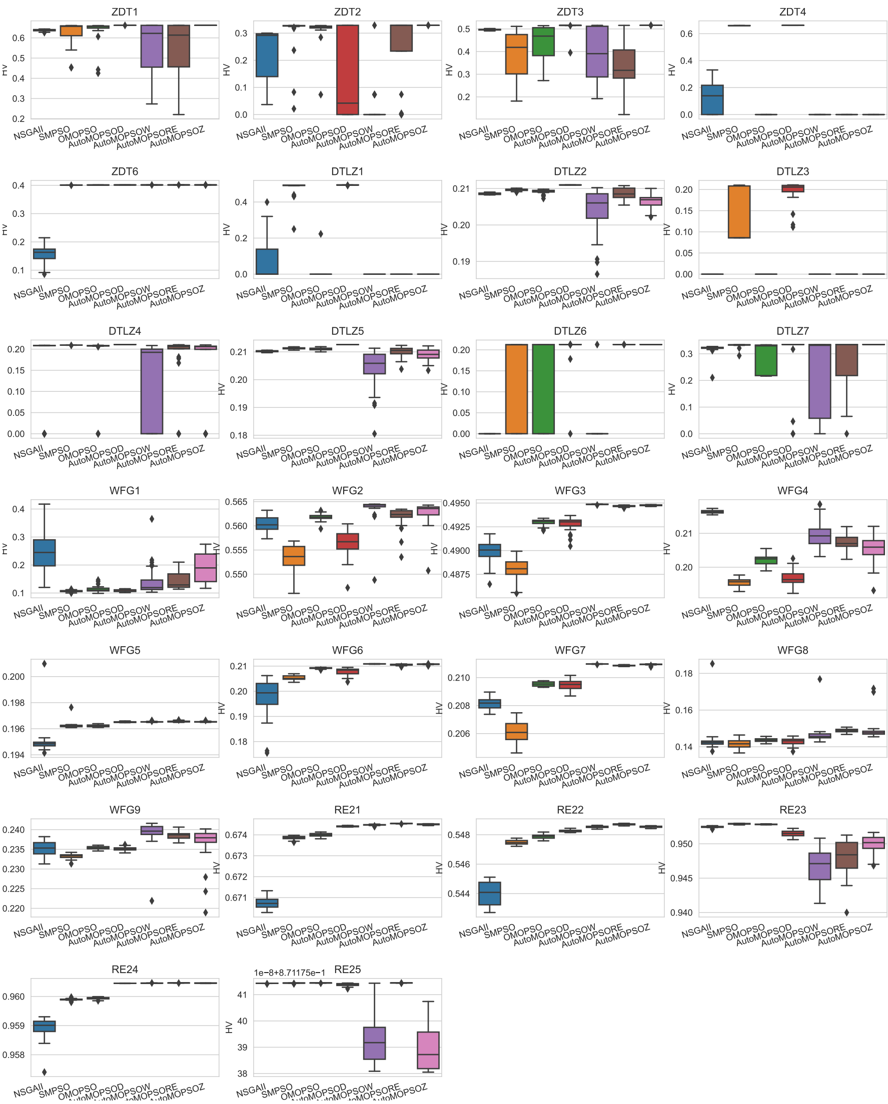

Fig. 6: Boxplots of the I_{HV} indicator values

Appendix B: Kolmogorov-Smirnov Test

This appendix includes the tables obtained when applying the Kolmogorov-Smirnov statistal test to the results of the $I_{\epsilon+}$ (Table 8) and I_{HV} (Table 9) quality indicators. The approach we have followed is to apply the test to all the algorithms with regards to the AutoMOPSO$_z$ variant, which is used as reference algorithm. The resulting tables report the p-values, and those cells where these values are less than 0.05 have been highlighted with a light gray background.

Table 8: Kolmogorov-Smirnov Test of the $I_{\epsilon+}$ quality indicator. The algorithm in the last column is the reference algorithm and each cell contains the p-value obtained when applying the test with the reference algorithm. Cells with gray background highlight p-values less than 0.05 (i.e., the null hypothesis – the two distributions are identical – is rejected).

	NSGAII	SMPSO	OMOPSO	AutoMOPSO$_d$	AutoMOPSO$_w$	AutoMOPSO$_{re}$	AutoMOPSO$_z$
ZDT1	1.58e-14	1.58e-14	1.58e-14	2.51e-07	1.94e-11	1.94e-11	–
ZDT2	1.58e-14	1.94e-11	1.58e-14	1.63e-04	7.91e-13	3.96e-05	–
ZDT3	1.58e-14	1.58e-14	1.58e-14	1.94e-11	7.91e-13	1.58e-14	–
ZDT4	1.58e-14	1.58e-14	3.35e-08	1.58e-14	1.63e-04	5.91e-04	–
ZDT6	1.58e-14	1.58e-14	3.64e-09	3.35e-08	1.63e-04	2.85e-01	–
DTLZ1	1.58e-14	1.58e-14	1.63e-04	1.58e-14	1.63e-04	3.96e-05	–
DTLZ2	3.10e-10	3.96e-05	8.49e-06	1.58e-14	4.75e-01	1.92e-03	–
DTLZ3	1.58e-14	1.58e-14	3.64e-09	1.58e-14	3.35e-08	2.51e-07	–
DTLZ4	1.63e-04	1.58e-06	1.48e-02	1.58e-14	1.92e-03	4.75e-01	–
DTLZ5	1.94e-11	3.56e-02	1.48e-02	1.94e-11	5.61e-03	4.75e-01	–
DTLZ6	1.58e-14	1.58e-14	1.58e-14	3.56e-02	7.91e-13	2.85e-01	–
DTLZ7	1.58e-14	1.58e-14	1.58e-14	3.64e-09	1.94e-11	1.94e-11	–
WFG1	1.58e-14	7.91e-13	3.10e-10	1.94e-11	3.96e-05	5.91e-04	–
WFG2	1.94e-11	7.91e-13	3.96e-05	7.91e-13	5.61e-03	1.48e-02	–
WFG3	1.58e-14	1.58e-14	7.91e-13	1.58e-14	1.48e-02	3.56e-02	–
WFG4	3.35e-08	7.91e-13	5.61e-03	1.94e-11	3.96e-05	4.75e-01	–
WFG5	1.58e-14	3.56e-02	1.58e-06	2.85e-01	4.75e-01	1.56e-01	–
WFG6	1.58e-14	1.58e-14	7.91e-13	1.58e-14	4.75e-01	2.85e-01	–
WFG7	1.58e-14	1.58e-14	1.58e-14	1.58e-14	1.63e-04	2.85e-01	–
WFG8	7.10e-01	3.56e-02	7.79e-02	5.91e-04	9.15e-01	4.75e-01	–
WFG9	3.35e-08	1.58e-06	3.96e-05	3.96e-05	5.61e-03	4.75e-01	–
RE21	1.58e-14	1.94e-11	3.35e-08	3.35e-08	2.85e-01	2.85e-01	–
RE22	1.58e-14	3.64e-09	8.49e-06	3.35e-08	7.79e-02	7.79e-02	–
RE23	1.58e-14	1.58e-14	1.58e-14	1.63e-04	1.92e-03	3.56e-02	–
RE24	1.58e-14	1.58e-14	1.58e-14	5.61e-03	9.15e-01	7.10e-01	–
RE25	1.58e-14	1.58e-14	1.58e-14	1.58e-14	7.10e-01	1.58e-14	–

Focusing on the $I_{\epsilon+}$ indicator, from Table 8 we observe that the p-values allowing to reject the null-hypothesis are most of the problems concerning algorithms NSGA-II, SMPSO, OMOPSO, and AutoMOPSO$_d$. For the AutoMOPSO$_w$ and AutoMOPSO$_{re}$ variants, there are 8 and 13 of the 25 problems, respectively, where the test indicates that the data distribution is the same. These results are consistent with the boxplots of Figure 5, as the cells containing p-values < 0.05 correspond to boxes that present a high degree of overlapping. For example, if we focus on problem RE25, the p-values of AutoMOPSO$_w$ and AutoMOPSO$_{re}$ are, respectively, $7.10e^{01}$ and $1.58e^{14}$ and the boxplot corresponding to that problem in Figure 5 shows that there is an overlapping between the boxes of AutoMOPSO$_z$ and AutoMOPSO$_{re}$, while this does not happen in the case of AutoMOPSO$_w$.

The results of the test for the I_{HV} indicator in Table 9 shows that number of cases where the distributions are similar are reduced with respect to the $I_{\epsilon+}$ values, and they are again consistent with the boxplots included in Figure 6.

Table 9: Kolmogorov-Smirnov Test of the I_{HV} quality indicator. The algorithm in the last column is the reference algorithm and each cell contains the p-value obtained when applying the test with the reference algorithm. Cells with gray background highlight p-values less than 0.05 (i.e., the null hypothesis – the two distributions are identical – is rejected).

	NSGAII	SMPSO	OMOPSO	AutoMOPSO$_d$	AutoMOPSO$_w$	AutoMOPSO$_{re}$	AutoMOPSO$_z$
ZDT1	1.58e-14	1.58e-14	1.58e-14	3.64e-09	7.91e-13	7.91e-13	–
ZDT2	1.58e-14	1.58e-14	1.58e-14	1.58e-06	7.91e-13	2.51e-07	–
ZDT3	1.58e-14	1.58e-14	1.58e-14	1.58e-14	1.58e-14	1.58e-14	–
ZDT4	1.58e-06	1.58e-14	1.00e+00	1.58e-14	1.00e+00	1.00e+00	–
ZDT6	1.58e-14	1.58e-14	1.58e-14	1.58e-14	8.49e-06	7.79e-02	–
DTLZ1	7.79e-02	1.58e-14	1.00e+00	1.58e-14	1.00e+00	1.00e+00	–
DTLZ2	3.64e-09	1.94e-11	2.51e-07	1.58e-14	2.85e-01	5.91e-04	–
DTLZ3	1.00e+00	1.58e-14	1.00e+00	1.58e-14	1.00e+00	1.00e+00	–
DTLZ4	1.63e-04	3.64e-09	1.63e-04	1.58e-14	1.92e-03	7.10e-01	–
DTLZ5	5.91e-04	2.51e-07	1.63e-04	1.58e-14	1.48e-02	1.56e-01	–
DTLZ6	1.58e-14	1.58e-14	1.58e-14	3.10e-10	1.58e-14	5.61e-03	–
DTLZ7	1.58e-14	1.58e-14	1.58e-14	3.10e-10	1.58e-14	7.91e-13	–
WFG1	1.56e-01	1.58e-14	3.64e-09	1.58e-14	1.63e-04	3.56e-02	–
WFG2	8.49e-06	7.91e-13	8.49e-06	1.94e-11	1.63e-04	3.96e-05	–
WFG3	1.58e-14	1.58e-14	1.58e-14	1.58e-14	7.91e-13	5.91e-04	–
WFG4	1.58e-14	7.91e-13	3.96e-05	3.10e-10	1.48e-02	2.85e-01	–
WFG5	7.91e-13	7.91e-13	1.58e-14	1.48e-02	9.15e-01	2.85e-01	–
WFG6	1.58e-14	1.58e-14	1.58e-14	1.58e-14	3.96e-05	1.92e-03	–
WFG7	1.58e-14	1.58e-14	1.58e-14	1.58e-14	3.96e-05	3.35e-08	–
WFG8	1.94e-11	7.91e-13	7.91e-13	7.91e-13	5.61e-03	1.48e-02	–
WFG9	3.96e-05	3.64e-09	3.64e-09	3.64e-09	1.92e-03	7.79e-02	–
RE21	1.58e-14	1.58e-14	1.58e-14	7.91e-13	5.91e-04	1.94e-11	–
RE22	1.58e-14	1.58e-14	1.58e-14	7.91e-13	7.10e-01	7.91e-13	–
RE23	1.58e-14	1.58e-14	1.58e-14	2.51e-07	8.49e-06	1.48e-02	–
RE24	1.58e-14	1.58e-14	1.58e-14	3.96e-05	1.56e-01	1.58e-06	–
RE25	1.58e-14	1.58e-14	1.58e-14	1.58e-14	7.10e-01	1.58e-14	–

Table 10: Settings of the AutoMOPSO algorithms when using three benchmark families for training. (CD: crowding distance, HV: hypervolume contribution, LHS: latin hypercube sampling, SS: scatter search). The characters in the subscripts zdw, zdr, zwr and dwr in AutoMOPSO stand, respectively, for the designs generated using combinations of the ZDT (z), DTLZ (d), WFG (w), and RE (r) problems. A value of '–' means that the parameter is disabled.

Parameter	**AutoMOPSO$_z$**	**AutoMOPSO$_{zdw}$**	**AutoMOPSO$_{zdr}$**	**AutoMOPSO$_{zwr}$**	**AutoMOPSO$_{dwr}$**
swarmSize	39	85	40	65	17
leaderArchive	HV	HV	HV	HV	HV
swarmInitialization	random	LHS	SS	LHS	SS
velocityInitialization	SPSO2007	SPSO2007	default	SPSO2007	SPSO2011
mutation	nonUniform	polynomial	uniform	uniform	uniform
mutationProbabilityFactor	1.65	0.8	1.71	0.35	0.75
mutationRepairStrategy	round	random	round	round	round
uniformMutationPert.	–	–	0.43	0.74	0.18
nonUniformMutationPert.	0.1002	–	–	–	–
polynomialMut.Dist.Index	–	103.57	–	–	–
linkedPol.Mut.Dist.Index	–	–	–	151.3372	–
mutationFrequency	10	5	10	4	9
inertiaWeightStrategy	constant	random	linearInc.	constant	random
weight	0.12	–	–	0.28	–
weightMin	–	0.18	0.36	–	0.20
weightMax	–	0.76	0.63	–	0.59
velocityUpdate	default	constrained	constrained	default	constrained
c1Min	1.89	1.33	1.04	1.55	1.26
c1Max	2.25	2.02	2.74	2.62	2.16
c2Min	1.50	1.38	1.64	1.15	1.01
c2Max	2.90	2.24	2.43	2.19	2.06
globalBestSelection	tournament	tournament	tournament	tournament	tournament
tournamentSize	9	8	8	2	3
velocityChangeLowerLimit	0.02	0.67	-0.52	0.32	-0.21
velocityChangeUpperLimit	-0.90	-0.95	-0.79	-0.08	0.94

Appendix C: Complementary Study

The question of the influence of the problems used as a training set is left open from the experimentation carried out in the paper. To investigate this issue further, we have conducted an additional experiment using a different approach to define the training set. Concretely, we have divided the families of benchmarks into groups of three, remaining the fourth group for validation; as a result, we have four new AutoMOPSO variants:

- AutoMOPSO$_{zdw}$ (training set: ZDT, DTLZ, WFG)
- AutoMOPSO$_{zdr}$ (training set: ZDT, DTLZ, RE)
- AutoMOPSO$_{zwr}$ (training set: ZDT, WFG, RE)
- AutoMOPSO$_{dwr}$ (trainng set: DTLZ, WFG, RE)

We have run irace to find the configurations of these variants, which are reported in Table 10. We include the components of AutoMOPSO$_z$, the version that showed the best performance, to facilitate comparison with it. We show the results of the I_{HV} quality indicator and Friedman's ranking in Tables 11 and 12, respectively. From the tables we observe that AutoMOPSO$_{zwr}$ reaches the first position of the ranking together with AutoMOPSO$_z$, thus confirming that the use of the ZDT for training leads to a MOPSO that provides the best overall performance in the context of the experimentation carried out.

From the configurations reported in Table 10, we see that all the variants use the external archive with the hypervolume-based density estimator and the tournament approach for global best selection. If we compare the parameters of AutoMOPSO$_z$

and AutoMOPSO$_{zwr}$, they only share those components as well as the default scheme for velocity update.

Table 11: Median and interquartile range (IQR) of the results of the I_{HV} quality indicator in the coplementary study. Cells with dark and light gray background highlights, respectively, the best and second best indicator values.

	AutoMOPSO$_{zdw}$	AutoMOPSO$_{zdr}$	AutoMOPSO$_{zwr}$	AutoMOPSO$_{dwr}$	AutoMOPSO$_{z}$
ZDT1	$6.62e-01_{1.3e-05}$	$6.62e-01_{1.5e-05}$	$6.62e-01_{2.4e-03}$	$6.62e-01_{5.0e-04}$	$6.62e-01_{1.2e-05}$
ZDT2	$3.29e-01_{1.7e-05}$	$3.29e-01_{2.3e-05}$	$3.29e-01_{2.4e-05}$	$3.29e-01_{3.8e-04}$	$3.29e-01_{8.3e-06}$
ZDT3	$5.16e-01_{4.3e-05}$	$5.16e-01_{1.5e-04}$	$5.12e-01_{7.2e-03}$	$5.15e-01_{6.3e-04}$	$5.16e-01_{1.4e-05}$
ZDT4	$0.00e+00_{0.0e+00}$	$6.62e-01_{2.8e-04}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$
ZDT6	$4.01e-01_{2.2e-05}$	$4.01e-01_{1.0e-04}$	$4.01e-01_{2.5e-05}$	$4.01e-01_{9.9e-05}$	$4.01e-01_{1.8e-05}$
DTLZ1	$0.00e+00_{2.5e-01}$	$4.94e-01_{1.2e-01}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{1.8e-01}$	$0.00e+00_{0.0e+00}$
DTLZ2	$2.11e-01_{2.1e-04}$	$2.11e-01_{9.6e-05}$	$2.11e-01_{1.5e-04}$	$2.11e-01_{9.9e-05}$	$2.07e-01_{2.8e-03}$
DTLZ3	$0.00e+00_{8.6e-02}$	$2.10e-01_{1.2e-01}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{4.1e-02}$	$0.00e+00_{0.0e+00}$
DTLZ4	$2.10e-01_{8.4e-04}$	$2.11e-01_{2.0e-04}$	$2.11e-01_{1.9e-04}$	$2.11e-01_{1.2e-04}$	$2.06e-01_{7.4e-03}$
DTLZ5	$2.12e-01_{2.2e-04}$	$2.12e-01_{8.0e-05}$	$2.12e-01_{1.7e-04}$	$2.12e-01_{9.2e-05}$	$2.09e-01_{4.1e-03}$
DTLZ6	$2.13e-01_{1.3e-05}$	$0.00e+00_{1.7e-01}$	$2.13e-01_{6.7e-06}$	$2.13e-01_{1.2e-05}$	$2.13e-01_{9.3e-06}$
DTLZ7	$3.35e-01_{9.0e-06}$	$3.35e-01_{2.0e-05}$	$3.35e-01_{3.7e-04}$	$3.34e-01_{5.8e-04}$	$3.35e-01_{5.4e-06}$
WFG1	$1.13e-01_{4.4e-03}$	$1.05e-01_{6.5e-03}$	$1.38e-01_{5.7e-02}$	$1.16e-01_{1.6e-02}$	$1.39e-01_{9.3e-02}$
WFG2	$5.63e-01_{7.4e-04}$	$5.60e-01_{1.9e-03}$	$5.63e-01_{1.3e-03}$	$5.63e-01_{4.5e-04}$	$5.64e-01_{7.4e-04}$
WFG3	$4.95e-01_{1.3e-04}$	$4.93e-01_{4.4e-04}$	$4.95e-01_{4.6e-05}$	$4.95e-01_{8.5e-05}$	$4.95e-01_{6.3e-05}$
WFG4	$2.01e-01_{2.3e-03}$	$1.96e-01_{3.1e-03}$	$2.04e-01_{2.5e-03}$	$2.03e-01_{1.8e-03}$	$2.09e-01_{6.6e-03}$
WFG5	$1.97e-01_{4.9e-05}$	$1.97e-01_{4.5e-05}$	$1.97e-01_{6.1e-05}$	$1.97e-01_{3.5e-05}$	$1.97e-01_{4.8e-05}$
WFG6	$2.10e-01_{2.5e-04}$	$2.09e-01_{8.6e-04}$	$2.11e-01_{1.1e-04}$	$2.11e-01_{9.4e-05}$	$2.11e-01_{1.1e-04}$
WFG7	$2.11e-01_{8.0e-05}$	$2.10e-01_{3.9e-04}$	$2.11e-01_{3.1e-05}$	$2.11e-01_{5.6e-05}$	$2.11e-01_{5.9e-05}$
WFG8	$1.43e-01_{2.4e-03}$	$1.36e-01_{3.0e-03}$	$1.48e-01_{1.6e-03}$	$1.45e-01_{1.8e-03}$	$1.47e-01_{1.5e-03}$
WFG9	$2.37e-01_{5.6e-04}$	$2.35e-01_{1.2e-03}$	$2.38e-01_{8.8e-04}$	$2.37e-01_{7.9e-04}$	$2.39e-01_{2.1e-03}$
RE21	$6.74e-01_{2.8e-05}$	$6.74e-01_{4.6e-05}$	$6.75e-01_{1.1e-05}$	$6.74e-01_{3.2e-05}$	$6.75e-01_{1.8e-05}$
RE22	$5.48e-01_{1.3e-04}$	$5.48e-01_{1.7e-04}$	$5.49e-01_{6.4e-05}$	$5.49e-01_{4.7e-05}$	$5.49e-01_{1.0e-04}$
RE23	$9.51e-01_{1.6e-03}$	$9.51e-01_{1.8e-03}$	$9.43e-01_{5.6e-03}$	$9.46e-01_{5.4e-03}$	$9.50e-01_{2.5e-03}$
RE24	$9.60e-01_{3.1e-06}$	$9.60e-01_{4.6e-06}$	$9.60e-01_{3.3e-06}$	$9.60e-01_{5.9e-06}$	$9.60e-01_{4.7e-06}$
RE25	$8.71e-01_{1.1e-08}$	$8.71e-01_{4.4e-10}$	$8.71e-01_{5.1e-11}$	$8.71e-01_{6.1e-11}$	$8.71e-01_{1.3e-08}$

Table 12: Average Friedman's rankings with Holm's Adjusted p-values (0.05) of the compared algorithms in the complementary study. Symbol * indicates the control algorithm and the column at the left contains the overall ranking of positions with regards to I_{HV}.

Algorithm	Ranking	p-value	Holm	Hypothesis
AutoMOPSO$_{zwr}$	2,577	0E0	0	-
AutoMOPSO$_{z}$	2,577	1E0	0,05	Accepted
AutoMOPSO$_{dwr}$	3,038	5,776E-32	0,025	Rejected
AutoMOPSO$_{zdw}$	3,192	1,79E-55	0,017	Rejected
AutoMOPSO$_{zdr}$	3,615	1,848E-154	0,013	Rejected