algorithms

MDPI

Article

Improving the Quantum Multi-Swarm Optimization with Adaptive Differential Evolution for Dynamic Environments

Vladimir Stanovov [1,*], Shakhnaz Akhmedova [1], Aleksei Vakhnin [1], Evgenii Sopov [1], Eugene Semenkin [1] and Michael Affenzeller [2]

[1] Department of System Analysis and Operations Research, Reshetnev Siberian State University of Science and Technology, 660037 Krasnoyarsk, Russia; shahnaz@inbox.ru (S.A.); alexeyvah@gmail.com (A.V.); evgenysopov@gmail.com (E.S.); eugenesemenkin@yandex.ru (E.S.)

[2] Heuristic and Evolutionary Algorithms Laboratory, University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg, Austria; michael.affenzeller@fh-hagenberg.at

* Correspondence: vladimirstanovov@yandex.ru

Abstract: In this study, the modification of the quantum multi-swarm optimization algorithm is proposed for dynamic optimization problems. The modification implies using the search operators from differential evolution algorithm with a certain probability within particle swarm optimization to improve the algorithm's search capabilities in dynamically changing environments. For algorithm testing, the Generalized Moving Peaks Benchmark was used. The experiments were performed for four benchmark settings, and the sensitivity analysis to the main parameters of algorithms is performed. It is shown that applying the mutation operator from differential evolution to the personal best positions of the particles allows for improving the algorithm performance.

Keywords: dynamic environments; differential evolution; particle swarm optimization; evolutionary algorithms

Citation: Stanovov, V.; Akhmedova, S.; Vakhnin, A.; Sopov, E.; Semenkin, E.; Affenzeller, M. Improving the Quantum Multi-Swarm Optimization with Adaptive Differential Evolution for Dynamic Environments. *Algorithms* **2022**, *15*, 154. https://doi.org/10.3390/a15050154

Academic Editor: Antonio Della Cioppa

Received: 29 March 2022
Accepted: 28 April 2022
Published: 30 April 2022

1. Introduction

The development of computational intelligence approaches has allowed applying these methods in different areas nowadays. In particular, the evolutionary computation (EC) [1] methods and swarm-based algorithms proved themselves to be problem-independent and universal approaches for solving optimization tasks with different types of complexity. Most of these algorithms are developed to address stationary environments, where the objective function does not change over time. However, optimization in dynamic environments has become popular in the last three decades, which is indicated in several surveys of the state-of-the-art [2–4]. Many real-world applications can be formulated as a dynamic optimization problem (DOP), where the problem itself changes over time. The changing environment during the optimization process is a challenging task for most bio-inspired algorithms, as they are mainly developed under an assumption that a single best final solution should be found by the end of the computational resource, and the goal function does not change.

In the non-stationary environment, the goal consists in tracking the optimum positions during the optimization process. The problems which are formulated as DOPs, are found in different areas of human activity such as path planning [4,5], pollution control [6], searching for survivors with unmanned aerial vehicles (drones) [7], and others. The development of novel approaches for DOPs relies on well-established benchmarks, namely the classical Moving Peaks Benchmark (MPB) [8], Generalized Dynamic Benchmark Generator (GDBG) [9], or the recently proposed Generalized Moving Peaks Benchmark (GMPB) [10,11] and the Deterministic Distortion and Rotation Benchmark (DDRB) [12].

The dominating methods often applied for DOPs are the particle swarm optimization (PSO)-based algorithms and their modifications such as multi-swarm approaches [13,14],

however, certain studies used Differential Evolution (DE) [15] or, for example, Artificial Bee Colony (ABC) algorithm [16]. Yet little has been done for the development of hybrid approaches, which would use the advantages of different methods. Thus, the goal of this study is to establish the ways in which DE can be added to PSO-based approaches, improving the resulting performance.

In this study, the multi-swarm Quantum Swarm Optimization (mQSO) [13] algorithm is modified with the Differential Evolution search operator. Several versions of the resulting mQSODE algorithm with different mutation operators are considered, and the sensitivity analysis for main parameters is conducted. The experiments are performed on the Generalized Moving Peaks Benchmark [10] with four scenarios. The main contributions of the current study can be summarized as follows:

1. It is shown that applying differential mutation to current positions of the particles is inferior to the variant when it is applied to personal best positions;
2. The probability of applying differential evolution search should be relatively small, and the PSO should be the main search engine to achieve competitive results;
3. The scaling factor in differential mutation should be carefully chosen, as it significantly influences performance, whereas crossover rate does not seem to have any significant influence;
4. Applying differential mutation with a small probability results in higher diversity in the population compared to mQSO given the same population size;
5. The GMPB favors larger population sizes than recommended values, derived from testing on other benchmarks.

The rest of the paper is organized as follows: the next section describes related work, i.e., algorithms and methods applied to dynamic optimization, after this the generalized moving peaks benchmark used in this study is given, next, the proposed approach is presented, after that, the experimental setup and results are provided, and finally, the conclusions are given.

2. Materials and Methods

2.1. Related Work

Without loss of generality, we can state the classic unconstrained single-objective continuous DOP as follows.

$$optimize\ f(\overline{x}, \overline{\varphi}, t),\ \overline{x} \in F(t) \subseteq S, t \in T \tag{1}$$

where $S \in R^d$, d is the search space dimensionality, S is the search space, t denotes the time, $\overline{\varphi}$ is the set of parameters of the current environment, $f : S \times T \to R$, is the objective function that returns a value $(f(\overline{x}, \overline{\varphi}, t)) \in R$ using the feasible solution from the search space at time t, $F(t)$—is a set of feasible solutions at time t.

There are a lot of approaches that have been proposed to handle DOPs, some of them are described below:

- Detection-based approaches. This group of methods can be divided into two subcategories: a solution reevaluation [8,17] and detection changes in an algorithm's behavior [18,19]. The solution reevaluation is based on the reevaluation of some solution at a given frequency. If the fitness value of this solution is changed, then the environment is changed too. The second way is based on monitoring the drop in the average performance of an algorithm over several generations. Additionally, some algorithms evaluate the diversity of fitness values.
- Introducing diversity when changes are detected. When solving the stationary optimization problem, the population has to converge to an optimum. When solving DOP, if the whole population is converged to some optimum and the environment is changed, the algorithm cannot react to the change quickly and effectively. The following approach has been proposed to diversify the population. In [18], a partial hypermutation step has been proposed. It replaces a predefined percentage of

individuals, generated randomly from the feasible search space. The proposed genetic programming approach [19] increases the mutation factor, reduces elitism, and increases the probability of crossover when the environment is changed.

- Maintaining diversity during the search. Methods from this group do not detect directly when the environment is changed. They are based on keeping the diversity of the population on the same level. In the random immigrants method [20], a predefined number of randomly generated individuals are added to the population in every generation. The sentinel placement method [21] initializes the predefined number of points that cover the search space. Authors use the proposed method to cover the search space more uniformly.
- Memory approaches. If the changes in the environment are periodical, i.e., the landscape of the problem can return, it is useful to contain previous solutions to save computational resources. The previous good solutions are stored in a direct memory [22,23].
- Prediction approaches. In this case, a heuristic tries to find some patterns that are predictable and use this information to increase the search performance when the environment is changed. A prediction of the optima movement is described in [24]. The prediction model is based on a sequence of optimum positions found in the previous environments.
- Multi-population approaches. The idea of the approach is to share responsibilities between populations. For example, the first population may focus on searching for the optimum while the second population focuses on tracking any environmental changes. The approach [25] uses the predefined number of small populations to find better solutions and one big population to track changes. Another method [26] uses the main big population to optimize the current environment and dedicates some small populations to track the changes in the environment.

Most of the modern approaches utilize the multi-population concept, for example, in [27], a general adaptive multi-swarm framework is proposed, in which the number of swarms is dynamically adapted and some of the swarms are set to an inactive state to save computational resource. In the AMP framework proposed in [28], several heuristics are applied, including population exclusion, avoidance of explored peaks, population hibernation and wakening, as well as the Brownian movement.

Other studies, such as [29] or [30], propose the use of clustering methods for dividing the population into sub-swarms, and depending on a threshold condition, close swarms could be merged into one. In some papers, such as [31,32] it was shown that the idea of having multiple dynamic swarms could be efficiently applied also to static test suites such as the well-known Congress on Evolutionary Computation competition problems [33], real-world problems, such as energy consumption optimization [34] or constrained optimization problems [35].

Although there is a certain amount of studies on dynamic optimization, which use DE-based algorithms, such as DynPopDE [15], most of them are focused on PSO due to its simplicity and ability to converge fast. However, in stationary environments, hybrid approaches are well-known, for example, in [36], a hybrid algorithm is applied to the optimal design of water distribution systems, in [37], the algorithm with repulsive strategy is proposed, and in [38], the soft island model with different types of populations is proposed. Considering these studies, the development of hybrid approaches combining PSO and DE could be a promising research direction.

2.2. Generalized Moving Peaks Benchmark

As mentioned before, real-world dynamic optimization problems have complex landscapes and, therefore, algorithms applied to them have to be able to find desirable solutions while also reacting to the environmental changes. The latter is important due to the fact that these changes cause the shift of the optimal solution, namely, the previously found solution becomes suboptimal and the new one should be found for a current environment.

Thus, in order to evaluate the performance of the proposed algorithms, it is crucial to use benchmark problems that can be described by the following features:

- Easy implementation;
- High configurability with respect to the number of components, the shape of components, dimension, environmental change frequency, and severity;
- Variety of characteristics (modularity, components with different condition numbers, different intensity of local and global modality, heterogeneity, different levels of irregularities, symmetric and asymmetric components, and so on).

One of the most popular and well-known generators for DOP benchmarks is the Moving Peaks Benchmark (MPB) [8], which is based on changing the components and their locations over time. However, this benchmark generator cannot be considered useful or practical due to its irrelevance to real-world problems. Therefore, the Generalized Moving Peaks Benchmark (GMPB) generator was later proposed [10]. GMPB is a benchmark generator with fully controllable features: it is capable of generating problems with fully non-separable to fully separable structure, with homogeneous or highly heterogeneous, balanced or highly imbalanced sub-functions, with unimodal or multimodal, symmetric or asymmetric and smooth or highly irregular components.

The GMPB benchmark generator, introduced in [10], has the following baseline function:

$$f^{(t)}(x) = \max_{k \in \{1,\dots,m\}} \left\{ h_k^{(t)} - \sqrt{T\left(\left(x - c_k^{(t)}\right)^T R_k^{(t)T}, k\right) W_k^{(t)} T\left(R_k^{(t)}\left(x - c_k^{(t)}\right)^T, k\right)} \right\} \tag{2}$$

$$T(y_j, k) = \begin{cases} exp\left(log(y_j) + \tau_k^{(t)}\left(sin\left(\eta_{k,1}^{(t)} log(y_j)\right) + sin\left(\eta_{k,2}^{(t)} log(y_j)\right)\right)\right), \; y_j > 0 \\ 0, \; y_j = 0 \\ exp\left(log(|y_j|) + \tau_k^{(t)}\left(sin\left(\eta_{k,3}^{(t)} log(|y_j|)\right) + sin\left(\eta_{k,4}^{(t)} log(|y_j|)\right)\right)\right), \; y_j > 0 \end{cases} \quad j = \overline{1,d} \tag{3}$$

In these formulas the following notations are used: x is a solution vector, d is the number of dimensions, m is the number of components, $c_k^{(t)}$ is the vector of center positions of the k-th component at time t, $R_k^{(t)}$ is the rotational matrix of component k in the environment t, $W_k^{(t)}$ is a width matrix (it is a $d \times d$ diagonal matrix where each diagonal element is the width of the component k), and, finally, $\eta_{k,l}^{(t)}$, $l = \overline{1,4}$, and $\tau_k^{(t)}$ are the irregularity parameters of the component k. Here, for each component, the height, width, irregularity, and all other parameters change as soon as the environmental change happens.

The mentioned function (1) can be used with basic parameters, thus, in the simplest form: in the case where the generated DOP would be symmetric, unimodal, and smooth, the result would be easily optimized. To make the generated problems more complex, the irregularity parameters $\eta_{k,l}^{(t)}$ and $\tau_k^{(t)}$ should be changed. By setting different values to those parameters the end-user can get irregular and/or multimodal optimization tasks.

The higher values of the irregularity parameters ($\eta_{k,l}^{(t)}$ and $\tau_k^{(t)}$) increase the number of local optima in a given peak. It should be noted that identical values of the irregularity parameters $\eta_{k,l}^{(t)}$ lead to the symmetric DOPs, while the components with different $\eta_{k,l}^{(t)}$ values are asymmetric. Additionally, each component's intensity of irregularities, number of local optima, and asymmetry degree change over time due to the fact that parameters $\eta_{k,l}^{(t)}$ and $\tau_k^{(t)}$ change over time.

The rotation matrix $R_k^{(t)}$ is obtained for each component k by rotating the projection of solution x onto all existing unique planes of the search space by a given angle. For that purpose, a Givens rotation matrix is constructed, which is firstly initialized as an identity matrix and then altered. Besides, it should be noted that the initial rotation matrix is obtained by using the Gram-Schmidt orthogonalization method on a matrix with normally distributed entries.

For DOPs obtained by the MPB generator, the width of each component or peak is, in other words, the same in all dimensions, therefore, the shape of components is cone-like with circular contour lines. It was changed for the GMPB benchmark generator, namely, each peak's width was changed from a scalar variable to a vector with d dimensions. Thus, a component generated by GMPB has a width value in each dimension.

The condition number of a component is the ratio of its largest width value to its smallest value, and if a component's width value is stretched in one axis's direction more than the other axes, then, the component is ill-conditioned. As result, the ill-conditioning degree of each component can be determined by calculating the condition number of the diagonal matrix $W_k^{(t)}$. So, the GMPB generator, unlike the MPB generator, is capable of creating components with various condition numbers, i.e., it can additionally generate ill-conditioned peaks.

The modularity can be obtained by composing several sub-functions generated by the GMPB (in that case each sub-function is obtained from the baseline function by varying its parameters listed above). Each sub-function in composition can have a different number of peaks and dimensions, thus, the landscapes of sub-functions can have different features, and, as result, the generated compositional function can be heterogeneous. Additionally, compositional DOPs generated by the GMPB have a lot of local optima in their landscape, which can change to the global optimum after environmental changes.

2.3. mQSO Algorithm

The multi-swarm Quantum Swarm Optimization algorithm is a well-known approach for dynamic optimization proposed in [13]. The main features of the mQSO are the usage of several swarms at the same time, application of exclusion, anti-convergence, and charged and quantum swarms. The swarm is divided into several sub-swarms with the aim to let every small swarm seek and track different local optima. However, a simple division into several small swarms is not enough: if there will be no interaction between sub-swarms, then some of them may converge to the same local optimum.

In mQSO, two forms of swarm interaction are applied: exclusion and anti-convergence. The exclusion mechanism makes sure that two or more swarms are not clustering around the same peak. To prevent this, a competition between swarms is used, in particular, when their swarm attractors, i.e., best solutions, are within the exclusion radius r_{excl}, the swarm, which is further from the optimum, is excluded.

The anti-convergence mechanism works as follows: when all of the swarms have converged to their corresponding local optima, the worst swarm is reinitialized, thus, some part of the total population is always searching for new local optima. The anti-convergence implements global information sharing between all swarms. The convergence to the local optimum is detected when the neutral swarm size is smaller than the predefined convergence radius r_{conv}.

The mQSO starts by randomly initializing N_s swarms of N_p particles each within the D-dimensional search space with positions x_{nij}, $n = 1, 2, \ldots, N_s$, $i = 1, 2, \ldots, N_p$, $j = 1, 2, \ldots, D$. In the implementation in this study, the mQSO uses two types of particles: neutral and quantum, but not charged, as used in [10]. The update rule for the neutral particles is as follows:

$$\begin{gathered} \vec{u_{ni}} \leftarrow w\left[\vec{u_{ni}} + c_1\vec{\varepsilon_1} \odot \left(\vec{p_{ng}} - \vec{x_{ni}}\right) + c_2\vec{\varepsilon_2} \odot \left(\vec{p_{ni}} - \vec{x_{ni}}\right)\right], \\ \vec{x_{ni}} = \vec{x_{ni}} + \vec{u_{ni}}, \end{gathered} \tag{4}$$

where $i = 1, 2, \ldots, N_p$ is the particle index in the population, $\vec{u_{ni}}$ is the velocity of i-th particle in n-th swarm, $n = 1, 2, \ldots, N_s$, $\vec{x_{ni}}$ is current particle position, $\vec{p_{ni}}$ is the personal best position, $\vec{\varepsilon_1}$ and $\vec{\varepsilon_2}$ are random vectors in $[0, 1]^D$, c_1 and c_2 are social and cognitive parameters, w is the inertia factor, p_{ng}—is the global best solution, $\odot$ denotes the element-

wise product. The quantum particles in mQSO are sampled within the D-dimensional ball of radius r_{cloud} around the best particle as follows:

$$\vec{x_{ni}} \in B_n(r_{cloud}). \tag{5}$$

The quantum local search is performed for N_q steps for all the best solutions $\vec{p_{ng}}$ in each swarm n.

2.4. Proposed Algorithm

In this study, the modification of the mQSO algorithm is proposed, in which the search operators from the differential evolution are applied. One of the most common DE search operators is the rand/1, in which the position of the individual is updated based on the position of three randomly chosen individuals with indexes r_1, r_2, and r_3, and the scaling factor F. In this study this operator is applied as follows:

$$\vec{v_{ni}} = \vec{x_{nr_1}} + F\left(\vec{x_{nr_2}} - \vec{x_{nr_3}}\right), \tag{6}$$

where F is the scaling factor parameter, usually within the [0, 1] interval, and $\vec{v_{ni}}$ is the mutant solution for individual i in swarm n.

As some of the previous studies have shown, the DE is not very efficient compared to PSO in optimizing dynamically changing environments. This is mainly due to the much faster convergence of PSO-like algorithms. However, they often suffer from premature convergence, which is usually not the case for DE-based algorithms. Thus, in this study, the rand/1 operator is applied to the particles' positions with a small probability of P_{DE}. There are two possible options for applying the rand/1 strategy within PSO, namely applying it to the current positions of particles, as in Equation (6), or to the personal best positions. In other words, the following equation can be applied instead:

$$\vec{v_{ni}} = \vec{p_{nr_1}} + F\left(\vec{p_{nr_2}} - \vec{p_{nr_3}}\right). \tag{7}$$

Setting the proper value for the scaling factor F is one of the key problems in parameter adaptation methods for DE, as the method is highly sensitive to this value. In this study, the scaling factor parameter F was sampled using the Cauchy distribution with location parameter mF and scale parameter 0.1 (denoted as $randc(mF, 0.1)$). The F value was generated until it fell within the [0, 1] interval. The usage of Cauchy distribution was originally proposed in the JADE [39] algorithm, where F values were tuned based on their successful application. Although the success-based adaptation of scaling factors is not considered in this study, sampling F values should allow for improving the search by increasing the diversity of generated trial solutions. The mF parameter, used as a location for Cauchy distribution, could be set to any value within [0, 1] to get different search properties.

In differential evolution the mutation step is followed by crossover, in particular, the binomial crossover is usually applied with probability Cr. The binomial crossover is described as follows:

$$u_{nij} = \begin{cases} v_{nij} \; if \; rand(0,1) < Cr \; or \; j = jrand \\ x_{nij} \; otherwise \end{cases}, \tag{8}$$

where $jrand$ is a randomly chosen index from $[1, D]$.

After crossover, in classical DE the selection operation is performed, where new solutions are remembered only if their fitness is better than the fitness of the corresponding parent. Unlike DE, in the proposed mQSODE algorithm the newly generated positions are always remembered independent of the fitness values.

The pseudocode of the proposed mQSODE algorithm is shown in Figure 1.

Set algorithm parameters N_s, N_p, mF, Cr, P_{DE}, r_{conv}, r_{excl}
Initialize N_s swarms of N_p particles $\overrightarrow{x_{ni}}$
Calcualte fitness $f(\overrightarrow{x_{ni}})$ and update personal best $\overrightarrow{p_{ni}}$
Determine best particle
While not terminated **do**
  For $n = 1 \dots N_s$
    For $i = 1 \dots N_p$
      If$(rand(0,1) < P_{DE})$
        Sample F from Cauchy distribution: $F = randc(mF, 0.1)$
        Generate $\overrightarrow{v_{ni}}$ with difference-based mutation (Equation (6) or (7))
        Perform crossover with probability Cr to get $\overrightarrow{u_{ni}}$ (Equation (8))
        Set $\overrightarrow{x_{ni}} \leftarrow \overrightarrow{u_{ni}}$
      Else
        Update particle velocity and position with (Equation (4))
      End
    End
    Check boundary conditions for all particles
    Calcualte fitness $f(\overrightarrow{x_{ni}})$ and update personal best $\overrightarrow{p_{ni}}$
    Perform local quantum search around $\overrightarrow{p_{ng}}$ (eq. 5)
  End
  For $n = 1 \dots N_s$
    For $m = 1 \dots N_s$
      If$(dist(\overrightarrow{p_{ng}}, \overrightarrow{p_{mg}}) < r_{excl})$
        If$(f(\overrightarrow{p_{ng}}) > f(\overrightarrow{p_{mg}}))$
          Reinitialize m-th swarm
        Else
          Reinitialize n-th swarm
        End
      End
    End
  End
  For $n = 1 \dots N_s$
    Calculate $|S_n| \leftarrow \max_{kl} \max_j |(\overrightarrow{x_{nk}} - \overrightarrow{x_{nl}}) * \overrightarrow{e_j}|$
    If$(|S_n| < r_{conv})$
      Mark n-th swarm as converged
    End
  End
  Reinitialize worst converged swarm
  If environment has changed
    Update $f(\overrightarrow{x_{ni}})$ values, personal and global best for each swarm
  End

Figure 1. Pseudocode of the proposed mQSODE algorithm.

3. Results

3.1. Experimental Setup

The experiments in this study are performed on the GMPB test suite, the parameters are set equal to those in [10] and given in Table 1.

Table 1. The set of Generalized Moving Peaks Benchmark (GMPB) parameters.

Parameter	Symbol	Value
Dimension	d	10
Shift severity	$\widetilde{s}$	2, 4
Number of components	m	10, 25
Angle severity	$\widetilde{\theta}$	$\pi/9$
Height severity	$\widetilde{h}$	7
Width severity	$\widetilde{w}$	1
Irregularity parameter τ severity	$\widetilde{\tau}$	0.05
Irregularity parameter η severity	$\widetilde{\eta}$	2
Search range	$[Lb, Ub]^d$	$[-50, 50]^d$
Height range	$[h_{min}, h_{max}]$	$[30, 70]$
Width range	$[w_{min}, w_{max}]^d$	$[1, 12]^d$
Angle range	$[\theta_{min}, \theta_{max}]$	$[-\pi, \pi]$
Irregularity parameter τ range	$[\tau_{min}, \tau_{max}]$	$[0, 0.4]$
Irregularity parameter η range	$[\eta_{min}, \eta_{max}]$	$[10, 25]$
Initial center position	$c_k^{(0)}$	$U[Lb, Ub]^d$
Initial height	$h_k^{(0)}$	$U[h_{min}, h_{max}]$
Initial width	$w_k^{(0)}$	$U[w_{min}, w_{max}]^d$
Initial angle	$\theta_k^{(0)}$	$U[\theta_{min}, \theta_{max}]$
Initial irregularity parameter τ	$\tau_k^{(0)}$	$U[\tau_{min}, \tau_{max}]$
Initial irregularity parameter η	$\eta_k^{(0)}$	$U[\eta_{min}, \eta_{max}]^d$
Initial rotation matrix	$R_k^{(0)}$	$GS\left(Norm(0,1)^{d \times d}\right)$
Change frequency	$\widetilde{v}$	2500, 5000
Number of Environments	T	100

There settings considered in this study were: a standard computational resource of 5000 function evaluations between changes, decreased to 2500 function evaluations; with the number of peaks increased to 25, and shift severities increased from 2 to 4. The settings are provided in Table 2.

Table 2. Tested settings.

Parameter	Symbol	Setting 1	Setting 2	Setting 3	Setting 4
Shift severities	$\widetilde{s}$	2	4	2	2
Numbers of components	m	10	10	25	10
Change frequency	$\widetilde{v}$	5000	5000	5000	2500

The following algorithms parameters were set for both mQSO and mQSODE algorithms:

- Number of swarms $N_s = 10$ (independent of the number of components),
- Number of quantum points $N_q = 5$,
- $w = 0.7298$, $c_1 = 2.05$ and $c_2 = 2.05$,
- $r_{cloud} = 2$, $r_{conv} = r_{excl} = 0.5(Ub - Lb)/N_s^d$
- Number of algorithm runs: 31.

The number of individuals in each swarm N_p, probability of applying differential mutation and binomial crossover P_{DE}, type of mutation operator, mF and Cr were varied in the experiments:

- $N_p = \{5, 9, 13, 17, 21, 25, 29, 33\}$
- $P_{DE} = \{0.1, 0.2, 0.3\}$
- Type of mutation: based on current points and based on personal best
- $mF = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$
- $Cr = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$

The tested variants further in the text are marked as mQSO_{N_p}, $\text{mQSODE}^{mF,Cr}_{N_p,P_{DE},x}$ for mutation based on current positions, and $\text{mQSODE}^{mF,Cr}_{N_p,P_{DE},p}$ for mutation based on personal best positions. If the upper indexes for mQSODE are missing, e.g., $\text{mQSODE}_{N_p,P_{DE},x}$, then $mF = 0.3$, $Cr = 1$.

There were two performance indicators used in this study: E_{BCC}, and E_O. The E_{BCC} is calculated as follows:

$$E_{BCC} = \frac{1}{T\widetilde{v}} \sum_{t=1}^{T} \sum_{\varphi=1}^{\widetilde{v}} \left(f^{(t)}(\vec{x}^{\circ(t)}) - f^{(t)}(\vec{x}^{*(t)}) \right), \tag{9}$$

where $\vec{x}^{\circ(t)}$ is the optimum position at time t, and $\vec{x}^{*(t)}$ is the best-found position. The offline error E_O is calculated as the average error of the best-found position over all fitness evaluations:

$$E_O = \frac{1}{T\widetilde{v}} \sum_{t=1}^{T} \sum_{\varphi=1}^{\widetilde{v}} \left(f^{(t)}(\vec{x}^{\circ(t)}) - f^{(t)}(\vec{x}^{*(t-1)\widetilde{v}+\varphi}) \right) \tag{10}$$

In the next subsection, the results of sensitivity analysis to the mentioned parameters are provided.

3.2. Results of Computational Experiments

In the first series of experiments, the influence of the population size on the performance of the mQSO algorithm is determined. In Table 3 the mean and standard deviations of the performance indicators are shown. The best results in Tables 3–9 are highlighted in bold case.

Table 3. Comparison of mQSO with different population sizes.

Algorithm	Setting 1		Setting 2	
	E_{BBC}	E_O	E_{BBC}	E_O
mQSO_5	25.53 ± 5.54	29.72 ± 5.94	32.67 ± 5.39	39.09 ± 5.67
mQSO_9	14.29 ± 3.07	18.05 ± 3.18	17.97 ± 3.53	24.11 ± 3.60
mQSO_{13}	11.71 ± 1.89	15.47 ± 1.98	15.22 ± 2.67	21.27 ± 2.80
mQSO_{17}	10.16 ± 1.81	14.03 ± 1.94	13.30 ± 2.25	19.38 ± 2.46
mQSO_{21}	9.40 ± 2.21	13.18 ± 2.21	12.60 ± 1.52	18.60 ± 1.71
mQSO_{25}	9.12 ± 1.58	13.13 ± 1.77	$\mathbf{12.31 \pm 1.60}$	$\mathbf{18.47 \pm 1.75}$
mQSO_{29}	9.00 ± 1.64	12.99 ± 1.76	12.66 ± 1.90	18.85 ± 2.19
mQSO_{33}	$\mathbf{8.70 \pm 1.29}$	$\mathbf{12.76 \pm 1.43}$	12.55 ± 1.50	18.86 ± 1.68
Algorithm	Setting 3		Setting 4	
	E_{BBC}	E_O	E_{BBC}	E_O
mQSO_5	24.36 ± 3.40	28.09 ± 3.68	27.21 ± 5.82	31.86 ± 6.23
mQSO_9	14.22 ± 1.97	17.68 ± 2.08	16.96 ± 2.84	21.34 ± 2.99
mQSO_{13}	11.83 ± 1.34	15.31 ± 1.41	14.42 ± 2.18	18.83 ± 2.36
mQSO_{17}	10.43 ± 1.21	13.95 ± 1.26	13.71 ± 2.06	18.39 ± 2.43
mQSO_{21}	10.06 ± 1.28	13.54 ± 1.35	13.65 ± 2.00	18.33 ± 2.33
mQSO_{25}	9.57 ± 1.17	13.19 ± 1.24	13.34 ± 1.63	$\mathbf{18.08 \pm 2.17}$
mQSO_{29}	$\mathbf{9.26 \pm 1.13}$	$\mathbf{12.89 \pm 1.29}$	$\mathbf{13.20 \pm 1.45}$	18.41 ± 2.06
mQSO_{33}	9.38 ± 1.02	13.14 ± 1.11	14.11 ± 1.55	19.11 ± 1.86

Table 4. Comparison of mQSODE with current and personal best positions in mutation.

Algorithm	**Setting 1**		**Setting 2**	
	E_{BBC}	E_O	E_{BBC}	E_O
$mQSODE_{5,0.1,x}$	17.66 ± 3.05	21.34 ± 3.29	22.46 ± 2.02	27.97 ± 1.96
$mQSODE_{9,0.1,x}$	12.47 ± 1.95	15.91 ± 2.14	17.90 ± 2.64	23.20 ± 2.72
$mQSODE_{13,0.1,x}$	12.32 ± 2.01	15.74 ± 1.97	17.75 ± 1.86	22.92 ± 2.03
$mQSODE_{17,0.1,x}$	12.43 ± 1.61	15.77 ± 1.81	18.05 ± 1.80	23.31 ± 1.93
$mQSODE_{21,0.1,x}$	12.87 ± 1.62	16.41 ± 1.85	18.31 ± 1.93	23.57 ± 2.14
$mQSODE_{25,0.1,x}$	13.00 ± 1.41	16.48 ± 1.64	18.35 ± 1.73	23.68 ± 2.01
$mQSODE_{29,0.1,x}$	13.16 ± 1.45	16.72 ± 1.63	18.52 ± 1.76	23.86 ± 1.91
$mQSODE_{33,0.1,x}$	13.31 ± 1.44	16.80 ± 1.68	19.09 ± 1.43	24.35 ± 1.52
$mQSODE_{5,0.1,p}$	18.56 ± 3.49	22.48 ± 3.75	24.40 ± 4.04	30.58 ± 4.17
$mQSODE_{9,0.1,p}$	10.72 ± 2.50	14.35 ± 2.68	14.49 ± 2.19	20.29 ± 2.17
$mQSODE_{13,0.1,p}$	9.23 ± 1.77	12.91 ± 1.85	12.87 ± 2.05	18.56 ± 2.23
$mQSODE_{17,0.1,p}$	8.61 ± 1.13	**12.05 ± 1.19**	12.58 ± 1.41	18.14 ± 1.65
$mQSODE_{21,0.1,p}$	8.50 ± 1.26	12.18 ± 1.39	**12.35 ± 1.21**	**18.05 ± 1.28**
$mQSODE_{25,0.1,p}$	8.52 ± 0.95	12.25 ± 1.03	12.85 ± 1.14	18.46 ± 1.35
$mQSODE_{29,0.1,p}$	**8.41 ± 0.90**	12.19 ± 1.03	13.13 ± 0.97	18.80 ± 1.11
$mQSODE_{33,0.1,p}$	8.84 ± 1.05	12.60 ± 1.13	13.19 ± 0.91	18.82 ± 1.14
Algorithm	Setting 3		Setting 4	
	E_{BBC}	E_O	E_{BBC}	E_O
$mQSODE_{5,0.1,x}$	17.97 ± 2.45	21.41 ± 2.62	23.33 ± 4.21	27.77 ± 4.60
$mQSODE_{9,0.1,x}$	13.48 ± 1.99	16.75 ± 2.11	18.73 ± 2.60	22.81 ± 2.68
$mQSODE_{13,0.1,x}$	13.08 ± 1.57	16.17 ± 1.67	18.55 ± 2.61	22.74 ± 2.84
$mQSODE_{17,0.1,x}$	13.58 ± 1.51	16.81 ± 1.58	19.00 ± 2.23	23.21 ± 2.65
$mQSODE_{21,0.1,x}$	13.91 ± 1.83	17.16 ± 2.07	18.78 ± 2.14	22.94 ± 2.22
$mQSODE_{25,0.1,x}$	13.94 ± 1.47	17.22 ± 1.56	19.33 ± 2.01	23.80 ± 2.55
$mQSODE_{29,0.1,x}$	14.47 ± 1.52	17.82 ± 1.59	20.12 ± 2.14	24.84 ± 2.54
$mQSODE_{33,0.1,x}$	14.30 ± 1.28	17.58 ± 1.41	20.60 ± 2.03	25.34 ± 2.30
$mQSODE_{5,0.1,p}$	17.96 ± 2.16	21.54 ± 2.17	22.16 ± 4.74	26.66 ± 5.08
$mQSODE_{9,0.1,p}$	11.24 ± 1.70	14.59 ± 1.71	15.39 ± 2.26	19.81 ± 2.48
$mQSODE_{13,0.1,p}$	9.61 ± 1.32	12.97 ± 1.35	13.60 ± 2.41	**17.68 ± 2.68**
$mQSODE_{17,0.1,p}$	9.50 ± 1.36	13.00 ± 1.46	**13.19 ± 1.65**	17.68 ± 1.88
$mQSODE_{21,0.1,p}$	**9.09 ± 1.06**	**12.48 ± 1.10**	13.48 ± 1.39	17.75 ± 1.65
$mQSODE_{25,0.1,p}$	9.65 ± 1.10	13.05 ± 1.18	14.19 ± 1.38	18.73 ± 1.94
$mQSODE_{29,0.1,p}$	9.66 ± 0.88	13.06 ± 0.94	14.73 ± 1.76	19.73 ± 2.27
$mQSODE_{33,0.1,p}$	10.05 ± 0.91	13.49 ± 0.99	15.39 ± 1.69	19.97 ± 2.11

As can be seen from Table 3, the GMPB test suite seems to favor larger population sizes, i.e., the best results were achieved with more than 20 individuals in each of the swarms. The reason for this could be that, unlike MPB, GMPB generates more complex terrains of the goal function after every change, and larger populations with increased diversity are able to locate better local optima. The best results were achieved by $mQSO_{29}$.

In Table 4, the mQSODE algorithm is tested with $P_{DE} = 0.1$ and two different types of mutation: using current and personal best positions of each particle.

The obtained results shown in Table 4 demonstrate that using the personal best positions of each particle instead of the current position allows for achieving significantly better results across different population sizes. The reason for such behavior could be that the current positions of particles are changing with every iteration, while personal best positions are rather fixed and contain some important information about the problem at hand, such as the location of local optima. Utilizing this information with differential search operators promotes exploration and moving particles in more preferable directions. The best results were achieved by $mQSODE_{21,0.1,p}$.

Table 5. Comparison of mQSODE, with $P_{DE} = 0.2$ and $P_{DE} = 0.3$.

Algorithm	Setting 1		Setting 2	
	E_{BBC}	E_O	E_{BBC}	E_O
$mQSODE_{5,0.2,p}$	18.16 ± 2.43	22.01 ± 2.61	23.83 ± 3.25	30.06 ± 3.42
$mQSODE_{9,0.2,p}$	10.41 ± 1.81	14.15 ± 2.10	14.23 ± 1.71	20.02 ± 1.72
$mQSODE_{13,0.2,p}$	**8.75 ± 1.08**	**12.25 ± 1.21**	13.06 ± 1.33	18.72 ± 1.43
$mQSODE_{17,0.2,p}$	8.82 ± 1.79	12.41 ± 1.91	**13.01 ± 1.43**	**18.57 ± 1.53**
$mQSODE_{21,0.2,p}$	9.13 ± 1.37	12.80 ± 1.52	13.23 ± 1.25	18.82 ± 1.53
$mQSODE_{25,0.2,p}$	8.82 ± 0.82	12.35 ± 1.06	13.85 ± 1.35	19.35 ± 1.57
$mQSODE_{29,0.2,p}$	9.36 ± 1.12	12.93 ± 1.29	14.09 ± 1.27	19.42 ± 1.45
$mQSODE_{33,0.2,p}$	9.81 ± 1.36	13.49 ± 1.46	14.57 ± 0.97	19.99 ± 1.17
$mQSODE_{5,0.3,p}$	19.80 ± 3.94	24.01 ± 4.40	25.93 ± 4.16	32.53 ± 4.37
$mQSODE_{9,0.3,p}$	10.78 ± 1.73	14.68 ± 1.84	15.31 ± 2.51	21.45 ± 2.70
$mQSODE_{13,0.3,p}$	9.51 ± 1.50	13.14 ± 1.70	13.62 ± 1.63	19.48 ± 1.74
$mQSODE_{17,0.3,p}$	8.97 ± 1.17	12.56 ± 1.31	13.83 ± 1.26	19.57 ± 1.45
$mQSODE_{21,0.3,p}$	9.25 ± 1.38	12.89 ± 1.54	14.05 ± 1.46	19.62 ± 1.72
$mQSODE_{25,0.3,p}$	9.45 ± 0.97	13.13 ± 1.23	14.38 ± 1.03	19.91 ± 1.16
$mQSODE_{29,0.3,p}$	9.66 ± 0.93	13.31 ± 1.09	14.95 ± 1.09	20.29 ± 1.19
$mQSODE_{33,0.3,p}$	10.34 ± 1.17	13.93 ± 1.30	15.25 ± 1.44	20.88 ± 1.71
Algorithm	Setting 3		Setting 4	
	E_{BBC}	E_O	E_{BBC}	E_O
$mQSODE_{5,0.2,p}$	18.76 ± 2.64	22.54 ± 2.74	21.69 ± 3.12	26.17 ± 3.25
$mQSODE_{9,0.2,p}$	10.95 ± 1.40	14.44 ± 1.42	14.40 ± 1.72	18.90 ± 2.16
$mQSODE_{13,0.2,p}$	**9.76 ± 1.48**	**13.16 ± 1.54**	**14.10 ± 2.31**	**18.53 ± 2.57**
$mQSODE_{17,0.2,p}$	9.97 ± 1.11	13.32 ± 1.09	14.76 ± 1.41	18.97 ± 1.49
$mQSODE_{21,0.2,p}$	9.91 ± 0.97	13.22 ± 1.05	14.67 ± 2.07	18.98 ± 2.25
$mQSODE_{25,0.2,p}$	9.97 ± 0.89	13.25 ± 1.01	15.17 ± 1.63	19.44 ± 2.00
$mQSODE_{29,0.2,p}$	10.61 ± 0.92	13.89 ± 1.03	15.85 ± 1.71	20.18 ± 2.02
$mQSODE_{33,0.2,p}$	11.06 ± 0.98	14.36 ± 1.08	17.16 ± 1.96	21.90 ± 2.36
$mQSODE_{5,0.3,p}$	19.00 ± 2.95	22.79 ± 3.23	23.22 ± 2.82	27.98 ± 3.14
$mQSODE_{9,0.3,p}$	11.92 ± 1.82	15.47 ± 1.88	16.19 ± 2.54	20.56 ± 2.61
$mQSODE_{13,0.3,p}$	10.29 ± 1.35	13.69 ± 1.38	14.91 ± 2.04	18.98 ± 2.34
$mQSODE_{17,0.3,p}$	10.33 ± 1.17	13.75 ± 1.28	14.67 ± 1.37	19.01 ± 1.53
$mQSODE_{21,0.3,p}$	10.16 ± 1.16	13.50 ± 1.29	15.09 ± 1.56	19.44 ± 1.96
$mQSODE_{25,0.3,p}$	10.87 ± 0.88	14.21 ± 1.01	15.96 ± 1.37	20.33 ± 1.52
$mQSODE_{29,0.3,p}$	11.02 ± 0.73	14.35 ± 0.90	16.95 ± 1.70	21.15 ± 1.99
$mQSODE_{33,0.3,p}$	11.53 ± 0.92	14.84 ± 1.06	17.45 ± 1.67	21.81 ± 2.02

Table 5 contains the comparison of mQSODE with increased P_{DE} probabilities, the personal best positions are used in mutation.

The results in Table 5 show that increasing the probability of using the DE search operator decreases the performance of mQSODE, and $P_{DE} = 0.3$ is worse than $P_{DE} = 0.2$ and, compared to Table 4, $P_{DE} = 0.2$ is worse than $P_{DE} = 0.1$. This means that switching the main search mechanism to DE is not desirable for dynamic environments, and the application of DE should be assisting PSO, but not replacing it. It is also important to mention that the best results for the case of $P_{DE} = 0.2$ were achieved with a smaller number of particles than with $P_{DE} = 0.1$, which means that more often applications of DE increase the diversity of individual swarms so that smaller swarm sizes are sufficient.

The sensitivity of mQSODE to crossover rates and scaling factor values were tested and obtained results are demonstrated in Tables 6 and 7.

The comparison of performance metric values in Tables 6 and 7 shows that DE search parameters do not have a significant influence on the final performance, however, the influence of scaling factor sampling parameter mF is larger than the influence of crossover rate Cr. The best performing configuration here is $mQSODE^{0.3,1.0}_{21,0.1,p}$.

Table 6. Comparison of mQSODE with different crossover rates.

Algorithm	Setting 1		Setting 2	
	E_{BBC}	E_O	E_{BBC}	E_O
$mQSODE^{0.3,0.1}_{21,0.1,p}$	8.70 ± 1.70	12.38 ± 1.79	12.39 ± 1.52	17.95 ± 1.66
$mQSODE^{0.3,0.2}_{21,0.1,p}$	8.35 ± 1.26	12.00 ± 1.36	12.41 ± 1.38	18.05 ± 1.69
$mQSODE^{0.3,0.3}_{21,0.1,p}$	8.29 ± 1.40	11.94 ± 1.44	12.38 ± 1.24	18.06 ± 1.47
$mQSODE^{0.3,0.4}_{21,0.1,p}$	8.63 ± 1.77	12.26 ± 1.89	12.37 ± 1.42	18.02 ± 1.48
$mQSODE^{0.3,0.5}_{21,0.1,p}$	8.49 ± 1.21	12.14 ± 1.25	12.17 ± 1.69	17.71 ± 1.76
$mQSODE^{0.3,0.6}_{21,0.1,p}$	8.25 ± 1.10	11.85 ± 1.11	12.06 ± 1.18	17.69 ± 1.35
$mQSODE^{0.3,0.7}_{21,0.1,p}$	8.56 ± 1.34	12.24 ± 1.37	**12.01 ± 1.24**	**17.52 ± 1.31**
$mQSODE^{0.3,0.8}_{21,0.1,p}$	8.46 ± 1.51	12.19 ± 1.56	12.59 ± 1.84	18.25 ± 2.01
$mQSODE^{0.3,0.9}_{21,0.1,p}$	8.50 ± 1.11	12.08 ± 1.31	12.59 ± 1.74	18.31 ± 1.87
$mQSODE^{0.3,1.0}_{21,0.1,p}$	**8.10 ± 0.96**	**11.80 ± 1.02**	12.19 ± 1.24	17.87 ± 1.25
Algorithm	Setting 3		Setting 4	
	E_{BBC}	E_O	E_{BBC}	E_O
$mQSODE^{0.3,0.1}_{21,0.1,p}$	9.11 ± 0.85	12.46 ± 0.83	**13.16 ± 1.33**	17.66 ± 2.01
$mQSODE^{0.3,0.2}_{21,0.1,p}$	9.30 ± 1.08	12.64 ± 1.11	13.17 ± 1.75	**17.25 ± 1.98**
$mQSODE^{0.3,0.3}_{21,0.1,p}$	9.18 ± 0.87	12.51 ± 0.89	13.84 ± 1.53	18.32 ± 2.03
$mQSODE^{0.3,0.4}_{21,0.1,p}$	9.28 ± 1.40	12.62 ± 1.49	13.24 ± 1.56	17.67 ± 1.86
$mQSODE^{0.3,0.5}_{21,0.1,p}$	**8.85 ± 1.14**	**12.13 ± 1.18**	13.55 ± 1.81	17.82 ± 1.87
$mQSODE^{0.3,0.6}_{21,0.1,p}$	9.00 ± 1.08	12.43 ± 1.22	13.53 ± 1.36	17.95 ± 1.48
$mQSODE^{0.3,0.7}_{21,0.1,p}$	9.50 ± 1.42	12.93 ± 1.54	13.51 ± 1.51	17.69 ± 1.85
$mQSODE^{0.3,0.8}_{21,0.1,p}$	9.21 ± 0.95	12.59 ± 1.04	13.57 ± 1.58	18.32 ± 2.14
$mQSODE^{0.3,0.9}_{21,0.1,p}$	9.30 ± 1.11	12.65 ± 1.18	13.83 ± 1.31	18.15 ± 1.81
$mQSODE^{0.3,1.0}_{21,0.1,p}$	9.12 ± 1.04	12.54 ± 1.12	13.70 ± 1.61	18.21 ± 1.79

Table 8 contains the comparison of two tested algorithms in this study, $mQSO_{29}$ and $mQSODE_{21,0.1,p}$ with alternative approaches on the same benchmark.

Table 8 shows that although the $mQSODE_{21,0.1,p}$ is not capable of outperforming some of the alternative methods, it is still highly efficient compared to PSO-based algorithms, such as FTmPSO and mPSO. In particular, it achieves better results in terms of offline error E_O especially for more challenging settings 3 and 4. It should be noted that unlike the algorithms from [10], in this study the shift severity learning was not performed, and the shift severity value was set to 2, which is used by quantum particles.

Table 9 contains the results of pairwise Mann-Whitney statistical tests applied to mQSO with two population sizes and mQSODE. The statistical tests were performed with a significance level $p = 0.01$, with normal approximation and tie-breaking. The values in Table 3 are the test result (Z-score), where the test result is either -1 (worse), 0 (equal), or 1 (better). For example, if the absolute value of Z was smaller than 2.58, the test result was 0.

With the same population size, the mQSODE algorithms were able to significantly outperform the mQSO in the third setting, i.e., when the number of components is increased to 25. Comparing with $mQSO_{29}$, the $mQSODE_{21,0.1,p}$ always performs better, except for the fourth setting, although these improvements are not significant according to the Mann-Whitney test.

Figure 2 shows the current error graphs of $mQSO_{29}$ and $mQSODE_{21,0.1,p}$ algorithms for four considered settings, and Figure 3 shows the offline error graphs.

Table 7. Comparison of mQSODE, with different scaling factor sampling parameters.

Algorithm	Setting 1		Setting 2	
	E_{BBC}	E_O	E_{BBC}	E_O
$\text{mQSODE}^{0.1,1.0}_{21,0.1,\text{p}}$	**8.09 ± 1.29**	**11.72 ± 1.38**	12.73 ± 1.87	18.43 ± 1.81
$\text{mQSODE}^{0.2,1.0}_{21,0.1,\text{p}}$	8.64 ± 1.45	12.38 ± 1.60	12.28 ± 1.65	17.93 ± 1.80
$\text{mQSODE}^{0.3,1.0}_{21,0.1,\text{p}}$	8.10 ± 0.96	11.80 ± 1.02	12.19 ± 1.24	17.87 ± 1.25
$\text{mQSODE}^{0.4,1.0}_{21,0.1,\text{p}}$	8.23 ± 1.08	11.88 ± 1.16	**12.17 ± 1.39**	**17.70 ± 1.50**
$\text{mQSODE}^{0.5,1.0}_{21,0.1,\text{p}}$	8.42 ± 1.24	12.12 ± 1.40	12.63 ± 1.52	18.20 ± 1.77
$\text{mQSODE}^{0.6,1.0}_{21,0.1,\text{p}}$	8.42 ± 1.42	11.87 ± 1.44	12.90 ± 1.30	18.36 ± 1.48
$\text{mQSODE}^{0.7,1.0}_{21,0.1,\text{p}}$	9.00 ± 1.45	12.48 ± 1.50	13.44 ± 1.49	19.04 ± 1.75
$\text{mQSODE}^{0.8,1.0}_{21,0.1,\text{p}}$	9.21 ± 1.33	12.67 ± 1.36	13.34 ± 1.63	18.65 ± 1.71
$\text{mQSODE}^{0.9,1.0}_{21,0.1,\text{p}}$	9.50 ± 1.78	13.23 ± 2.01	13.78 ± 2.02	19.30 ± 2.12
$\text{mQSODE}^{1.0,1.0}_{21,0.1,\text{p}}$	9.02 ± 1.31	12.58 ± 1.35	13.65 ± 1.70	19.13 ± 1.78
Algorithm	Setting 3		Setting 4	
	E_{BBC}	E_O	E_{BBC}	E_O
$\text{mQSODE}^{0.1,1.0}_{21,0.1,\text{p}}$	9.37 ± 1.30	12.72 ± 1.34	**13.29 ± 1.28**	17.88 ± 1.74
$\text{mQSODE}^{0.2,1.0}_{21,0.1,\text{p}}$	**8.92 ± 0.95**	**12.30 ± 1.02**	13.41 ± 1.74	**17.87 ± 2.07**
$\text{mQSODE}^{0.3,1.0}_{21,0.1,\text{p}}$	9.12 ± 1.04	12.54 ± 1.12	13.70 ± 1.61	18.21 ± 1.79
$\text{mQSODE}^{0.4,1.0}_{21,0.1,\text{p}}$	9.19 ± 1.01	12.61 ± 1.06	13.65 ± 1.39	18.09 ± 1.60
$\text{mQSODE}^{0.5,1.0}_{21,0.1,\text{p}}$	9.45 ± 1.02	12.82 ± 1.08	14.18 ± 1.68	18.92 ± 2.05
$\text{mQSODE}^{0.6,1.0}_{21,0.1,\text{p}}$	9.65 ± 1.40	12.88 ± 1.36	14.56 ± 1.88	19.03 ± 2.16
$\text{mQSODE}^{0.7,1.0}_{21,0.1,\text{p}}$	9.81 ± 1.37	13.16 ± 1.45	14.79 ± 1.63	19.04 ± 2.04
$\text{mQSODE}^{0.8,1.0}_{21,0.1,\text{p}}$	9.63 ± 1.20	12.87 ± .29	14.92 ± 2.08	19.08 ± 2.34
$\text{mQSODE}^{0.9,1.0}_{21,0.1,\text{p}}$	9.80 ± 1.00	12.98 ± 1.03	14.95 ± 1.69	19.26 ± 1.79
$\text{mQSODE}^{1.0,1.0}_{21,0.1,\text{p}}$	10.34 ± .28	13.62 ± 1.23	14.87 ± 1.53	19.03 ± 1.92

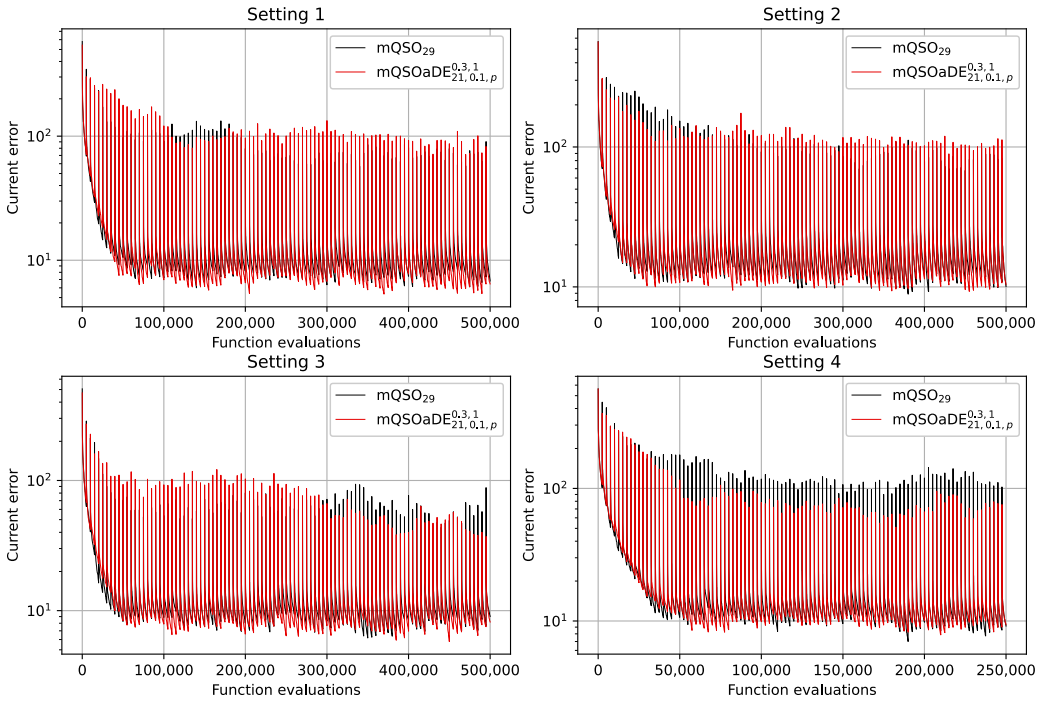

Figure 2. Current error graphs of mQSO_{29} and $\text{mQSODE}_{21,0.1,\text{p}}$ in four benchmark settings.

Table 8. Comparison of mQSO and mQSODE with alternative approaches from [10].

Algorithm	Setting 1		Setting 2	
	E_{BBC}	E_O	E_{BBC}	E_O
$mQSO_{29}$	9.00 ± 1.64	12.99 ± 1.76	12.66 ± 1.90	18.85 ± 2.19
$mQSODE_{21,0.1,p}$	8.50 ± 1.26	12.18 ± 1.39	12.35 ± 1.21	18.05 ± 1.28
FTmPSO	7.11 ± 0.26	12.70 ± 0.34	8.70 ± 0.26	16.06 ± 0.35
mCMA-ES	**5.11 ± 0.22**	**7.59 ± 0.24**	**7.39 ± 0.26**	**11.65 ± 0.28**
mPSO	8.51 ± 0.22	12.88 ± 0.27	11.63 ± 0.19	18.52 ± 0.26
Algorithm	Setting 3		Setting 4	
	E_{BBC}	E_O	E_{BBC}	E_O
$mQSO_{29}$	9.26 ± 1.13	12.89 ± 1.29	13.20 ± 1.45	18.41 ± 2.06
$mQSODE_{21,0.1,p}$	9.09 ± 1.06	12.48 ± 1.10	13.48 ± 1.39	17.75 ± 1.65
FTmPSO	7.99 ± 0.16	13.48 ± 0.21	11.57 ± 0.32	18.04 ± 0.39
mCMA-ES	**6.14 ± 0.15**	**8.49 ± 0.16**	**8.56 ± 0.31**	**11.06 ± 0.33**
mPSO	9.58 ± 0.17	13.86 ± 0.21	12.68 ± 0.27	17.67 ± 0.34

Table 9. Pairwise Mann-Whitney tests of mQSO and mQSODE, all settings, test results, and standard scores.

Measure	Setting 1	Setting 2	Setting 3	Setting 4
		$mQSO_{21}$ vs. $mQSODE_{21,0.1,p}$		
E_O	0 (1.485)	0 (0.993)	1 (3.062)	0 (−0.021)
E_{BBC}	0 (1.696)	0 (1.528)	1 (3.118)	0 (1.035)
		$mQSO_{29}$ vs. $mQSODE_{21,0.1,p}$		
E_O	0 (1.077)	0 (0.598)	0 (0.683)	0 (−0.978)
E_{BBC}	0 (1.936)	0 (1.668)	0 (1.457)	0 (1.316)

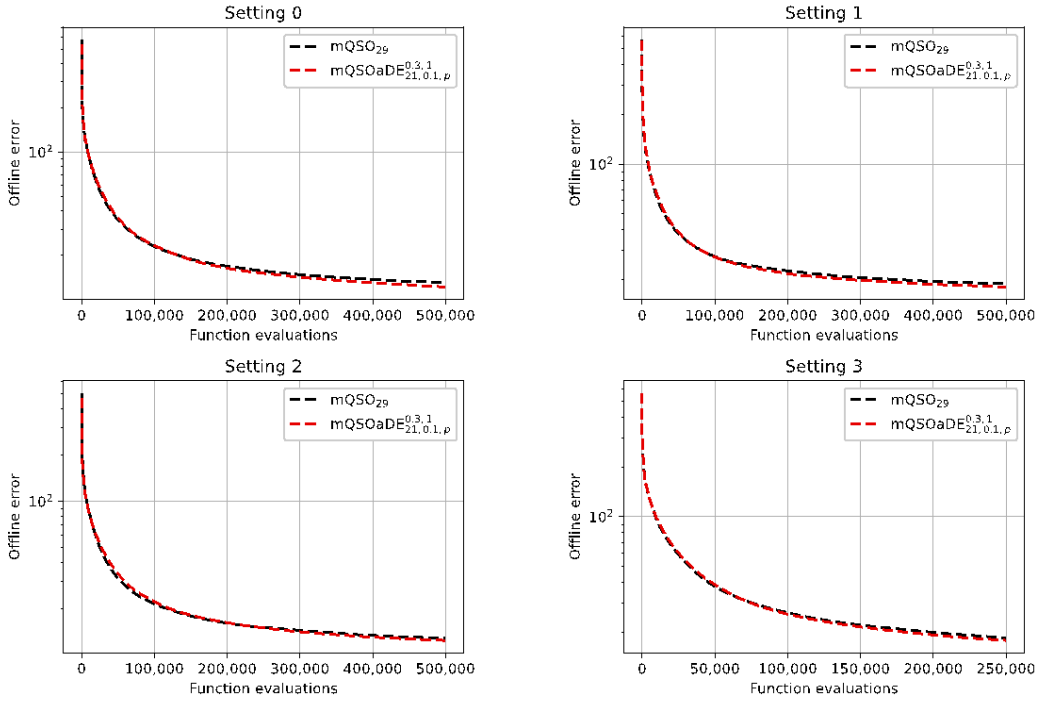

Figure 3. Offline error graphs of three tested algorithms with two benchmark cases: 2000 and 5000 function evaluations between changes.

Analyzing Figures 2 and 3 it can be noted that mQSODE$_{21,0.1,p}$ is more conservative at the beginning, i.e., during the first several environmental changes, while mQSO$_{29}$ achieves better values. However, at the final environmental changes both current and offline errors of the modified algorithm are better, which indicates that it makes a better job tracking the local optima. To analyze the reasons for increased performance in the long term, the diversity measure was used for three algorithms, namely mQSO$_{21}$, mQSO$_{29}$, and mQSODE$_{21,0.1,p}$. The diversity was measured at every iteration as follows:

$$DM = \frac{1}{N_s} \sum_{i=1}^{N_p} \sum_{j=i}^{N_p} \sum_{k=1}^{D} \left(x_{nik} - x_{njk} \right)^2$$

In other words, the DM was calculated as the sum of pairwise distances between all particles in each swarm and averaged over the number of swarms. Figure 4 shows the diversity measures for one of the runs with setting 1 of the three mentioned algorithms.

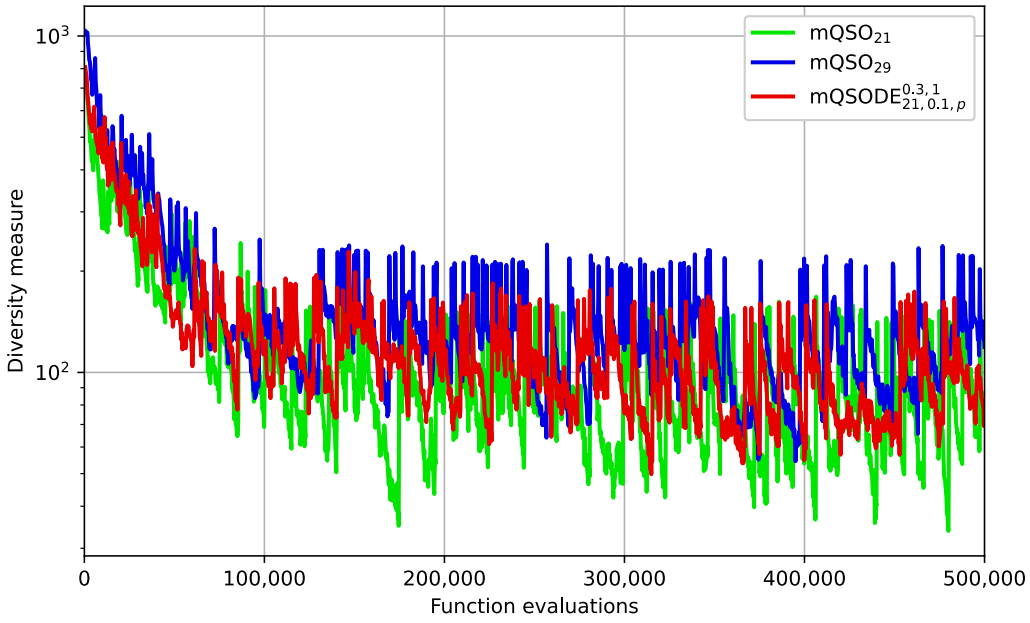

Figure 4. Diversity measures of mQSO$_{21}$, mQSO$_{29}$ and mQSODE$_{21,0.1,p}$, setting 1.

The graphs in Figure 4 show that given the same population size, the algorithm without a DE search operator has significantly smaller diversity, up to several times smaller. When comparing mQSODE$_{21,0.1,p}$ with mQSO$_{29}$, which has a larger population, the diversity of the mQSO after an environmental change is usually larger, however, before the change when the swarms are converged, the diversity measures of these algorithms are comparable, which explains their similar performance.

To compare the efficiency of all the tested modifications, the ranking procedure from the Friedman statistical test was applied, which will further be referred to as the Friedman ranking. To perform such a comparison, all algorithms' results over 31 independent runs were ranked for every benchmark setting and every performance measure. After this, the ranks assigned to each algorithm participating in the comparison were summed together to form a single rank, which is depicted as a bar in a bar plot in Figure 5. Such a procedure allows for aggregating different performance measures and test scenarios in a single measure, highlighting the best approaches.

The presented Friedman ranking in Figure 5 shows that the mQSODE$_{21,0.1,p}$ is the best algorithm among the tested configurations. It also shows that applying larger P_{DE} values are inefficient, as well as using current particles' positions for differential mutation.

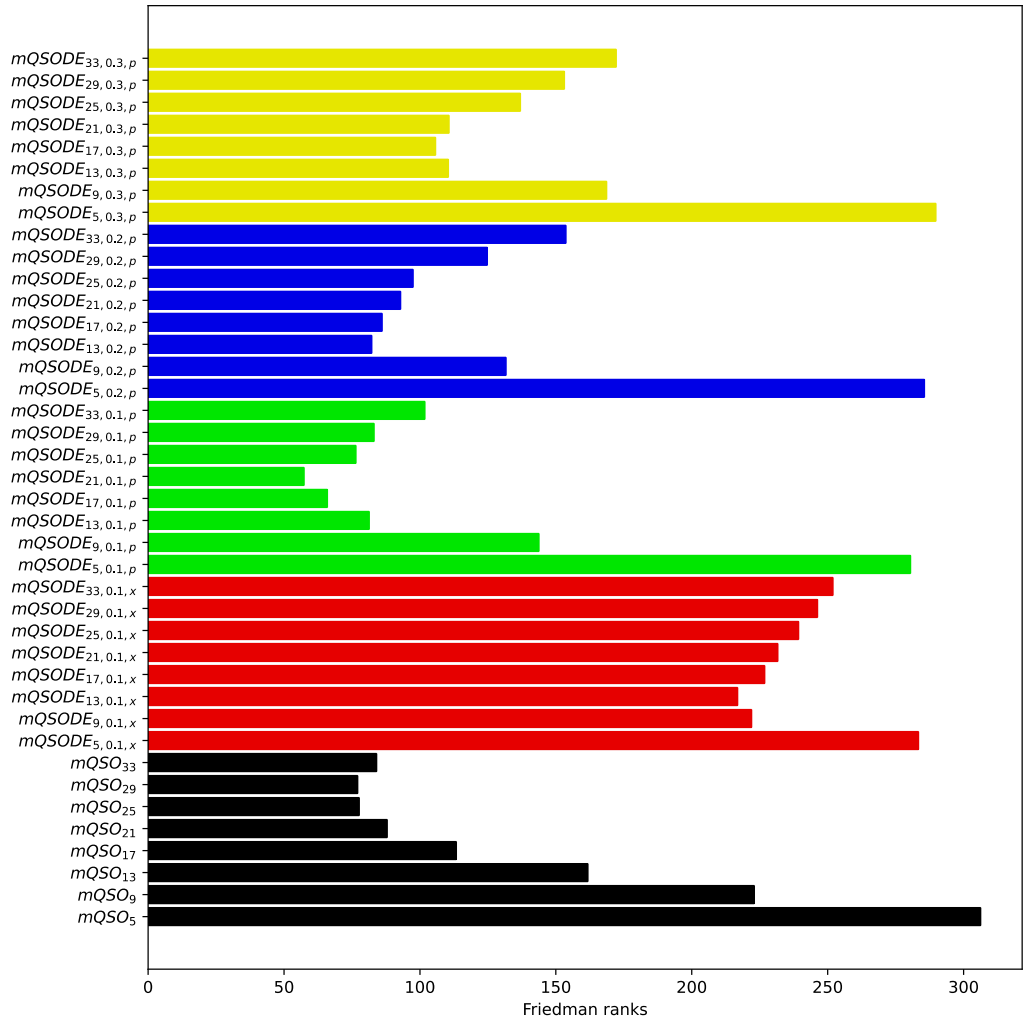

Figure 5. Bar plot Friedman ranks of tested algorithms, aggregated over four settings.

4. Discussion

The performed computational experiments and sensitivity analysis have shown that it is possible to develop an efficient hybrid approach, combining PSO and DE. The key to efficient performance here seems to be in the usage of personal best positions of particles instead of current positions because personal best points contain important information for the search process.

Although within this study only one baseline algorithm was considered, namely mQSO, it is possible to interpolate the proposed algorithmic scheme for any other PSO-based algorithm, and most of the used approaches for dynamic optimization are based on a type of swarm intelligence algorithm derived from particle swarm optimization. Thus, the findings of this study could be considered rather general; however, additional experiments in other scenarios are required to prove the concept.

It is rather obvious that setting the probability of applying DE to a fixed value during the whole algorithm run is not the best possible option, and a specific adaptation mechanism for P_{DE} should be developed. In our preliminary experiments with several adaptation schemes, we were not able to come up with an efficient method that would significantly improve the performance of the GMPB test suite. The reason for that was that DE mutation generates successful solutions more often so allowing the P_{DE} to be adapted simply increases this probability to the upper threshold. This, in turn, results in too high P_{DE} values, which decrease the search efficiency, as was shown in the experiments in this

study. In other words, the adaptation scheme should consider this fact and be less greedy for improvements.

In addition to P_{DE} adaptation, application of DE operators burdens researchers with additional parameters to be tuned, namely the scaling factor F and crossover rate Cr. In this study, it was shown that the sensitivity to these values is not very large, probably thanks to the rotation-invariant nature of PSO and applied sampling of F values from the Cauchy distribution. Nevertheless, incorporating known adaptation schemes, such as success-history adaptation, or developing new specific methods is another direction of further studies.

Finally, it should be mentioned that in this study only one differential mutation strategy was tested, i.e., the classical rand/1, although other schemes are known to be superior in certain cases. If some of them would be shown to perform better than rand/1 for dynamic environments, this will allow making another step in the direction toward efficient DOPs solvers.

5. Conclusions

In this study, the modification of the Quantum Multi-Swarm Optimization algorithm was proposed. The algorithm incorporating the additional search operator from Differential Evolution, mQSODE, has shown to perform better in different challenging settings of the generalized moving peaks benchmark. Thus, in this study, it was shown that although the DE algorithms usually show worse results in dynamically changing environments, the exploration capabilities of their mutation operator can be efficiently used to modify the existing algorithms if they are applied to personal best positions of the points in PSO. Further research directions may include applying other differential search operators to mQSO, developing adaptation schemes, or modifying other algorithms in a similar manner.

Author Contributions: Conceptualization, V.S., S.A. and A.V.; methodology, V.S., S.A., A.V. and E.S. (Evgenii Sopov); software, V.S. and S.A.; validation, A.V., S.A. and E.S. (Eugene Semenkin); formal analysis, E.S. (Evgenii Sopov) and M.A.; investigation, S.A. and A.V.; resources, V.S. and E.S. (Evgenii Sopov); data curation, S.A. and A.V.; writing—original draft preparation, V.S., S.A. and A.V.; writing—review and editing, E.S. (Evgenii Sopov), E.S. (Eugene Semenkin) and M.A.; visualization, V.S.; supervision, E.S. (Eugene Semenkin) and M.A.; project administration, E.S. (Evgenii Sopov) and E.S. (Eugene Semenkin); funding acquisition, E.S. (Eugene Semenkin) and M.A. All authors have read and agreed to the published version of the manuscript.

Funding: The reported study was funded by RFBR and FWF according to the research project №21-51-14003.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yar, M.H.; Rahmati, V.; Oskouei, H.R.D. A Survey on Evolutionary Computation: Methods and Their Applications in Engineerng. *Mod. Appl. Sci.* **2016**, *10*, 131–139. [CrossRef]
2. Nguyen, T.T.; Yang, S.; Branke, J. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm Evol. Comput.* **2012**, *6*, 1–24. [CrossRef]
3. Yazdani, D.; Cheng, R.; Yazdani, D.; Branke, J.; Jin, Y.; Yao, X. A Survey of Evolutionary Continuous Dynamic Optimization Over Two Decades—Part A. *IEEE Trans. Evol. Comput.* **2021**, *25*, 609–629. [CrossRef]
4. Yazdani, D.; Cheng, R.; Yazdani, D.; Branke, J.; Jin, Y.; Yao, X. A Survey of Evolutionary Continuous Dynamic Optimization Over Two Decades—Part B. *IEEE Trans. Evol. Comput.* **2021**, *25*, 630–650. [CrossRef]
5. Elshamli, A.; Abdullah, H.A.; Areibi, S. Genetic algorithm for dynamic path planning. *Can. Conf. Electr. Comput. Eng.* **2004**, *2*, 677–680.
6. Michalewicz, Z.; Schmidt, M.; Michalewicz, M.; Chiriac, C. Adaptive business intelligence: Three case studies. *Stud. Comput. Intell.* **2007**, *51*, 179–196.
7. Kyriakakis, N.A.; Marinaki, M.; Matsatsinis, N.F.; Marinakis, Y. Moving peak drone search problem: An online multi-swarm intelligence approach for UAV search operations. *Swarm Evol. Comput.* **2021**, *66*, 100956. [CrossRef]

8. Branke, J. Memory enhanced evolutionary algorithms for changing optimization problems. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99, Washington, DC, USA, 6–9 July 1999; Volume 3, pp. 1875–1882.
9. Li, C.; Yang, S. A generalized approach to construct benchmark problems for dynamic optimization, in Simulated Evolution and Learning. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 5361, pp. 391–400.
10. Yazdani, B.D.; Omidvar, M.N.; Cheng, R.; Branke, J.; Nguyen, T.; Yao, X. Benchmarking Continuous Dynamic Optimization: Survey and Generalized Test Suite. *IEEE Trans. Cybern.* **2020**, *1*, 1–14. [CrossRef]
11. Yazdani, D.; Branke, J.; Omidvar, M.N.; Li, X.; Li, C.; Mavrovouniotis, M.; Nguyen, T.T.; Yang, S.; Yao, X. *IEEE CEC 2022 Competition on Dynamic Optimization Problems Generated by Generalized Moving Peaks Benchmark*; Technical Report; Southern University of Science and Technology: Shenzhen, China, 2021.
12. Ahrari, A.; Elsayed, S.M.; Sarker, R.A.; Essam, D.L.; Coello, C.A. A Novel Parametric benchmark generator for dynamic multimodal optimization. *Swarm Evol. Comput.* **2021**, *65*, 100924. [CrossRef]
13. Blackwell, T.M.; Branke, J. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Trans. Evol. Comput.* **2006**, *10*, 459–472. [CrossRef]
14. Yazdani, D.; Nasiri, B.; Sepas-Moghaddam, A.; Meybodi, M.R. A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization. *Appl. Soft Comput.* **2013**, *13*, 2144–2158. [CrossRef]
15. Plessis, M.C.; Engelbrecht, A.P. Differential evolution for dynamic environments with unknown numbers of optima. *J. Glob. Optim.* **2013**, *55*, 73–99. [CrossRef]
16. Jia, D. A Culture-Based Artificial Bee Colony Algorithm for Optimization in Dynamic Environments. *J. Adv. Comput. Intell. Intell. Inform.* **2022**, *26*, 23–27. [CrossRef]
17. Hu, X.; Eberhart, R. Adaptive particle swarm optimisation: Detection and response to dynamic systems. In Proceedings of the IEEE Congress on Evolutionary Computation, Honolulu, HI, USA, 12–17 May 2002; pp. 1666–1670.
18. Cobb, H.G. *An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments*; Technical Report AIC-90-001; Naval Research Lab: Washington, DC, USA, 1990.
19. Riekert, M.; Malan, K.M.; Engelbrecht, A.P. Adaptive genetic programming for dynamic classification problems. In Proceedings of the IEEE Congress on Evolutionary Computation, Trondheim, Norway, 18–21 May 2009; pp. 674–681.
20. Grefenstette, J.J. Genetic algorithms for changing environments. *Parallel Probl. Solving Nat.* **1992**, *2*, 137–144.
21. Morrison, R.W. *Designing Evolutionary Algorithms for Dynamic Environments*; Springer: Berlin/Heidelberg, Germany, 2004.
22. Daneshyari, M.; Yen, G. Dynamic optimization using cultural based PSO. In Proceedings of the IEEE Congress on Evolutionary Computation, New Orleans, LA, USA, 5–8 June 2011; pp. 509–516.
23. Simoes, A.; Costa, E. Memory-based CHC algorithms for the dynamic—Traveling salesman problem. In Proceedings of the Genetic and Evolutionary Computation Conference, New York, NY, USA, 12–16 July 2011; pp. 1037–1044.
24. Hatzakis, I.; Wallace, D. Dynamic multi-objective optimization with evolutionary algorithms: A forward-looking approach. In Proceedings of the Genetic and Evolutionary Computation Conference, New York, NY, USA, 8–12 July 2006; pp. 1201–1208.
25. Oppacher, F.; Wineberg, M. The shifting balance genetic algorithm: Improving the GA in a dynamic environment. In Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, 13–17 July 1999; Volume 1, pp. 504–510.
26. Ursem, R.K. Multinational GA optimization techniques in dynamic environments. In Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, 10–12 July 2000; pp. 19–26.
27. Qin, J.; Huang, C.; Luo, Y. Adaptive multi-swarm in dynamic environments. *Swarm Evol. Comput.* **2021**, *63*, 100870. [CrossRef]
28. Li, C.; Nguyen, T.; Yang, M.; Mavrovouniotis, M.; Yang, S. An Adaptive Multipopulation Framework for Locating and Tracking Multiple Optima. *IEEE Trans. Evol. Comput.* **2016**, *20*, 590–605. [CrossRef]
29. Tao, X.; Guo, W.; Li, X.; He, Q.; Liu, R.; Zou, J. Fitness peak clustering based dynamic multi-swarm particle swarm optimization with enhanced learning strategy. *Expert Syst. Appl.* **2022**, *191*, 116301. [CrossRef]
30. Li, C.; Yang, S.; Yang, M. An Adaptive Multi-Swarm Optimizer for Dynamic Optimization Problems. *Evol. Comput.* **2014**, *22*, 559–594. [CrossRef]
31. Xia, X.; Tang, Y.; Wei, B.; Zhang, Y.; Gui, L.; Li, X. Dynamic multi-swarm global particle swarm optimization. *Computing* **2020**, *102*, 1587–1626. [CrossRef]
32. Xia, X.; Tang, Y.; Wei, B.; Gui, L. Dynamic Multi-Swarm Particle Swarm Optimization Based on Elite Learning. *IEEE Access* **2019**, *7*, 184849–184865. [CrossRef]
33. Awad, N.H.; Ali, M.Z.; Liang, J.J.; Qu, B.Y.; Suganthan, P.N. *Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Operation*; Technical Report; Nanyang Technological University: Singapore, 2016.
34. Song, L.; Shi, J.; Pan, A.; Yang, J.; Xie, J. A Dynamic Multi-Swarm Particle Swarm Optimizer for Multi-Objective Optimization of Machining Operations Considering Efficiency and Energy Consumption. *Energies* **2020**, *13*, 2616. [CrossRef]
35. Zhao, Q.; Li, C. Two-Stage Multi-Swarm Particle Swarm Optimizer for Unconstrained and Constrained Global Optimization. *IEEE Access* **2020**, *8*, 124905–124927. [CrossRef]
36. Sedki, A.; Ouazar, D. Hybrid particle swarm optimization and differential evolution for optimal design of water distribution systems. *Adv. Eng. Inform.* **2012**, *26*, 582–591. [CrossRef]
37. Lynn, N.; Ali, M.; Suganthan, P.N. Population topologies for particle swarm optimization and differential evolution. *Swarm Evol. Comput.* **2018**, *39*, 24–35. [CrossRef]

38. Akhmedova, S.; Stanovov, V.; Semenkin, E. Soft island model for population-based optimization algorithms. In *International Conference on Swarm Intelligence*; Springer: Cham, Switzerland, 2018; pp. 68–77.
39. Zhang, J.; Sanderson, A.C. JADE: Self-adaptive differential evolution with fast and reliable convergence performance. In Proceedings of the IEEE Congress on Evolutionary Computation, Singapore, 25–28 September 2007; pp. 2251–2258.