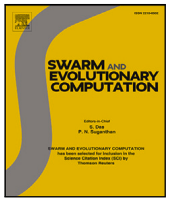
Swarm and Evolutionary Computation 84 (2024) 101445

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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



A supervised parallel optimisation framework for metaheuristic algorithms

Eugenio J. Muttio [a,*], Wulf G. Dettmer [a], Jac Clarke [a], Djordje Perić [a], Zhaoxin Ren [a], Lloyd Fletcher [b]

[a] Faculty of Science and Engineering, Swansea University, Bay Campus, Fabian Way, Swansea, SA18EN, Wales, United Kingdom
[b] UKAEA, Culham Science Centre, Abingdon, Oxfordshire, OX14 3DB, United Kingdom

ARTICLE INFO

Keywords:
Optimisation
Parallel computation
Metaheuristics
Population-based algorithms

ABSTRACT

A Supervised Parallel Optimisation (SPO) is presented. The proposed framework couples different optimisation algorithms to solve single-objective optimisation problems. The supervision balances the exploration and exploitation capabilities of the distinct optimisers included, providing a general framework to solve problems with diverse characteristics. In this work, five optimisation algorithms are included in the ensemble: Particle Swarm Optimisation (PSO), Genetic Algorithm (GA), Covariance Matrix Adaption-Evolution Strategy (CMA-ES), Differential Evolution (DE), and Modified Cuckoo Search (MCS). A geometric path-finding problem with numerous local minima is used to demonstrate the advantage of SPO. The effectiveness of the approach is compared with that of stand-alone incidences of the integrated optimisation strategies and with state-of-the-art algorithms. In addition, a benchmark test suit composed of engineering applications is utilised to validate the general applicability of SPO with respect to a variety of problems. The good solutions generated by SPO are shown to be generally reproducible, while isolated algorithms, at best, render good solutions only occasionally.

1. Introduction

Optimisation is a field in continuous development due to the wide range of applications found in science, engineering, economics, communication, and other fields. The optimisation field has received a renewed impetus in the last three decades or so due to the advances in machine learning, where the *training* stage typically involves the search for an optimal solution. Consequently, the optimisation field is continuously evolving in order to address the requirements of emerging applications and novel software technologies.

A traditional optimisation approach takes into account the gradient of the objective function to determine a possible direction of the solution. However, real-world problems are generally discontinuous, non-differentiable, discrete, noisy, multimodal, and possibly dynamic. To address these challenges, a range of gradient-free strategies referred to as *meta-heuristics* emerged in the second half of the 20th century and exponentially increased in the last few decades due to their success in applications to a wide range of complex real-world problems.

In general, a meta-heuristic algorithm is characterised by initialising a random population of agents that develop through generations to find an improved solution. The selection process is based on each agent's fitness (function evaluation), and, may contain operations such as crossover between agents, mutation, random walks, etc. Some of the best-known meta-heuristic algorithms include genetic algorithms (GA) [1], simulated annealing (SA) [2], particle swarm optimisation (PSO) [3], CMA evolution strategy (CMA-ES) [4], differential evolution (DE) [5–8], cuckoo search (CS) [9], bat algorithm (BA) [10], firefly algorithm (FA) [11,12], and more recently, Electromagnetic field optimisation (EFO) [13], whale optimisation algorithm (WOA) [14,15], and squirrel search algorithm (SSA) [16]. It should be noted, that variations of these strategies and novel metaheuristic algorithms are being continuously developed targeting both generic optimisation problems and specific applications. Challenges to be addressed include the problem-dependent suitability and performance of meta-heuristics, premature convergence [17–19], local sub-optimal solutions and poor reproducibility.

We argue that a combination of algorithms with different performance capabilities is advantageous when dealing with problems that involve a complex solution space. The desired behaviour includes sufficient exploration, which permits the identification of potential regions, and an exploitation capability that intensifies the local search. Strategies involving operations such as mutation, crossover and random walks are known to preserve exploration, whereas algorithms that are based on the kinematics of a swarm population are excellent for solution refinement. Hybridisation strategies merge the algorithmic

* Corresponding author.
E-mail addresses: e.j.muttiozavala@swansea.ac.uk (E.J. Muttio), w.g.dettmer@swansea.ac.uk (W.G. Dettmer), jac.clarke@swansea.ac.uk (J. Clarke), d.peric@swansea.ac.uk (D. Perić), zhaoxin.ren@swansea.ac.uk (Z. Ren), lloyd.fletcher@ukaea.uk (L. Fletcher).

https://doi.org/10.1016/j.swevo.2023.101445
Received 20 June 2023; Received in revised form 19 November 2023; Accepted 29 November 2023
Available online 6 December 2023

procedure of two or more established optimisers to achieve a more versatile functionality. Common hybridisation optimisers include genetic algorithms (GA) with particle swarm optimisation (PSO) [20,21], a simulated annealing and PSO hybrid approach [22,23], cuckoo search (CS) inspired by PSO [24–26], a CS-PSO hybrid with DE for global search [27], a DE and PSO combination [28–30], a PSO including Lévy flights [31], a clustering algorithm based on gravitational search algorithms (GSA) and k-means (GSA-KM) [32], a Memetic Algorithm (MA) composed by an evolutionary framework and a set of local search algorithms [33], and many more.

An alternative strategy is to combine the special features of different algorithms by running the algorithms independently and performing merging or seeding processes of their populations within the overall optimisation strategy; such strategies are commonly referred to as *Ensemble strategies*. Wu et al. [34] present a comprehensive review of these promising strategies, and provide their categorisation according to the decision of the assembling process. Ensemble approaches have been proposed utilising multiple instances of a single class of algorithm, e.g. an ensemble of DE variants [35–39], PSO variants [40,41], including a behaviour pool [41,42], or a combination of different type of EAs [43–51]. It should be noted that due to the high computational effort required for the solution of practical real-world problems, parallel optimisation is considered essential. Numerous studies on communication in a parallel setting for optimisation are found in literature, including, among others, the efficiency between processors [52], the correlation of variables in objective functions [53], and parallel architectures [54,55].

The objective of this work is the development of a novel generalised strategy for practical real-world optimisation problems. The strategy is capable of coupling multiple independent optimisation algorithms executed within a Supervised Parallel Optimisation (SPO) framework. In order to demonstrate the main features of the proposed SPO strategy a challenging optimisation problem is first considered consisting of finding the shortest path between two points subject to a large number of non-penetrable randomly distributed obstacles. In addition, diverse engineering applications have been solved to demonstrate the general applicability of the supervised framework. It should be noted that the proposed ensemble strategy is designed to extract the best features of separate optimisation algorithms in order to enhance the optimisation strategy for complex real-world problems.

This article is organised as follows: The optimisation algorithms included in this ensemble approach are described in Section 2, which include PSO, GA, CMA-ES, DE and MCS. The proposed supervised parallel optimisation strategy is introduced in Section 3, where the general algorithmic structure and the two crucial mechanisms of SPO are fully described. In Section 4 the path-finding optimisation problem is defined, and the performance of the proposed methodology is tested by performing an analysis of convergence behaviour, hyperparameter analysis and evaluation of algorithmic complexity. In addition, a comparative analysis is carried out by contrasting solutions obtained by stand-alone algorithms included in the ensemble and state-of-the-art algorithms. The solutions obtained for the engineering application problems are summarised in Section 5. Finally, conclusions are presented in Section 6.

2. Meta-heuristic algorithms

Although the number of meta-heuristic algorithms that can be included in the framework is not limited, it is encouraged to select optimisers with different characteristics. In this work, five algorithms with diverse capabilities are selected: A genetic algorithm (GA) that provides operations such as mutation and crossover among solutions, a particle swarm optimisation (PSO) that is capable of performing a global search while refining locally, a covariance matrix adaptation evolution strategy (CMA-ES) that converge fast when exploiting a solution due to the approximation to a quasi-Newton method, a differential evolution (DE) that is formulated based on a crossover mechanism, and a modified cuckoo search (MCS) that uses random walks to explore large areas while elitist mutation operations are helpful to intensify the best candidates. The first four aforementioned optimisers are implemented utilising a Python multi-objective optimisation library (Pymoo) [56], whereas a Python version of the modified cuckoo search (MCS) is adapted from Walton et al. [57].

2.1. Particle Swarm Optimisation (PSO)

Particle Swarm Optimisation (PSO) [3] is considered a reference among the so-called *swarm intelligence* methods due to its simplicity and speed. Each member of the population, or "particle", has a position that represents a potential solution. This position has an associated fitness, or "cost", defined by the objective function. Each particle's position is updated iteratively until it reaches a termination criterion. The swarm converges towards the best region under a simple set of influences, including the local memory of its best position, the swarm's knowledge of the global best position and the particle inertia. The velocities v_d of the particles are updated by

$$^{(t+1)}v_d^i = \omega\ ^{(t)}v_d^i + c_1 r_1 (p_d^i - {}^{(t)}x_d^i) + c_2 r_2 (g_d - {}^{(t)}x_d^i) \tag{1}$$

where the left superscripts (t) and $(t+1)$ refer to the current and new iterations respectively, the subscript d refers to the design variable (or coordinate) of each particle i, p_d^i is the particle's local best position, g_d is the swarm global best position, $^{(t)}x_d^i$ is the particle's current position, r_1 and r_2 are both random scalar coefficients, ω is the inertia coefficient, c_1 is the local best coefficient and c_2 is the global best coefficient. The position x_d^i of each particle is updated by

$$^{(t+1)}x_d^i = {}^{(t)}x_d^i + {}^{(t+1)}v_d^i \tag{2}$$

where $^{(t+1)}x_d^i$ corresponds to the updated position of particle i at iteration $(t+1)$, d is the design variable, and v_d^i is the updated velocity from Eq. (1).

2.2. Genetic Algorithms (GAs)

Genetic Algorithms (GAs) were introduced in the 1960s by Professor John Holland and his collaborators at the University of Michigan [1]. The essential characteristics of GAs include the representation of individuals as chromosomes, manipulation of these by genetic operators, and selection of the best candidates with the aim of converging towards an optimal solution. The three main genetic operators include a *crossover* process swapping elements of two chromosomes aiming to converge in a subspace; a *mutation* operation changes parts of one individual randomly, which increases the diversity; and a *selection* that allows propagating the best solutions on to next generations. Numerous GA variants have been presented since its introduction, focused especially on the improvement of the genetic operators. A simulated binary crossover (SBX) operator has shown good performance for real and binary coded GAs, in which two children c_1 and c_2 are created as follows

$$c_1 = 0.5\left[(p_1 + p_2) - \beta_1' \left|p_2 - p_1\right|\right] \tag{3}$$

$$c_2 = 0.5\left[(p_1 + p_2) + \beta_2' \left|p_2 - p_1\right|\right] \tag{4}$$

where p_1 and p_2 are any two parent solutions, $\beta_{1,2}'$ is found by using a cumulative polynomial probability distribution based on two non-dimensionalised factors $\beta^{L,U}$ taking into account the spread ratio between children and parents [58]. Among many mutation operators, a polynomial mutation [59] is suggested due to its efficient scheme and diversity preservation, stating that for a parent solution p

$$p' = \begin{cases} p + \bar{\delta}_L \left(p - x_d^L\right) & \text{for } u \le 0.5 \\ p + \bar{\delta}_R \left(x_d^U - p\right) & \text{for } u > 0.5 \end{cases} \tag{5}$$

where p' is the mutated solution, $u \in [0,1]$ is a random number, x_d^L and x_d^U correspond to the lower and upper bounds of each design variable d, and the two $\bar{\delta}_{L,R}$ parameters are calculated as follows

$$\bar{\delta}_L = (2u)^{1/(1+\eta_m)} - 1, \quad \text{for } u \leq 0.5 \tag{6}$$

$$\bar{\delta}_R = 1 - (2(1-u))^{1/(1-\eta_m)}, \quad \text{for } u > 0.5 \tag{7}$$

where η_m is a user-defined index parameter. A desired behaviour presented in GAs is that, as the process evolves, multiple offspring can explore diverse regions of the search space alleviating premature convergence problems.

2.3. CMA-ES algorithm

Evolution strategies (ES) are algorithms based on the use of mutation and selection mechanisms. CMA-ES is a second-order approach to estimating a positive definite matrix within an iterative procedure, proving very useful when applied to ill-conditioned objective functions [4,60]. This leads to a similar approximation of the inverse Hessian matrix in the classical quasi-Newton optimisation method. The population is generated by sampling a multivariate normal distribution

$$\mathbf{x}^i \sim \mathcal{N}\left(\mathbf{m}, (\sigma)^2\, \mathbf{C}\right) \tag{8}$$

where $\mathbf{x}^i$ is the (ith) offspring (search point), $\sim$ denotes the same distribution in both sides of the equation, $\mathbf{m}$ is the mean value of the search distribution, σ is the overall standard deviation (step size) at current generation, $\mathbf{C}$ is the covariance matrix, and $\mathcal{N}$ is the multivariate normal search distribution defined by

$$\mathcal{N}(\mathbf{m}, \mathbf{C}) \sim \mathbf{m} + \mathcal{N}(0, \mathbf{C}) \sim \mathbf{m} + \mathbf{mBD}\mathcal{N}(0, \mathbf{I}) \tag{9}$$

where the covariance matrix is decomposed as $\mathbf{C} = \mathbf{B}(\mathbf{D})^2\mathbf{B}^T$, $\mathbf{B}$ are the eigenvectors of $\mathbf{C}$, the squared diagonal elements of the diagonal $\mathbf{D}$ are the corresponding eigenvalues, $\mathcal{N}(0, \mathbf{I})$ are independent realisations of the multivariate normal distribution with zero mean and covariance equal to the identity matrix $\mathbf{I}$.

CMA-ES has minimal user control avoiding tedious parameter tuning for a specific problem. The algorithm has been empirically successful and outperformed other methods on low-dimensional functions and functions that can be solved with a small number of function evaluations. However, as indicated in [61], CMA-ES has disadvantages such as premature stagnation when solving large-scale optimisation problems.

2.4. Modified Cuckoo Search (MCS)

The cuckoo search (CS) algorithm [9] is inspired by the brood parasitism of certain cuckoo bird species and by the foraging exhibited by many animals. The description of CS can be simplified into the following set of rules: Each cuckoo lays a single egg at a time and leaves it in a random nest; the nests containing the eggs with the best fitness values are protected and carried on to the next generation. Lastly, as the number of available nests is a fixed value, a probability $P_a \in (0,1)$ is introduced to allow for the removal of an egg if it is discovered. This algorithm combines local and global random walks, where the latter is carried out by Lévy flights i.e.

$${}^{(t+1)}\mathbf{x}^i = {}^{(t)}\mathbf{x}^i + \alpha \oplus \text{Lévy}(\lambda) \tag{10}$$

where ${}^{(t+1)}\mathbf{x}^i$ corresponds to the updated nest position vector, $\alpha > 0$ controls the step size of a flight and should be related to the scales of the problem and the product $\oplus$ means entrywise multiplications. A Lévy flight is essentially a random walk that is drawn from a Lévy distribution, providing a more efficient method to explore the design space.

A CS variant denominated modified cuckoo search (MCS) includes a decreasing α coefficient, which enhances exploitation as the agents evolve towards a potentially better solution and a crossover mechanism between the current solutions.

2.5. Differential Evolution (DE)

Differential Evolution (DE) [5,6] is a search method that makes use of N_P D-dimensional parameter vectors such as population for each generation. DE performs the following operations: A mutation, by adding a weighted difference between two vectors, a crossover, that mixes the mutated vector's parameters with a predetermined "target" vector, and a selection of the best solution with respect to the new and target vector. For each target vector $\mathbf{x}^i$, $i = 1, 2, 3, \ldots, N_P$, mutation is performed according to

$${}^{(t+1)}v_d^i = {}^{(t)}x_d^{r1} + F \cdot ({}^{(t)}x_d^{r2} - {}^{(t)}x_d^{r3}) \tag{11}$$

where ${}^{(t+1)}v^i$ is a mutant vector at iteration $(t+1)$ with random indices $r1, r2, r3 \in 1, 2, \ldots, N_P$, that represent different non-repeated integers, the subscript d refers to the design variable, and a real constant factor $F > 0 \in [0, 2]$ controls the amplification of the variation.

3. Supervised parallel framework

Ensemble strategies are promising approaches that combine the behaviour of established metaheuristics. The proposed approach includes a novel supervised framework with straightforward mechanisms that initialise optimisation instances, monitor them and decide which remains active. A parallelisation of the methodology is recommended as one thread is assigned to supervise and the rest are optimisation instances (workers). Based on the classification described by Wu et al. [34], the strategy proposed in this work is considered as a *high-level ensemble* in which the intrinsic formulation of the algorithms is not altered. It is implemented following a *cooperative multi-population* approach built by the supervisor when receiving messages from the workers that have assigned optimisation instances with their own developed population.

3.1. Parallel supervisor-worker structure

On a multi-processor machine, one of the processors adopts the role of the *supervisor*, while the remaining processors take on the role of the *workers*. The supervisor is in charge of initialising each worker with an optimisation algorithm predefined by the user, which, in this work, can be a combination of PSO, GA, CMA-ES, DE or MCS. Each worker starts an isolated optimisation algorithm, i.e. runs a stand-alone optimiser in one processor. At the beginning of the working process, the population is initialised by a random uniform distribution. Whenever each worker completes a defined number of generations N_{gen}, it reports its current best solution to the supervisor. This process is asynchronous as each optimiser has a different performance speed. When the supervisor receives a message from each worker, it starts filling a repository of size N_{rep} with the best solutions reported so far. In that sense, the supervisor is continuously monitoring and sorting new incoming messages. While the implementation in this work is based on Python and the parallel communication is implemented using Message Passing Interface for Python (MPI4Py), the algorithmic structure described in Algorithm 1 is easily extended to any programming language.

There are two crucial features of this approach, both performed by the supervisor. The first one is the *stopping* of a worker that is triggered when the supervisor does not observe sufficient improvement in the relatively poor solutions reported by the same worker. If a stalled worker is detected, the supervisor stops the current optimisation process and reinitialises the optimisation process on the corresponding worker. Then, depending on a given probability, the *seeding procedure* is activated, in which the new algorithm can initialise its population with one or more of the best solutions collected in the supervisor's repository. This is an important feature because certain algorithms that could not perform adequately in the first stage of the optimisation process, commonly denominated as the exploration phase, can thus benefit from previous solutions obtained by other types of workers and focus on that region. Three fundamental steps of the process: (a) initialisation, (b) reporting/stopping and (c) seeding, are schematically displayed in Fig. 1 and are further explained in the following sections.

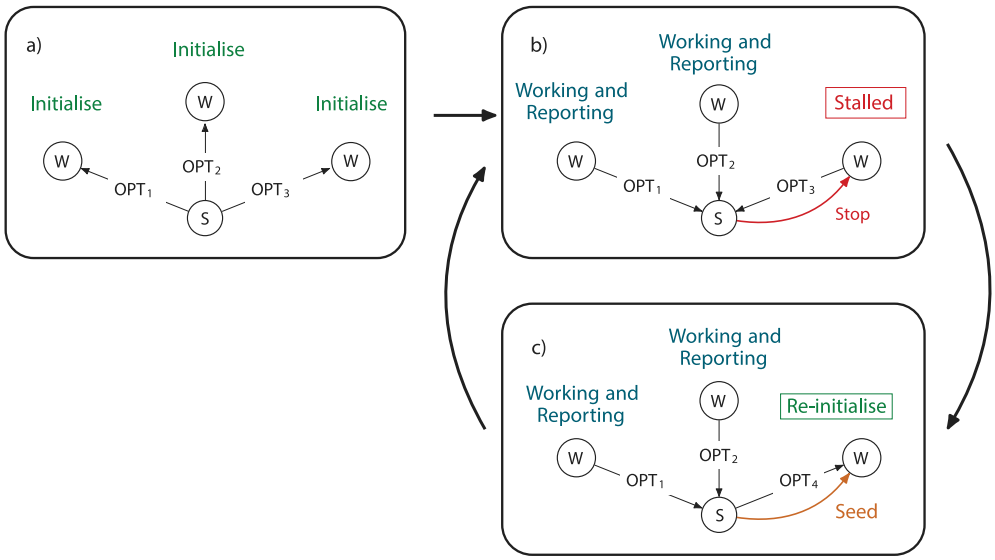


Fig. 1. Supervised parallel structure and roles of the processors in the proposed strategy. Three stages are depicted: (a) processor initialisation by the supervisor (S), (b) workers (W) report their performance to the supervisor (S) and the supervisor stops stalled workers, and, (c) the supervisor re-initialises the inactive worker with a new optimiser including a seed from its repository.

Algorithm 1 Supervisor-Worker Structure.

```
if Supervisor then
    while workersAlive > 0 do                          ▷ Monitor while workers exist
        status ← MPI.status                            ▷ Verify MPI status
        if status.tag is 1000 then                     ▷ Worker is idle
            pop ← SeedingProcedure()                   ▷ Initialise population
            MPI.send(optimiser(pop))                   ▷ Initialise optimiser and assign hyperparameters
        else if status.tag is 2000 then                ▷ Receive data from worker
            workerData ← MPI.recv(source=anyWorker)    ▷ Worker data include objective function evaluation (cost), design parameters and MPI rank.
            if workerSol is better than supervisorSol then
                supervisorSol ← workerSol              ▷ Update best solution obtained and store it in seeds' repository
                saveRepository(supervisorSol)
            stallWorker ← stoppingCriteria(workerData) ▷ Verify if the worker is stalled
            if stallWorker is True and Worker is not Top then
                stallWorker ← True (Stop)              ▷ Verify if worker is one of the top
            MPI.send(stallWorker)                      ▷ Send supervisor's decision (continue/stop) to optimiser
        else if status.tag is 3000 then                ▷ Tell supervisor worker has finished
            workersAlive -= 1
else if Worker then
    workerReady ← True                                 ▷ Worker ready is set to True
    continueFlag ← True                                ▷ Continue work is set to True
    while continueFlag is True do
        MPI.send(workerReady, tag=1000)                ▷ Tell supervisor worker is available
        MPI.recv(optimiser)                            ▷ Receive optimiser algorithm from supervisor
        optimiser.run()                                ▷ Run assigned optimiser, which will send its solution data (tag 2000) to the supervisor at every checkpoint
    workerReady ← False
    MPI.send(workerReady, tag=3000)                    ▷ Tell supervisor worker has finished
```

3.2. Stopping criteria

The workers report regularly their best cost and solution to the supervisor at each checkpoint, or update frequency, at every N_{gen} generations. The parallel communication and repository update is set to the same checkpoint for all optimisers. However, due to the different optimisers' speeds, worker-supervisor communication could occur more often to some optimisers than others. The supervisor monitors the current solution sent by each worker and keeps the history of the previous solutions. Then, the supervisor can assess if the worker is not improving sufficiently and can classify the optimisation process as *stalled*. When this occurs, the supervisor stops the worker if it is not one

of the $N_{topset} < N_{workers}$ workers, and a new optimisation algorithm is started. The overall process stops when N_{runs} optimisation procedures have been completed or a time limit has been imposed. The criterion used by the supervisor to detect stagnation can be written as

$$\frac{\epsilon_m}{\epsilon_{m-N_{\text{stall}}}} > 1 - \textit{stall tolerance} \quad \Rightarrow \quad \text{Optimisation has stalled.} \tag{12}$$

where ϵ_m is the mth cost reported to the supervisor by the corresponding worker and a *stall tolerance* is a parameter defined by the user. The critical number of checkpoints reached without sufficient improvement N_{stall} is calculated from

$$N_{\text{stall}} = \bar{N}_{\text{stall}} \left(\frac{\bar{\epsilon}}{\epsilon_m} \right)^p \tag{13}$$

where $\bar{N}_{\text{stall}}$ is an initial number of stalled solutions allowed. The exponent p may be chosen as 1, 2, 3, or higher if required, and controls how much longer the workers are allowed to explore solutions of more advanced quality. The reference cost $\bar{\epsilon}$ is computed automatically by the performance of the initial workers. At the start of the proposed optimisation framework, the first workers are considered *explorers* as the initial population is randomly generated, and, it is likely that some of them are stalled at $N_{\text{stall}} = \bar{N}_{\text{stall}}$. When this happens for the $N_{\bar{\epsilon}}$ time in every optimisation algorithm, the reference cost $\bar{\epsilon}$ is set to the average of the cost $\epsilon_{N_{\bar{\epsilon}}}$ among the type of optimisers included.

$$\bar{\epsilon} = \frac{1}{N_{alg}} \sum_{i=1}^{N_{alg}} \epsilon_m^i \tag{14}$$

where N_{alg} is the number of different types of optimisation algorithms run by the workers, e.g. this work considers PSO, GA, DE, CMA-ES and MCS.

To better exemplify this process, consider the case of using just one optimisation algorithm and defining $N_{\bar{\epsilon}} = 1$, then, the reference cost $\bar{\epsilon}$ is computed when the first worker is stalled. If using more than one optimisation algorithm, the cost of the stalled workers is stored until reaching $N_{\bar{\epsilon}}$ to compute the optimiser's average reference. This is particularly important when considering more than one algorithm, as their performance can be significantly dissimilar in the exploration phase. When the reference cost $\bar{\epsilon}$ is established, the number of stalled solutions allowed will increase as stated by Eq. (13). Algorithm 2 describes the steps to determine if a worker is declared stalled.

3.3. Seeding procedure

During the optimisation procedure, the workers are constantly sending messages to the supervisor with the current best location found. The supervisor receives these messages, arranges them according to the cost, and stores them in a *seed* repository of size N_{rep}, taking precautions to avoid duplicates of the gathered solutions. The seeding procedure can happen only after the first worker has been declared stalled. In that instant, the supervisor should re-initialise a new optimiser to avoid having an inactive worker. The optimisation algorithm may be the same as before or not, but the population is different, as it may be initialised randomly or with a solution (seed) from a previous worker. This is advantageous in the following scenario; consider an algorithm A that is an excellent explorer in a given problem, but it is unable to refine its solution, hence, it cannot improve for a certain duration and the supervisor decides to stop it. Then, consider an algorithm B that is an excellent exploiter but is inefficient during exploration. The proposed strategy couples both algorithms by running an exploiter algorithm B that has been seeded by an explorer algorithm A, maximising the capabilities of both.

The process has been implemented in a way that not all workers are initialised with seeds, thus allowing for the preservation of diversity in the general population and avoiding over-exploiting the same region of the solution space. The probability $\nu \in [0, 1]$ for seeding as opposed to randomly initialising the new population is set by the user. Experiments done by the authors suggest that values $\nu > 0.9$ are disadvantageous as they over-emphasise exploitation. The number of seeds introduced into the population of a worker is given by a random uniform (RU) distribution and controlled by another parameter, denoted by a percentage of the algorithm population $\phi \in [0, 1]$. This means that not all the workers may have the same amount of seeds, which again, helps to preserve diversity. The general seeding procedure is summarised in Algorithm 3.

4. Illustrative example: Geometric path-finding problem

A geometric path-finding problem has been designed to test the performance of the proposed strategy. The objective is to minimise the length of a path described by the coordinates of a sequence of points, subject to avoiding penetration of several randomly placed circular obstacles. Note that this problem differs from discrete graph-based path-finding problems, commonly applied in transport networks such as roads, railways, and others, in which the node locations are fixed. Stochastic On-Time Arrival (SOTA) and derived methods [62–65] have been proposed to find the shortest travel time in such discrete graph systems. Conversely, the proposed geometric path-finding problem as described below does not feature a discrete graph structure.

Considering the large number of obstacles shown in Fig. 2, the model problem described here features numerous local minima and allows for experimentation with large numbers of design variables. Hence, it is expected that stand-alone evolutionary optimisation strategies are likely to suffer from premature convergence issues. It can be argued that the optimisation process has to address two tasks of very different characteristics, firstly the identification of the correct gaps between the obstacles and secondly the straightening of the several sections of the path. The problem is sufficiently complex to represent challenging applications and to test the supervised parallel optimisation strategy proposed in Section 3.

4.1. Problem definition

A rectangular domain with $x \in [0, 30]$ and $y \in [-15, 15]$, contains $N_c = 48$ randomly positioned circular obstacles of varying radii as shown in Fig. 2. The objective of the optimisation problem is to compute the shortest path from Point A with $(x, y) = (0, 0)$ to Point B with $(x, y) = (30, 0)$, such the path does not intersect any of the circular obstacles. The path is defined by a sequence of N_p points that are connected by straight line segments. The points are equally spaced in the x-direction. Hence, the set of design variables reduces to an N_p-dimensional array $y = y_1, y_2, \ldots, y_{N_p}$ that contains the y-coordinates of the points. A penalty formulation is used to avoid the intersection of the path with any of the circles. Hence, denoting the path length and the obstacle penetration by, respectively, $l(y)$ and $p(y)$, the cost function can be written as

$$cost = l(y) + k\ p(y) \tag{15}$$

where the penalty factor is set to $k = 1$. The length of the path is computed from

$$l(y) = \sum_{i=1}^{N_p - 1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{16}$$

while the penetration can be evaluated from

$$p(y) = \sum_{i=1}^{N_p - 1} \sum_{j=1}^{N_c} \max \left(0, R_j - \sqrt{(X_j - x_i)^2 + (Y_j - y_i)^2} \right) \tag{17}$$

where, R_j, X_j and Y_j represent, respectively, the radii and the coordinates of the centre points of the circular obstacles. The penetration is illustrated in Fig. 3. Recall that the coordinates x_i are known from the equal spacing of the points in the x-direction.

Algorithm 2 Stopping Criteria.

$\epsilon_m \leftarrow$ Worker cost ▷ The worker sends the cost of its best solution

ϵ_{his}.append(ϵ_m) ▷ Store cost history per worker

while $\left(\frac{\epsilon_m}{\epsilon_{m-N_{\text{stall}}}} < 1 - tolerance\right)$ **do** ▷ Verification of stalled worker by Eq. (12)

    remove(ϵ_{his}.first) ▷ Remove the first cost received

if $\bar{\epsilon}$ **is set then**

    $N_{\text{stall}} \leftarrow \bar{N}_{\text{stall}} \left(\frac{\bar{\epsilon}}{\epsilon_m}\right)^p$ ▷ Compute a new number of stalled messages allowed.

if Size(ϵ_{his}) > N_{stall} **then** ▷ Verify if a worker is stalled

    StallWorker ← **True** ▷ Worker is declared stalled

    Optim.StallCounter += 1 ▷ Stall counter per each optimisation algorithm

    if (All) Optim.StallCounter > $N_{\bar{\epsilon}}$ **then**

        OptimFlag ← **True** ▷ Check if every optimiser has at least $N_{\bar{\epsilon}}$ stalled runs

    if $\bar{\epsilon}$ **not set** and OptimFlag is **True then**

        $\bar{\epsilon} \leftarrow \frac{1}{N_{alg}} \sum_{i=1}^{N_{alg}} \epsilon_m^i$ ▷ Reference by averaging the stalled $N_{\bar{\epsilon}}$ cost of all optimisers

Algorithm 3 Seeding Procedure.

Pop ← RU(PopSize) ▷ Initialise population using a random uniform distribution RU

if RepExists **then**

    if RandNum < ν **then** ▷ Verify probability ν of seeding a population

        MaxSeeds = $\phi \times$ Pop ▷ Maximum number of seeds constrained by percentage ϕ

        SeedsFromRep ← random(0, MaxSeeds) ▷ Number of seeds is a random number

        for pi ← 1 to size(SeedsFromRep) **do**:

            RandSeed ← random(0, RepSize)

            RandPop ← random(0, PopSize)

            Pop[RandPop] ← Repository[RandSeed] ▷ A random particle from the population is replaced by a random seed from the repository

4.2. Results and discussion

The proposed methodology has been tested for the path-finding problem described in Section 4.1. The number of points defining the path, i.e. the number of design variables chosen is 200. The optimisation algorithms included in the supervised approach are PSO, GA, CMA-ES, DE and MCS, as introduced in Section 2. The recommended parameters, detailed in Appendix C, have been used to set up each optimiser, i.e. without parameter experimentation done a priori. In addition, *explorer* (V1) and *exploiter* (V2) versions of PSO and MCS are included by adjusting the parameters to continuously maintain diversity in the population and to perform intensification, respectively. The experiment is carried out in a parallel system using 16 processors, hence, one CPU is reserved for the supervisor and $N_{workers} = 15$, and a time limit has been imposed to 15 h of computation.

4.2.1. Convergence behaviour

The convergence behaviour of the proposed methodology has been presented in Fig. 4 in which the convergence plot of every optimisation instance is superimposed and shown in different colours. It can be noticed that a vast number of workers with high costs are clustered in the initial exploration phase. The workers are allowed to continue if they are capable of decreasing their costs sufficiently, or, on the contrary, they are stopped. After the reference cost $\bar{\epsilon}$ is defined, the workers remain active and intensify the local search. This results in a characteristic *tree* shape in Fig. 4(a). Every new worker can be initialised by a previous solution, or *seed*, which is indicated on the plot by a black point in the centre of each marker. The probability of seeding a worker is chosen as $\nu = 0.5$, while the maximum proportion of the seeded population is $\phi = 1.0$, i.e. some workers could start having their entire population seeded. As expected, it is less likely that one algorithm remains the best in the entire process, but the best solution can be found by different algorithms through each phase, hence, a triangle marker is used to identify when an optimiser has been the best at some point.

Fig. 4(b) shows the convergence behaviour over time exhibiting that optimisers with exploitation capabilities take over and start refining the solution after approximately three hours of exploration. To maintain diversification, new explorers are continuously initialised in the

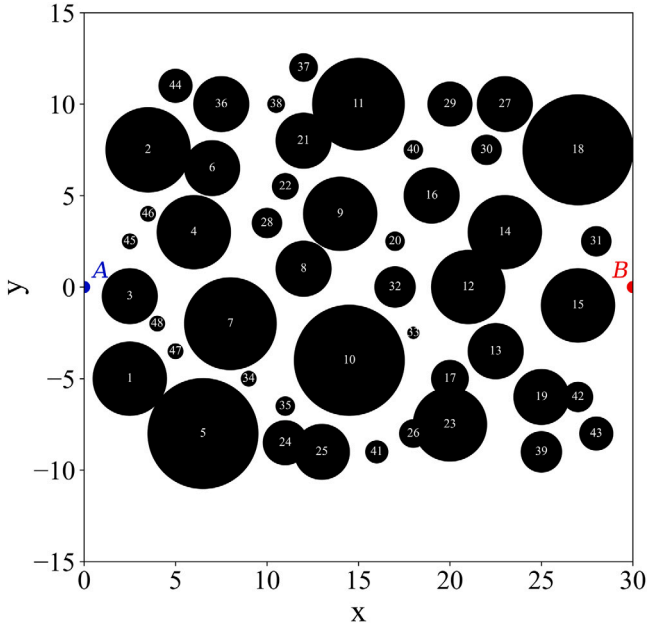


Fig. 2. Path-finding problem domain and obstacles imposed.

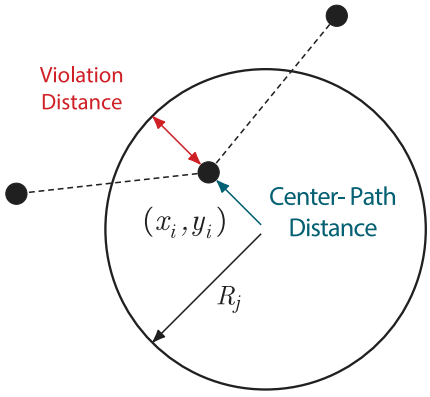


Fig. 3. Definition of the obstacle penetration.

remaining time, keeping the optimisation instances that show sufficient progress and stopping optimisers with poor improvement. Fig. 4(c) is a magnification of Fig. 4(a) in the refinement region, in which a GA optimiser acts as a link between exploration and exploitation phases by seeding other optimisers; this is identified with triangle markers in between costs 40–50 of Fig. 4(c). This is also shown in Fig. 4(d), which illustrates the seeding process over time in the refinement region, as different optimisers take over the best solution. The exploiter MCSV2 is the last optimiser refining a solution, indicating that it is a suitable algorithm for exploitation, however, notice that the combination of different optimisation instances such as DE, PSO and PSOV1 are needed to accomplish the final cost.

An additional analysis that helps to understand the behaviour of the proposed strategy is to examine the performance of each type of optimiser included in the ensemble. This can be done indirectly by counting the messages received by the supervisor during the optimisation process. Three possible reasons suggest why a type of optimiser sends more messages than others:

1. The supervisor allows to run an optimiser for longer due to its good performance.
2. More instances of a suitable optimiser are active taking the place of other optimisers.
3. The speed of an optimiser is considerably faster than the rest of the optimisers.

The first two cases are straightforward possible scenarios, in which both can occur individually or simultaneously. The third case has the following two possible causes:

(a) The optimiser has good performance, hence, the three aforementioned cases may occur at the same time.
(b) The optimiser has poor performance, but, accomplishes sending at least $\bar{N}_{stall}$ solutions at every initialisation, having an impact on the number of messages due to its speed of computation.

On the other hand, if an optimiser sends a low number of messages, the straightforward reason is that the supervisor is not allowing it to run for longer (poor performance). However, an optimiser may be much slower than the rest of them, but, its solutions are excellent. The number of received messages is highly related to the level of exploration (more initialisations) and exploitation (optimisers run for longer). Hence, it is useful to analyse the messages sent by an optimiser and their level of accuracy to understand the decisions taken by the supervisor. Fig. 5 displays the message analysis, where the grey bar indicates the optimiser message percentage m_o

$$m_o = \frac{Nm_o}{Nm_s} \cdot 100\% \tag{18}$$

where Nm_o is the total number of messages per optimiser, and Nm_s is the total number of messages that the supervisor received. The coloured bar indicates the corresponding percentage when the message was the best solution.

Fig. 5(a) corresponds to the message analysis of the best solution achieved from ten experiments considered, Fig. 5(b) shows the mean and standard deviation, and Fig. 5(c) is the ratio between the best message received with respect to the total number of messages per optimiser (coloured bar from 5(b)), the value of the ratio is the mean value of the ten experiments considered. It can be observed that the three versions of MCS have similar m_o and percentage of being the best optimiser, whereas the exploiter PSOV2 has a major impact during the optimisation process, most noticeable in the refinement section, compared to the other two PSO versions considered. CMA-ES does not perform on the same level as the others for this problem as the number of messages when it performs best is negligible. The GA algorithm has the second place in terms of the total number of messages m_o, but it has the fifth place of best messages (Fig. 5(c)). In general, DE is the algorithm that has larger m_o and has the greater percentage of being the best, which suggests that it is a suitable algorithm within the ensemble for this problem, followed by GA, PSO, MCS and the last is CMA-ES. Note that this analysis helps to understand the behaviour of an algorithm within the ensemble. However, despite identifying DE as an optimiser with more incidence of best messages this does not guarantee that running a stand-alone DE algorithm will be sufficient for solving the problem. A comparison of the proposed ensemble strategy and each isolated algorithm is discussed in the following section.

4.2.2. Comparison with isolated optimisers

A comparison exercise has been carried out by considering the same optimisers included in the proposed approach, however, functioning as stand-alone procedures. The explorer and exploiter versions of PSO and MCS are not included in this comparison as their performance is very poor and does not make sense to run an isolated optimisation procedure. To perform a fair comparison, the same computational effort has been taken into account for the stand-alone optimisers by running as many independent optimisers as workers used in the proposed approach, i.e. as $N_{workers} = 15$, or 15 CPUs, in the supervised approach, then, 15 independent runs are carried out for each optimiser. This test is performed 10 times with the proposed approach, which means that each independent optimiser is run 150 times. The convergence of the

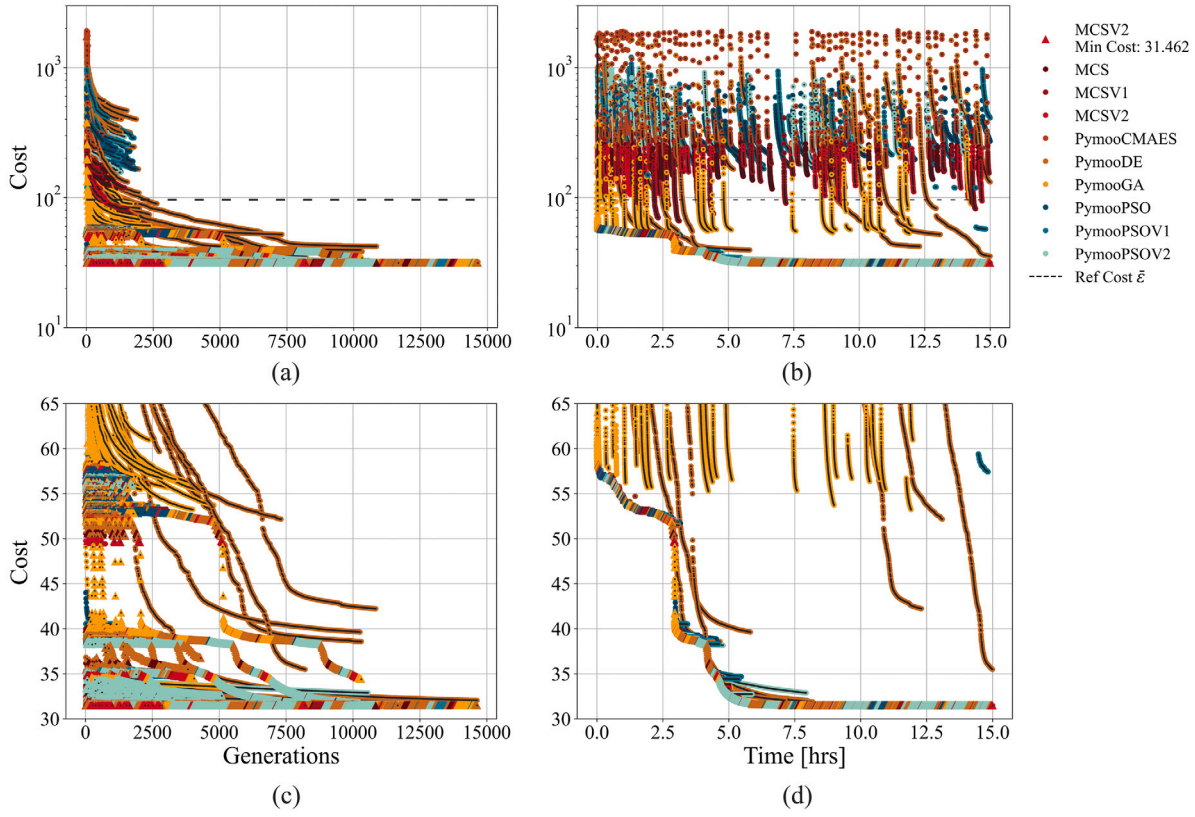
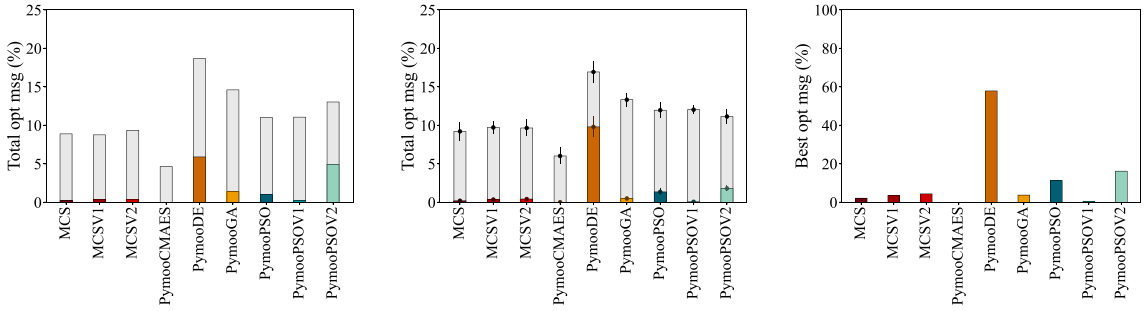


Fig. 4. Convergence behaviour of the supervised parallel ensemble strategy: (a) superimposed convergence-generations plot of every optimisation instance, (b) superimposed convergence plot over time of every optimisation instance, (c) refinement region of the convergence plot (a), and (d) refinement region of the convergence plot over time (b).



(a) Best solution. (b) Mean and standard deviation. (c) Best message ratio.

Fig. 5. Number of messages in percentage sent by each optimiser (grey) and percentage of these messages when the optimiser was the best solution (colour by each optimiser). (a) Messages percentage in the best solution achieved, (b) messages percentage mean and standard deviation of ten experiments considered, and (c) ratio of the best message sent with respect to the total messages per optimiser, the result shown is the mean across ten experiments.

best solution achieved, the mean and standard deviation are presented in Fig. 6, in which the vertical axis is the objective function while the horizontal is the computation time, with a maximum of 15 h utilising 15 CPUs. It is shown that SPO consistently finds the best solution with a higher level of accuracy. Table 1 presents the best solution achieved by each optimiser, the mean, worst, standard deviation and median of the 10 experiments carried out by the supervised approach, and the 150 runs by the stand-alone optimisers.

Fig. 7 presents the solution to the problem by the supervised approach and the stand-alone optimisers. It can be seen that the solution obtained by the proposed approach is more accurate than the rest of the algorithms working alone. The fine-tuned solution of SPO, which in the last stage was found by an exploiter version of MCS, provides straight segments in between the obstacles, proving to be a balanced approach between exploration and exploitation. Although the closest competitor is the isolated MCS, its best solution crosses through an obstacle, suggesting that this optimiser has not converged in the imposed time constraint, but, it could refine the solution if it continues working. The poorest behaviour in this problem was performed by CMA-ES, which is capable of obtaining straight lines, but, the overall path shows large jumps between distant regions in the domain. Therefore, CMA-ES is well suited to accomplish local refinement, but, not capable of performing a satisfactory exploration. The isolated DE presents poor behaviour in this problem, however, there is a consistent solution found by this optimiser exhibited by the standard deviation in Table 1, which suggests that DE provides consistent solutions when integrated within the SPO.

4.2.3. SPO hyperparameter analysis

A hyperparameter analysis for the proposed SPO approach has been carried out to determine the sensitivity of the cost with respect to

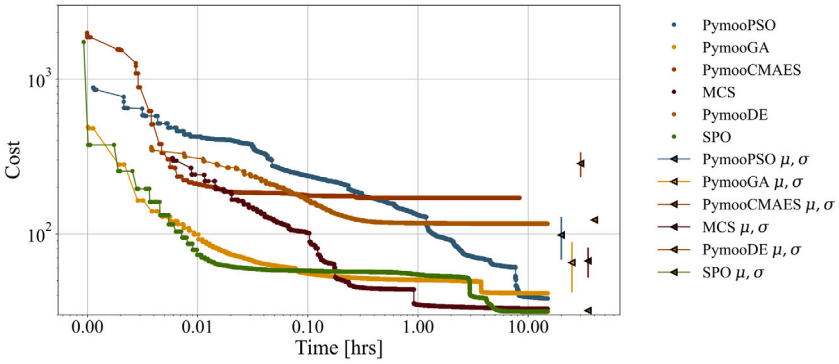


Fig. 6. Convergence comparison of the best solution obtained by stand-alone optimisers and the proposed approach. The mean μ and standard deviation σ of the solutions throughout the 10 experiments are computed using the last result obtained.

Table 1
Best, worst, mean, standard deviation and median by stand-alone optimisers and the proposed SPO.

Optimiser	Best	Mean	Worst	Std	Median
Pymoo PSO	38.2086	98.2314	216.5675	30.2439	97.5021
Pymoo GA	41.3171	65.2243	171.0270	23.3537	56.2161
Pymoo CMAES	171.0815	284.9940	417.0519	52.7251	285.3436
Pymoo DE	116.2521	123.0248	131.7709	3.5586	123.1253
MCS	32.8160	66.8578	108.5957	14.7948	65.4331
SPO	31.4619	31.9412	34.4326	0.9110	31.4635

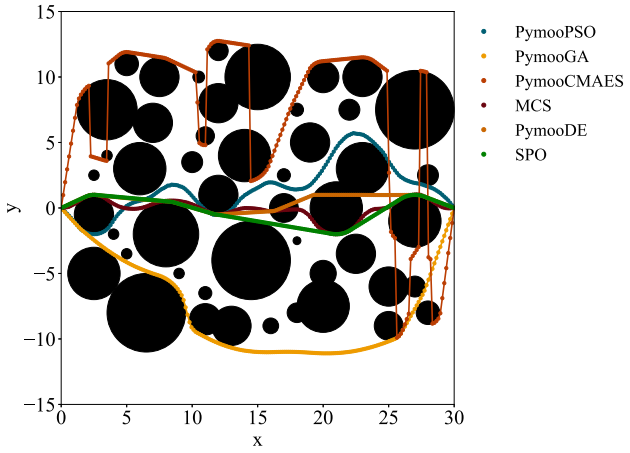


Fig. 7. Solution comparison of stand-alone optimisers against the proposed strategy.

relevant parameters of the proposed strategy, including (a) the exponent p, (b) the initial number of allowed stalled solutions $\bar{N}_{stall}$, both from Eq. (13), (c) the *stall tolerance* from Eq. (12), (d) the number of stall solutions per optimiser $N_{\bar{c}}$ from Eq. (14), (e) the *checkpoint* frequency, and (f) the seeding probability ν. Table 2 indicates a set of values for each parameter tested. Each experiment is set with a *base* configuration (column 3) and individual hyperparameters are modified to observe the behaviour of the strategy while solving the path-finding problem with the same conditions imposed in the previous section (design variables equal to $N_p = 200$, time limit = 15 h, $N_{workers} = 15$, $N_{CPUs} = 16$). Each configuration has been tested five times to obtain an adequate average behaviour.

Fig. 8(a) shows the best cost (dot), the mean (solid line) and the standard deviation (filled range) obtained using each hyperparameter setting. In combination with the base configuration, the cost does not exhibit an important variation with respect to the power parameter p, the number of stall solutions per optimiser $N_{\bar{c}}$, and the *checkpoint* (update frequency), whereas the initial number of stalled solutions

Table 2
SPO hyperparameters included in the sensitivity analysis for the path-finding problem.

SPO parameter	1	2	3	4	5
p	1	2	3	4	5
$\bar{N}_{stall}$	1	5	10	20	30
stall tolerance	0.9	0.1	0.01	0.001	0.001
$N_{\bar{c}}$	1	5	20	50	100
checkpoint	10	50	100	500	1000
ν	0.1	0.25	0.5	0.75	1.0

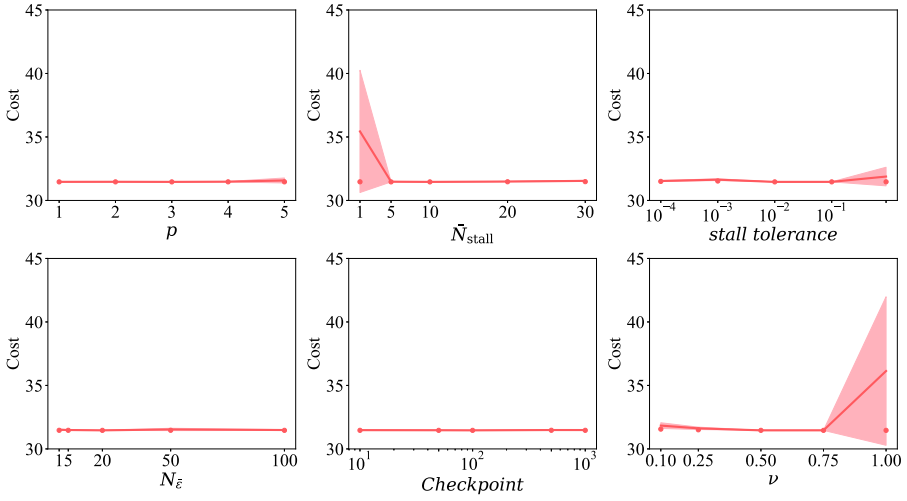
$\bar{N}_{stall}$ is stable from values greater than 5, the *stall tolerance* presents better behaviour with values lower than 10^{-1}, and the seeding probability ν is less prone to diverge for values smaller than 0.75. However, it can be noticed, that an optimal solution is achieved through all the configurations at least once for the path-finding problem. The results suggest that the proposed strategy is robust enough to obtain excellent solutions without the need for tedious hyperparameter tuning. In practical terms, a user does not require previous experience with the strategy and the problem intended to be solved to obtain an accurate response.

Furthermore, a second study was carried out focusing on the time of computation when the strategy achieves a *near optimal* solution with cost $\bar{c}$. The objective is to establish whether certain hyperparameter configurations are capable of yielding higher efficiency compared to others. A cost threshold has been defined as equal to $\bar{c} = 31.55$, which is close to the best solution achieved with cost $c = 31.462$. Fig. 8(b) shows the computation time that each hyperparameter configuration requires to overtake the threshold. It can be observed that the base configuration is the most efficient setting taking approximately 6 h to reach $\bar{c}$ and the remaining time to further refine the solution. This behaviour is seen in all the time-hyperparameter plots.

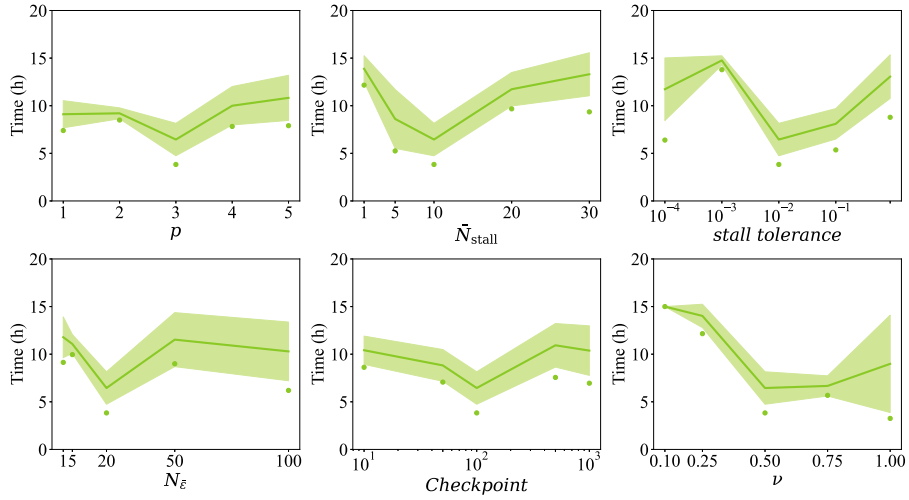
After performing the hyperparameter analysis for the path-finding problem, a recommended practical setting to solve a problem utilising the SPO strategy is to use the base configuration shown in column 3 of Table 2, and then start modifying the values according to the response observed. Recall that the individual optimiser hyperparameters are set according to the library default settings (Appendix C), with the exception of the population size that has been set to half of the number of design variables to provide sufficient exploration and to maintain algorithmic efficiency. A hyperparameter analysis of individual optimisers is out of the scope of this work, emphasising that the focus is to demonstrate the supervised strategy performance without interfering with internal optimisers' mechanisms. This is a more desired approach in practice when dealing with unexplored problems.

4.2.4. Algorithmic complexity

An additional measure of performance is the *algorithmic complexity*, in which an optimiser's efficiency can be tested based on the time



(a) Cost analysis.



(b) Computation time analysis.

Fig. 8. SPO hyperparameter analysis considering: (a) the solution achieved in 15 h of computation, and (b) the computation time of achieving a near-optimal solution threshold of 31.55. The best solution is represented by a dot, the mean by a solid line and the standard deviation by a filled range.

required to complete a number of function evaluations. A simple expression has been used by Kumar et al. [66] to compute the algorithmic complexity

$$AC_T = \frac{T_2 - T_1}{T_1} \tag{19}$$

where AC_T is the algorithmic complexity value calculated based on the required computation time by imposing a fixed number of evaluations, T_1 corresponds to the computation time required to perform N_{eval} number of objective function evaluations, and T_2 is the time required by an optimiser to compute the same N_{eval} number of function evaluations. Within the proposed parallel supervised approach, the individual optimisers are controlled by the supervisor and they can differ in number of evaluations. Hence, a modification of Eq. (19) is proposed

$$AC_E = \frac{E_1^w - E_2}{E_1^w} \tag{20}$$

where AC_E is the algorithmic complexity value obtained based on the number of evaluations in a fixed time, E_2 is the total number of function evaluations computed by the proposed parallel supervised strategy, and $E_1^w = E_1 \cdot N_{workers}$ is the number of objective function evaluations considering the number of parallel processors assigned to SPO. Note that the complexity obtained for the proposed strategy encompasses the complexity of each optimiser included in the ensemble, the supervisor-worker mechanisms (stopping criterion, seeding procedure) and the message exchange between processors. To compute the algorithmic complexity, the geometric path-finding problem defined in Section 4.2 was used as a test case for three imposed time limits of computation: 1 min, 30 min and 1 hr, and the number of workers used in this test is $N_{workers} = 10$ (11 CPUs). Table 3 summarises the number of function evaluations E_1^w, the supervised strategy function evaluations E_2 and the obtained algorithmic complexity AC_E for each corresponding time limit. The results obtained indicate that the complexity for 1 min is the lowest, and the complexity value is maintained for 30 min, and 1 hr, suggesting that in the 1st min there are not as many communications, whereas complexity reaches higher values up to a limit when the supervisor-worker dynamic and message exchange is occurring constantly. The number of evaluations ratio with respect to 1 min of computation (columns three and five) is a practical measure to identify the evaluation rate compared to the algorithmic complexity that saturates to a limit value.

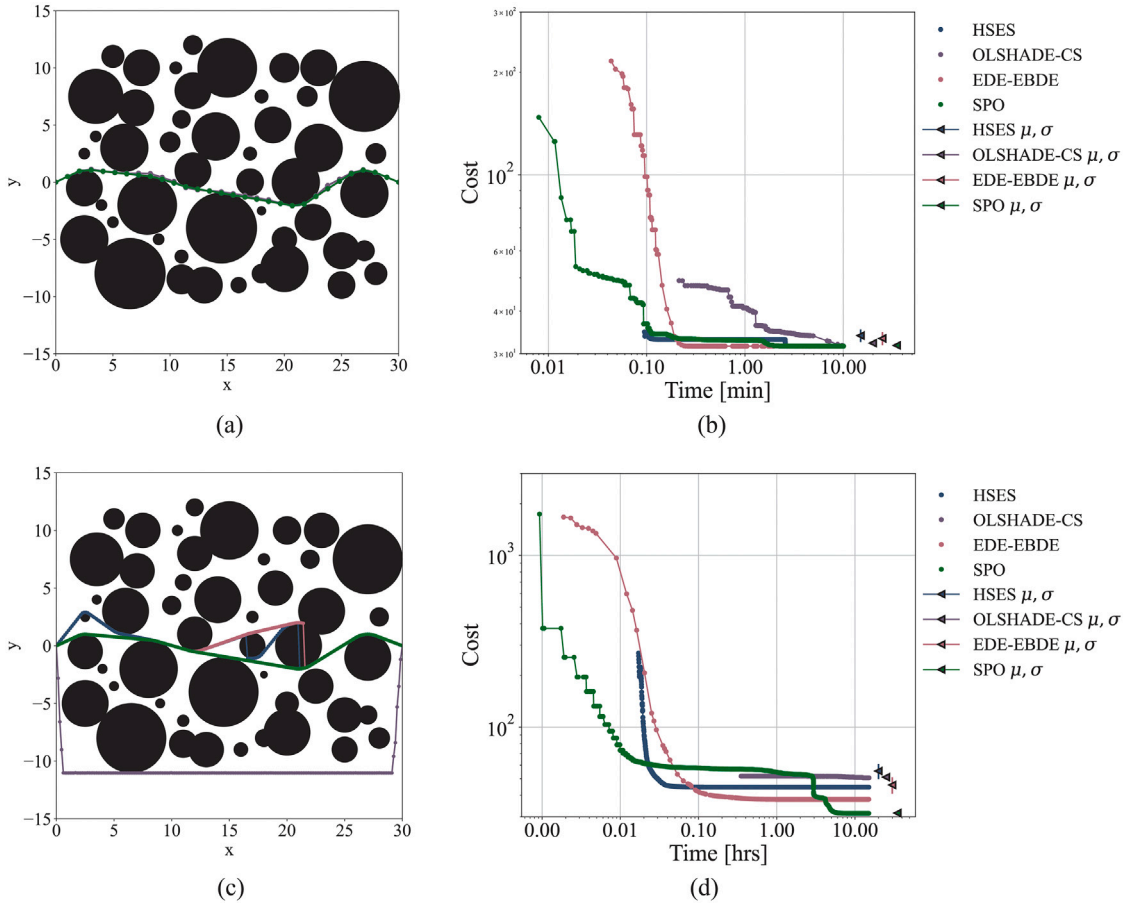


Fig. 9. Path solution comparison of state-of-the-art optimisers against the proposed supervised strategy and the convergence behaviour in the geometric path-finding problem, in which the mean μ and standard deviation σ of the solutions throughout the experiments are depicted for the last result. Figures (a) and (b) correspond to a 30 design variables problem, while Figures (c) and (d) correspond to the 200 design variables problem.

Table 3
Algorithmic complexity calculated for the path-finding problem considering three computation times.

Time	E_1^w	E_1^w/E_1^w(1 Min)	E_2	E_2/E_2(1 Min)	AC_E
1 min	194 960	1.0	171 493	1.0	0.12036
30 min	14 782 810	75.82	8 635 083	50.35	0.41587
1 h	29 965 120	153.70	16 915 512	98.64	0.43549

Table 4
Parameter setting for the three state-of-the-art algorithms used as a comparison in the path-finding problem for 30 and 200 design variables respectively.

Algorithm	30 design variables	200 design variables
HSES	Population Size $M = 200$; $N = 160$	Population Size $M = 900$; $N = 720$
OLSHADE-CS	$Q = 80$; $N_p^{init} = 6D^2$; $N_p^{init} = 4$	$Q = 220$; $N_p^{init} = D^2$; $N_p^{init} = 4$
EDE-EBDE	$L\_Rate = 0.7763$; $EDE\_best\_rate = 0.1264$; $Memory\_size = 6$	$L\_Rate = 0.8$; $EDE\_best\_rate = 0.1$; $Memory\_size = 5$

4.2.5. Comparison with state-of-the-art algorithms

The proposed supervised strategy has been compared against three state-of-the-art algorithms in the geometric path-finding problem considering two discretisations: 30 and 200 design variables. Two of the chosen optimisers are suggested in the comprehensive performance analysis of more than 500 metaheuristic algorithms provided by Ma et al. [67], corresponding to the Hybrid Sampling-Evolution Strategy HSES [68] and a hybrid of two enhanced DE variants named EDE-EBDE [69]. In addition, a third improved DE variant is considered, namely OLSHADE-CS [70] that presents an orthogonal array-based initialisation and a novel selection strategy. The parameter setting chosen for each algorithm corresponds to the suggested values in each published paper and from the exhaustive analysis done by Ma et al. [67]. These parameters are tuned for problems with 30 design variables which align with the first case of this analysis. The parameters for the 200 design variables are extended accordingly to provide a favourable comparison. The magnitudes of the parameters considered in each case are given in Table 4.

We test the efficiency of the proposed strategy and the state-of-the-art algorithms in moderate and large-scale problems. To this end, a 30 design variables path-finding problem is considered first as follows: The supervised approach is defined with $N_{workers} = 5$ and 10 repetitions to account for statistical analysis, hence, 50 runs have been performed for the three state-of-the-art optimisers to provide a reasonable comparison. Figs. 9 (a) and (b) show the path solution and the convergence behaviour, in which the proposed SPO approach and the three considered algorithms perform adequately as the path solution is accurate and the obstacles are avoided correctly. The best solution is achieved by the proposed SPO approach, EDE-EBDE and HSE, whereas OLSHADE-CS presents a sub-optimal solution with a small variation in length around two obstacles. The convergence plot shows that the best solution is achieved fastest by EDE-EBDE, followed by the proposed SPO approach, HSES, and OLSHADE-CS.

In the 200 design variables case, the supervised approach solution corresponds to the same analysis presented in Section 4.2 with $N_{workers} = 15$ and 10 repetitions, while the state-of-the-art algorithms

Table 5
Best, worst, mean, standard deviation and median by state-of-the-art optimisers and the proposed SPO in the geometric path-finding problem with 30 and 200 design variables.

	30 design variables					200 design variables				
Optimiser	Best	Mean	Worst	Std	Median	Best	Mean	Worst	Std	Median
HSES	31.5558	33.8702	37.3847	1.4227	31.9937	44.6498	55.4486	72.8450	5.5712	54.8743
OLSHADE-CS	31.7308	32.2090	33.0020	0.41088	31.9937	50.6037	51.0607	51.3590	0.1597	51.0504
EDE-EBDE	31.5557	33.2330	36.3829	1.4588	33.1395	37.8881	45.9686	59.5449	5.1231	44.6919
SPO	31.5561	31.7160	32.8325	0.3814	31.5600	31.4619	31.9412	34.4326	0.9110	31.4635

Table 6
Engineering applications with the corresponding number of design variables, constraints, reference objective function values, and the performance of the proposed SPO strategy and considered comparison algorithms, highlighting best, worst, mean, standard deviation and median of the solutions obtained [66].

Problem name	D	$N_g + N_h$	$f_R(x)$	Algorithm	Best	Mean	Worst	Std	Median
(a) Heat Exchanger Network Design (Case 1)	9	0	1.89E+02						
				SPO	1.89E+02	1.89E+02	1.89E+02	1.79E−08	1.89E+02
				IUDE	1.89E+02	2.29E+02	1.85E+02	8.06E+01	2.60E+02
				ϵMAgES	1.89E+02	4.55E+02	4.37E+02	2.23E+02	4.92E+02
				iLSHADE$_\epsilon$	1.90E+02	2.06E+02	2.29E+02	1.93E+01	1.94E+02
(b) Pressure vessel design	4	4	5.88E+03						
				SPO	5.74E+03	5.99E+03	6.05E+03	1.23E+01	6.05E+03
				IUDE	6.06E+03	6.06E+03	6.09E+03	6.16E+00	6.06E+03
				ϵMAgES	6.06E+03	7.38E+03	1.19E+04	1.93E+03	6.41E+03
				iLSHADE$_\epsilon$	6.06E+03	8.48E+03	1.49E+04	3.14E+03	6.11E+03
(c) Three-bar truss design problem	2	3	2.64E+02						
				SPO	2.64E+02	2.64E+02	2.64E+02	0.0	2.64E+02
				IUDE	2.64E+02	2.64E+02	2.64E+02	0.0	2.64E+02
				ϵMAgES	2.64E+02	2.65E+02	2.74E+02	2.88E+00	2.64E+02
				iLSHADE$_\epsilon$	2.64E+02	2.64E+02	2.64E+02	1.99E−02	2.64E+02
(d) Hydro-static thrust bearing design problem	4	7	1.62E+03						
				SPO	1.49E+03	1.52E+03	1.61E+03	4.81E+01	1.49E+03
				IUDE	1.86E+03	1.93E+03	2.60E+02	2.37E+03	2.60E+02
				ϵMAgES	1.62E+03	2.35E+03	6.34E+02	1.41E+03	2.26E+03
				iLSHADE$_\epsilon$	1.66E+03	1.76E+03	6.12E+02	1.28E+03	2.09E+03
(e) Himmelblau's Function	5	6	−3.07E+04						
				SPO	−3.07E+04	−3.07E+04	−3.07E+04	3.63E−12	−3.07E+04
				IUDE	−3.07E+04	−3.07E+04	−3.07E+04	3.71E−12	−3.07E+04
				ϵMAgES	−3.07E+04	−3.07E+04	−3.07E+04	3.56E−12	−3.07E+04
				iLSHADE$_\epsilon$	−3.07E+04	−3.07E+04	−3.07E+04	3.64E−12	−3.07E+04

are run 150 times. Note that all optimisers have a termination criterion defined as an imposed time limit of 15 h of computation. Figs. 9(c) and (d) show the achieved path solution and the convergence behaviour respectively. As can be seen, the three state-of-the-art algorithms failed to replicate the best solution obtained by the proposed SPO strategy. The algorithms EDE-EBDE and HSES are close to achieving an accurate solution, as both find a reasonable path, however, they crossed an obstacle between $x = 15$ and $x = 25$. In this test, OLSHADE-CS performs poorly, as the best solution found is a path that avoids the zone with obstacles, which results in a non-optimal path. Furthermore, the convergence behaviour of the algorithms suggests premature convergence of the three state-of-the-art algorithms as they reach a local minimum within the first hour of the optimisation process, whereas the proposed SPO strategy takes longer due to the exploration phase, but it eventually achieves the optimal path. The results obtained suggest that the solution of large-scale problems with multiple local minima, such as the path-finding problem with a fine discretisation, requires techniques to avoid premature stagnation and refinement of solutions, which are the two main objectives of the supervised strategy presented in this work.

Table 5 provides statistical results obtained by the solution of the path-finding problem with 30 and 200 design variables respectively. In the case of the 30 design variables, the four strategies are capable of reaching an adequate solution, however, the statistical mean shows that the proposed SPO strategy provides better solution in all experiments. When considering the 200 design variables problem, the high level of complexity becomes apparent, so the proposed SPO strategy is the only strategy to achieve the best solution. Furthermore, it should be noticed that the best solution found by any of the state-of-the-art algorithms is not better than the standalone MCS optimiser, and comparable to the PSO and GA tested in Section 4.2.4. However, their statistical mean is better and more consistent than the aforementioned standalone optimisers suggesting that a correct implementation of the state-of-the-art optimisers within the supervised approach are likely to benefit the efficiency of the overall SPO performance.

5. Selected engineering applications

In order to test the supervised parallel strategy performance on practical real-world problems, selected engineering applications from the test-suite suggested by Kumar et al. [66] are considered in this section. The complexity of these problems arises in the formulation of the objective function and the number of constraints involved. The test problems chosen are related to different fields such as industrial chemical processes, mechanical and civil engineering. Note that the constraint handling is done via penalisation which agrees with the current implementation of the ensemble algorithms. The constraints are handled as

$$\bar{g}(x) = \max(0, g(x))^2 \tag{21}$$

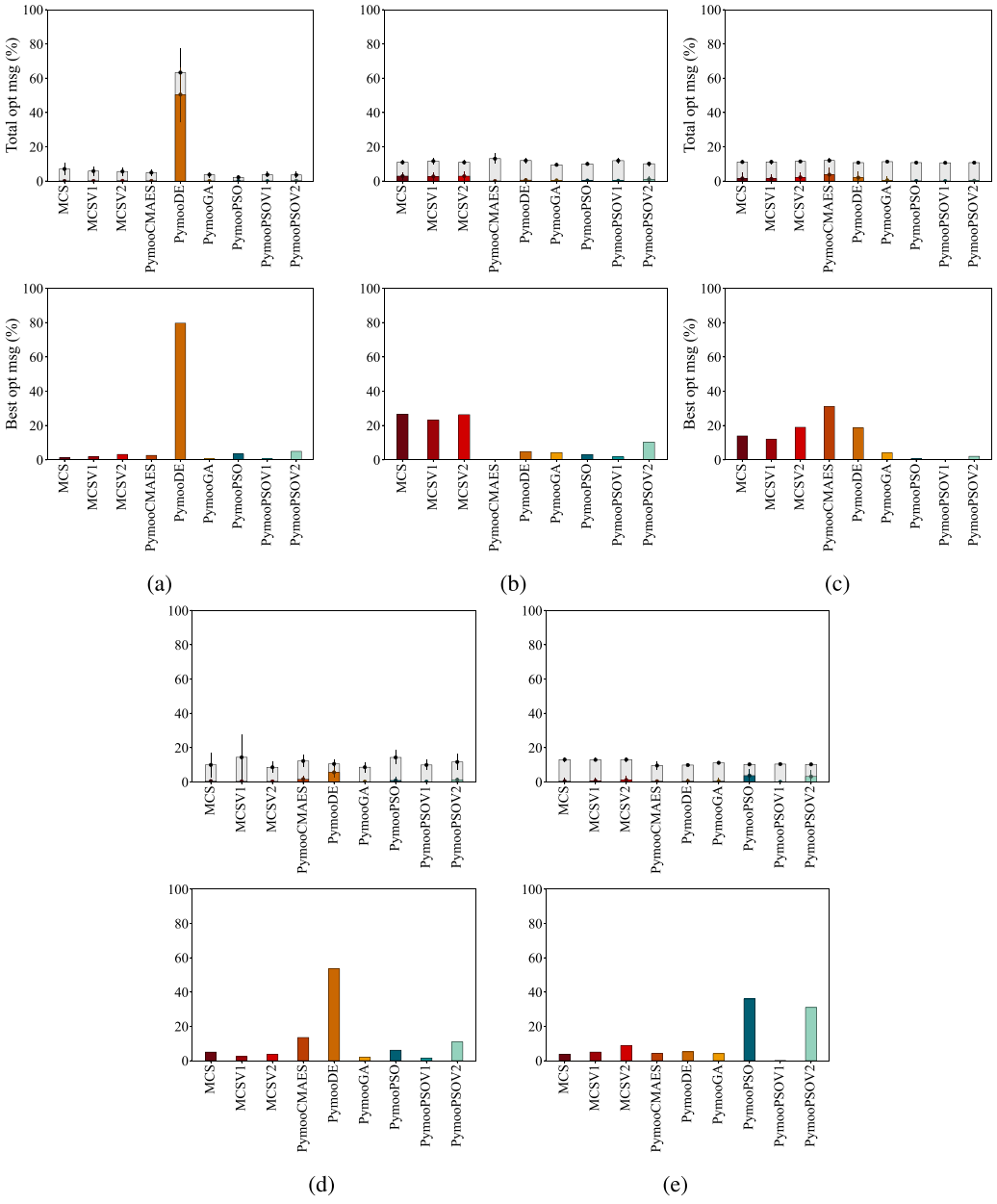


Fig. 10. Message analysis performed for the engineering applications test suit (a) - (e) from Table 6. The number of messages sent by the optimiser is displayed using grey bars, while the coloured bars represent the message when the optimiser is the best. (*Colour should be used.*).

$$\bar{h}(x) = (h(x))^2 \tag{22}$$

where x are the design variables, $g(x)$ is an inequality constraint with the shape $g(x) \leq 0$, $h(x)$ is an equality constraint, and $\bar{g}$ and $\bar{h}$ are soft inequality and equality constraints. Hence, the general objective function for the problems has the form

$$cost = f(x) + k\left(\sum_{i}^{N_g} \bar{g}_i(x) + \sum_{i}^{N_h} \bar{h}_i(x)\right) \tag{23}$$

where $f(x)$ is the objective function, N_g is the number of inequality constraints, $\bar{g}_i(x)$ are soft inequalities and $\bar{h}_i(x)$ are soft equalities, k is a penalty factor defining how *hard* the constraint is considered.

The selected engineering applications have been solved by the proposed SPO strategy and compared against three state-of-the-art algorithms namely Improved Unified Differential Evolution (IUDE) [71], Matrix Adaptation Evolution Strategy (ϵMAgES) [72] and LSHADE44 with an Improved ϵ Constraint-handling Method (iLSHADE$_\epsilon$) [73]. Each problem solved by SPO utilises the optimisers' hyper-parameters indicated in Appendix C, the SPO configuration and the magnitude of the penalty k per problem are indicated in Appendix D, the number of workers is $N_{workers} = 10$ ($N_{CPUs} = 11$), and 25 simulations are executed for each problem, whereas the solutions obtained by the comparison algorithms are reported in Kumar et al. [66].

Table 6 contains the problem name along with the number of design variables D, the number of constraints $N_g + N_h$, its reference objective function value $f_R(x)$, and summarises best, mean, worst, standard deviation and median solution achieved by the proposed strategy and the considered comparison algorithms. It can be observed that the results obtained for the engineering applications (a), (c), and (e) agree with the reference cost, whereas in cases (b), and (d) the cost obtained has been improved. In all cases, the mean, worst and median agree with the reference cost adequately, and the standard deviation obtained is remarkably small, indicating the high replicability of the solutions. In comparison with the considered algorithms, it can be noticed that

SPO has a comparable performance or in some cases, the solution is improved with respect to the selected engineering applications.

Additionally, a message analysis from SPO has been carried out to compare the individual optimiser performance. Fig. 10 displays two plots per engineering application (a)–(e): The top bar graph displays the total optimiser message percentage m_o with respect to all messages received by the supervisor and the bottom bar graph corresponds to the ratio of messages when the optimiser is the best with respect to the total number of messages delivered by that optimiser. This information exhibits the performance of the optimisers within the ensemble in different applications. It is considered that the number of messages is an indirect measure of the supervisor's decision to let an optimiser work for longer, and also the optimiser's speed in delivering messages. The ratio of best messages is an indicator of the optimiser's suitability for different problems.

In problem (a), DE has a greater frequency of sending messages resulting in a mean m_o above 60%, whereas the rest of the optimisers share a balanced number of messages. Considering the ratio of best optimiser messages, DE also has the best performance with approximately 80% of its messages being the best during the optimisation. The contribution of the other optimisers in this problem is minimal, with PSOV2 and PSO occupying the second and third places in the best message ratio analysis. In problem (b), a balanced number of messages is observed, and the best message percentage is shared by the three MCS versions. CMA-ES has a similar incidence of messages, but the number of best messages is zero suggesting that it has low algorithmic complexity, i.e. it is fast when computing and sending messages, but it never achieves a better solution than other optimisers. Problem (c) has one the most balanced number of optimiser messages in the considered applications test suit, indicating that the supervisor kept all the ensembled optimisers working equivalently. However, the best solution is sent by CMA-ES, followed by DE, MCSV2, MCS, MCSV1 and GA. In problem (d), the overall number of messages is balanced with the exception of MCSV1 which has the largest variance. In this problem, DE has the best message ratio compared to the other optimisers. In Problem (e), the supervisor kept a proportional number of messages among optimisers, however, the best ratio is achieved by PSO and PSOV2.

6. Conclusions

A supervised parallel optimisation approach is presented. This strategy couples established algorithms in a supervisor-worker structure. It uses the tools of monitoring, stopping and seeding to optimise the use of the available computational resources. The supervision effectively combines the exploration and exploitation capabilities of the different optimisers, providing a generalised framework suited to solve problems with diverse characteristics. Provided that the optimisation strategies followed by the workers include a variety of algorithms, the proposed supervised approach makes the success of the optimisation procedure independent of any tuning of hyperparameters, which is otherwise generally crucial.

The strategy performance has been tested with a geometric path-finding problem, which features a large number of design variables and a multitude of local minima. While none of the stand-alone procedures succeeded in finding the optimal solution, the proposed supervised strategy is capable of finding the minimal path length, which is constructed by straight lines, within the time limit. Thus, it has been demonstrated that the proposed supervised strategy is superior to the stand-alone algorithms by a large margin. Moreover, a hyperparameter analysis has been carried out exhibiting the robustness of the proposed strategy set with diverse configurations, and an algorithmic complexity analysis provides information on the performance of the strategy and its parallel communication exchange between supervisor and workers. In order to demonstrate the robustness and efficiency of the proposed SPO strategy a comparison with the state-of-the-art algorithms has been performed considering the path-finding problem. It is shown that the proposed SPO strategy provides the solution for an optimal path using both coarse and fine discretisation, while the state-of-the-art algorithms can obtain an accurate solution for coarse discretisation only that relies on the reduced set of design variables.

In order to validate the general applicability of the proposed strategy, a benchmark test suit composed of engineering applications is solved demonstrating that the proposed strategy can match, and in some cases improve, the standard algorithms with respect to the attainment of a reference cost. Furthermore, the performance of the type of optimisers included in the ensemble is measured to understand the internal mechanisms of the supervisor and provide a hint on the most suitable algorithms for the problems considered in this work. A notable application, in which the proposed supervised parallel optimisation strategy has recently shown promising results, is the training of recurrent neural networks [74].

CRediT authorship contribution statement

Eugenio J. Muttio: Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Software, Validation, Writing – original draft, Writing – review & editing. **Wulf G. Dettmer:** Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Supervision, Validation, Writing – original draft, Writing – review & editing. **Jac Clarke:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Validation, Writing – original draft. **Djordje Perić:** Conceptualization, Formal analysis, Investigation, Project administration, Supervision, Validation, Writing – original draft, Writing – review & editing. **Zhaoxin Ren:** Formal analysis, Resources, Supervision, Validation. **Lloyd Fletcher:** Formal analysis, Resources, Supervision, Validation.

Declaration of competing interest

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

Data availability

The link to the code release is cited in the Appendix A.

Acknowledgements

Eugenio J. Muttio gratefully acknowledges research support provided by UKAEA and EPSRC through the Doctoral Training Partnership (DTP) scheme. This work has been part-funded by the EPSRC Energy Programme [grant number EP/W006839/1].

We acknowledge the support of Supercomputing Wales and AccelerateAI projects, which are part-funded by the European Regional Development Fund (ERDF) via the Welsh Government.

Appendix A. Code release

The Supervised Parallel Optimisation (SPO) strategy presented in this work has been implemented in Python. The code is continuously maintained in a public GitHub repository [75] that can be easily accessed. However, as the code is developed, the results generated on a posterior date could differ from those presented in this work. A specific code release capable of generating the results presented can be found in a Zenodo repository [76].

The user is encouraged to read the documentation provided in the chosen version of the code (release/developer) in which the installation, usage and examples are detailed. The code will be made publicly available once the present work is published.

Table B.7
Circular obstacles location and radii defined within the domain of the problem.

Obstacle	Location		Radius
	x	y	
1	2.50	−5.00	2.00
2	3.50	7.50	2.30
3	2.50	−0.50	1.50
4	6.00	3.00	2.00
5	6.50	−8.00	3.00
6	7.00	6.50	1.50
7	8.00	−2.00	2.50
8	12.00	1.00	1.50
9	14.00	4.00	2.00
10	14.50	−4.00	3.00
11	15.00	10.00	2.50
12	21.00	0.00	2.00
13	22.50	−3.50	1.50
14	23.00	3.00	2.00
15	27.00	−1.00	2.00
16	19.00	5.00	1.50
17	20.00	−5.00	1.00
18	27.00	7.50	3.00
19	25.00	−6.00	1.50
20	17.00	2.50	0.50
21	12.00	8.00	1.50
22	11.00	5.50	0.70
23	20.00	−7.50	2.00
24	11.00	−8.50	1.20
25	13.00	−9.00	1.50
26	18.00	−8.00	0.75
27	23.00	10.00	1.50
28	10.00	3.50	0.80
29	20.00	10.00	1.20
30	22.00	7.50	0.80
31	28.00	2.50	0.80
32	17.00	0.00	1.10
33	18.00	−2.50	0.30
34	9.00	−5.00	0.40
35	11.00	−6.50	0.50
36	7.50	10.00	1.50
37	12.00	12.00	0.75
38	10.50	10.00	0.45
39	25.00	−9.00	1.10
40	18.00	7.50	0.50
41	16.00	−9.00	0.60
42	27.00	−6.00	0.80
43	28.00	−8.00	0.90
44	5.00	11.00	0.90
45	2.50	2.50	0.40
46	3.50	4.00	0.40
47	5.00	−3.50	0.40
48	4.00	−2.00	0.40

Appendix B. Definition of obstacles

The circular obstacles included in the domain of the problem are defined by the location of the centre and the radius. Table B.7 summarises the parameters to define the obstacles.

Appendix C. Optimiser hyperparameters

Suggested SPO and individual hyperparameters utilised in the solution of the path-finding problem of Section 4 are provided in this section. Note that for the explorer and exploiter versions of PSO and MCS optimisers, the strategy incorporates a *hyperparameters pool* that selects a random parameter value from a given range.

- **Supervised Parallel Optimisation**
  - Initial number of stalled messages $\bar{N}_{\text{stall}}$: 10
  - Exponent p: 3
  - Stall tolerance: 0.01
  - Stall average $N_{\bar{\epsilon}}$ per algorithm: 20
  - Number of top workers allowed to continue: 5
  - Seeding probability $\nu = 0.5$
- **Pymoo Genetic Algorithm**
  - Population size: 100
  - Number of offspring: 50
- **Pymoo CMA-ES**
  - Population size: 100
  - Initial standard deviation σ: 0.5
- **Pymoo PSO**
  - Population size: 100
  - Inertia ω: 0.9
  - Cognitive impact c_1: 2.0
  - Social impact c_2: 2.0
  - Max velocity rate: 0.2
  - Adaptive ω, c_1, c_2: *True*
- **Pymoo PSO V1 Explorer**
  - Population size: 100
  - Inertia ω: [0.5–0.9]
  - Cognitive impact c_1: [2.0–3.9]
  - Social impact c_2: [0.1–2.5]
  - Max velocity rate: 0.2
  - Adaptive ω, c_1, c_2: *False*
- **Pymoo PSO V2 Exploiter**
  - Population size: 100
  - Inertia ω: [0.1–0.6]
  - Cognitive impact c_1: [0.2–2.0]
  - Social impact c_2: [2.0–3.9]
  - Max velocity rate: 0.2
  - Adaptive ω, c_1, c_2: *False*
- **Modified Cuckoo Search**
  - Population size: 100
  - Minimum nests: 25
  - Discard fraction p_a: 0.7
  - Max step A: 100
  - Step size power pwr: 0.5
- **Modified Cuckoo Search V1 Explorer**
  - Population size: 100
  - Minimum nests: 25
  - Discard fraction p_a: [0.5–0.9]
  - Max step A: [10–1000]
  - Step size power pwr: [0.25–0.6]
- **Modified Cuckoo Search V2 Exploiter**
  - Population size: 100
  - Minimum nests: 25
  - Discard fraction p_a: [0.2–0.6]
  - Max step A: [1000–1000000]
  - Step size power pwr: [0.5–0.9]

Appendix D. Engineering applications details

Table D.8 summarises the SPO hyperparameters and penalty factors k utilised to solve each engineering application discussed in Section 5.

Table D.8
SPO hyperparameters and penalty factor k per each engineering application.

Problem name	p	$\bar{N}_{stall}$	*stall tolerance*	$N_{\bar{e}}$	*checkpoint*	ν	k
(a) Heat Exchanger Network Design (Case 1)	3	10	0.001	100	100	0.5	1000
(b) Pressure vessel design	3	3	0.001	10	10	0.5	1*e*6
(c) Three-bar truss design problem	3	10	0.01	10	10	0.5	1*e*6
(d) Hydro-static thrust bearing design problem	3	10	0.01	10	10	0.5	1*e*6
(e) Himmelblau's Function	3	5	0.01	5	100	0.5	1*e*6

References

[1] J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, control, and Artificial Intelligence, MIT Press, 1992.

[2] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.

[3] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, Vol. 4, 1995, pp. 1942–1948, http://dx.doi.org/10.1109/ICNN.1995.488968.

[4] N. Hansen, A. Ostermeier, Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1996, pp. 312–317, http://dx.doi.org/10.1109/ICEC.1996.542381.

[5] R. Storn, On the usage of differential evolution for function optimization, in: Proceedings of North American Fuzzy Information Processing, 1996, pp. 519–523, http://dx.doi.org/10.1109/NAFIPS.1996.534789.

[6] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (1997) 341–359.

[7] S. Das, S.S. Mullick, P.N. Suganthan, Recent advances in differential evolution – An updated survey, Swarm Evol. Comput. 27 (2016) 1–30.

[8] K.R. Opara, J. Arabas, Differential Evolution: A survey of theoretical analyses, Swarm Evol. Comput. 44 (2019) 546–558.

[9] X.S. Yang, S. Deb, Cuckoo search via Lévy flights, in: 2009 World Congress on Nature Biologically Inspired Computing, NaBIC, 2009, pp. 210–214, http://dx.doi.org/10.1109/NABIC.2009.5393690.

[10] X.S. Yang, A new metaheuristic bat-inspired algorithm, in: Nature Inspired Cooperative Strategies for Optimization, NICSO 2010, in: Studies in Computational Intelligence, Springer, Berlin, Heidelberg, 2010, pp. 65–74, http://dx.doi.org/10.1007/978-3-642-12538-6_6.

[11] X.S. Yang, Firefly algorithm, stochastic test functions and design optimisation, Int. J. Bio-Inspir. Comput. 2 (2) (2010) 78–84.

[12] I. Fister, I. Fister Jr., X.S. Yang, J. Brest, A comprehensive review of firefly algorithms, Swarm Evol. Comput. 13 (2013) 34–46.

[13] H. Abedinpourshotorban, S. Mariyam Shamsuddin, Z. Beheshti, D.N.A. Jawawi, Electromagnetic field optimization: A physics-inspired metaheuristic optimization algorithm, Swarm Evol. Comput. 26 (2016) 8–22.

[14] S. Mirjalili, A. Lewis, The whale optimization algorithm, Adv. Eng. Softw. 95 (2016) 51–67.

[15] F.S. Gharehchopogh, H. Gholizadeh, A comprehensive survey: Whale Optimization Algorithm and its applications, Swarm Evol. Comput. 48 (2019) 1–24.

[16] M. Jain, V. Singh, A. Rani, A novel nature-inspired algorithm for optimization: Squirrel search algorithm, Swarm Evol. Comput. 44 (2019) 148–175.

[17] M. Rocha, J. Neves, Preventing premature convergence to local optima in genetic algorithms via random offspring generation, in: I. Imam, Y. Kodratoff, A. El-Dessouki, M. Ali (Eds.), Multiple Approaches to Intelligent Systems, Springer, Berlin, Heidelberg, 1999, pp. 127–136.

[18] C. Vanaret, J.-B. Gotteland, N. Durand, J.-M. Alliot, Preventing premature convergence and proving the optimality in evolutionary algorithms, in: P. Legrand, M.-M. Corsini, J.-K. Hao, N. Monmarché, E. Lutton, M. Schoenauer (Eds.), Artificial Evolution, Springer International Publishing, 2014, pp. 29–40.

[19] M. Bhattacharya, A synergistic approach for evolutionary optimization, in: Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 2105–2110, http://dx.doi.org/10.1145/1388969.1389031.

[20] B. Yang, Y. Chen, Z. Zhao, A hybrid evolutionary algorithm by combination of PSO and GA for unconstrained and constrained optimization problems, in: 2007 IEEE International Conference on Control and Automation, 2007, pp. 166–170, http://dx.doi.org/10.1109/ICCA.2007.4376340.

[21] P. Ghamisi, J.A. Benediktsson, Feature selection based on hybridization of genetic algorithm and particle swarm optimization, IEEE Geosci. Remote Sens. Lett. 12 (2) (2015) 309–313.

[22] F. Zhao, Q. Zhang, D. Yu, X. Chen, Y. Yang, A hybrid algorithm based on PSO and simulated annealing and its applications for partner selection in virtual enterprise, in: D.-S. Huang, X.-P. Zhang, G.-B. Huang (Eds.), Advances in Intelligent Computing, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2005, pp. 380–389, http://dx.doi.org/10.1007/11538059_40.

[23] N. Sadati, T. Amraee, A.M. Ranjbar, A global Particle Swarm-Based-Simulated Annealing Optimization technique for under-voltage load shedding problem, Appl. Soft Comput. 9 (2) (2009) 652–657.

[24] A. Ghodrati, S. Lotfi, A hybrid CS/PSO algorithm for global optimization, in: J.-S. Pan, S.-M. Chen, N.T. Nguyen (Eds.), Intelligent Information and Database Systems, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 89–98, http://dx.doi.org/10.1007/978-3-642-28493-9_11.

[25] R. Chi, Y.-x. Su, D.-h. Zhang, X.-x. Chi, H.-j. Zhang, A hybridization of cuckoo search and particle swarm optimization for solving optimization problems, Neural Comput. Appl. 31 (1) (2019) 653–670.

[26] X. Li, M. Yin, A particle swarm inspired cuckoo search algorithm for real parameter optimization, Soft Comput. 20 (4) (2016) 1389–1413.

[27] J. Dash, B. Dam, R. Swain, Optimal design of linear phase multi-band stop filters using improved cuckoo search particle swarm optimization, Appl. Soft Comput. 52 (2017) 435–445.

[28] T. Hendtlass, A combined swarm differential evolution algorithm for optimization problems, in: Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Engineering of Intelligent Systems, in: IEA/AIE '01, Springer-Verlag, Berlin, Heidelberg, 2001, pp. 11–18.

[29] W.-J. Zhang, X.-F. Xie, DEPSO: Hybrid particle swarm with differential evolution operator, in: SMC03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance. Vol. 4, 2003, pp. 3816–3821, http://dx.doi.org/10.1109/ICSMC.2003.1244483.

[30] R. Xu, J. Xu, D.C. Wunsch, Clustering with differential evolution particle swarm optimization, in: IEEE Congress on Evolutionary Computation, 2010, pp. 1–8, http://dx.doi.org/10.1109/CEC.2010.5586257.

[31] Z. Wang, Y. Chen, S. Ding, D. Liang, H. He, A novel particle swarm optimization algorithm with Lévy flight and orthogonal learning, Swarm Evol. Comput. 75 (2022) 101207.

[32] A. Hatamlou, S. Abdullah, H. Nezamabadi-pour, A combined approach for clustering based on K-means and gravitational search algorithms, Swarm Evol. Comput. 6 (2012) 47–52.

[33] F. Neri, C. Cotta, Memetic algorithms and memetic computing optimization: A literature review, Swarm Evol. Comput. 2 (2012) 1–14.

[34] G. Wu, R. Mallipeddi, P.N. Suganthan, Ensemble strategies for population-based optimization algorithms – A survey, Swarm Evol. Comput. 44 (2019) 695–711.

[35] M.Z. Ali, N.H. Awad, P.N. Suganthan, Multi-population differential evolution with balanced ensemble of mutation strategies for large-scale global optimization, Appl. Soft Comput. 33 (2015) 304–327.

[36] G. Wu, R. Mallipeddi, P.N. Suganthan, R. Wang, H. Chen, Differential evolution with multi-population based ensemble of mutation strategies, Special issue on Discovery Science, Inform. Sci. 329 (2016) 329–345.

[37] S.X. Zhang, S.Y. Zheng, L.M. Zheng, An efficient multiple variants coordination framework for differential evolution, IEEE Trans. Cybern. 47 (2017) 2780–2793.

[38] G. Wu, X. Shen, H. Li, H. Chen, A. Lin, P. Suganthan, Ensemble of differential evolution variants, Inform. Sci. 423 (2018) 172–186.

[39] P. Singh, R. Kottath, An ensemble approach to meta-heuristic algorithms: Comparative analysis and its applications, Comput. Ind. Eng. 162 (2021) 107739.

[40] S.M. Elsayed, R.A. Sarker, E. Mezura-Montes, Self-adaptive mix of particle swarm methodologies for constrained optimization, Inform. Sci. 277 (2014) 216–233.

[41] N. Lynn, P.N. Suganthan, Ensemble particle swarm optimizer, Appl. Soft Comput. 55 (2017) 533–548.

[42] A.P. Engelbrecht, Heterogeneous particle swarm optimization, in: Swarm Intelligence, Vol. 6234, Springer, Berlin, Heidelberg, 2010, pp. 191–202, http://dx.doi.org/10.1007/978-3-642-15461-4_17.

[43] J. Robinson, S. Sinton, Y. Rahmat-Samii, Particle swarm, genetic algorithm, and their hybrids: optimization of a profiled corrugated horn antenna, in: IEEE Antennas and Propagation Society International Symposium. Vol. 1, IEEE Cat. No.02CH37313, 2002, pp. 314–317, http://dx.doi.org/10.1109/APS.2002.1016311.

[44] J.A. Vrugt, B.A. Robinson, J.M. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Trans. Evol. Comput. 13 (2) (2009) 243–259.

[45] J. Zhang, T.-S. Pan, J.-S. Pan, A parallel hybrid evolutionary particle filter for nonlinear state estimation, in: 2011 First International Conference on Robot, Vision and Signal Processing, 2011, pp. 308–312, http://dx.doi.org/10.1109/RVSP.2011.77.

[46] Y. Xue, S. Zhong, Y. Zhuang, B. Xu, An ensemble algorithm with self-adaptive learning techniques for high-dimensional numerical optimization, Appl. Math. Comput. 231 (2014) 329–346.

[47] S.Y. Yuen, X. Zhang, On composing an algorithm portfolio, Memet. Comput. 7 (3) (2015) 203–214.
[48] S. Elsayed, N. Hamza, R. Sarker, Testing united multi-operator evolutionary algorithms-II on single objective optimization problems, in: 2016 IEEE Congress on Evolutionary Computation, CEC, 2016, pp. 2966–2973, http://dx.doi.org/10.1109/CEC.2016.7744164.
[49] A.A. Nik, F.M. Nejad, H. Zakeri, Hybrid PSO and GA approach for optimizing surveyed asphalt pavement inspection units in massive network, Autom. Constr. 71 (2016) 325–345.
[50] H. Garg, A hybrid PSO-GA algorithm for constrained optimization problems, Appl. Math. Comput. 274 (2016) 292–305.
[51] K. Wansasueb, S. Bureerat, S. Kumar, Ensemble of four metaheuristic using a weighted sum technique for aircraft wing design, Eng. Appl. Sci. Res. 48 (4) (2021) 385–396.
[52] J.F. Schutte, J.A. Reinbolt, B.J. Fregly, R.T. Haftka, A.D. George, Parallel global optimization with the particle swarm algorithm, Internat. J. Numer. Methods Engrg. 61 (13) (2004) 2296–2315.
[53] J.-F. Chang, S.-C. Chu, J. Roddick, J.-S. Pan, A parallel particle swarm optimization algorithm with communication strategies, J. Inf. Sci. Eng. 21 (2005) 809–818.
[54] G. Venter, J. Sobieszczanski-Sobieski, Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations, J. Aerosp. Comput. Infor. Commun. 3 (3) (2006) 123–137.
[55] M. Waintraub, R. Schirru, C.M.N.A. Pereira, Multiprocessor modeling of parallel Particle Swarm Optimization applied to nuclear engineering problems, Prog. Nucl. Energy 51 (6) (2009) 680–688.
[56] J. Blank, K. Deb, pymoo: Multi-objective optimization in Python, IEEE Access 8 (2020) 89497–89509.
[57] S. Walton, O. Hassan, K. Morgan, M. Brown, Modified cuckoo search: a new gradient free optimisation algorithm, Chaos, Solitions Fract. 44 (2011) 710–718.
[58] K. Deb, A. Kumar, Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems, Complex Systems (1995) 431–454.
[59] K. Deb, D. Deb, Analysing mutation schemes for real-parameter genetic algorithms, Int. J. Artif. Intell. Soft Comput. 4 (1) (2014) 1–28.
[60] N. Hansen, The CMA evolution strategy: A comparing review, in: Towards a New Evolutionary Computation. Studies in Fuzziness and Soft Computing. Vol. 192, 2007, pp. 75–102, http://dx.doi.org/10.1007/3-540-32494-1_4.
[61] J. Jin, C. Yang, Y. Zhang, An improved CMA-ES for solving large scale optimization problem, in: Y. Tan, Y. Shi, M. Tuba (Eds.), Advances in Swarm Intelligence, Springer International Publishing, 2020, pp. 386–396.
[62] Y.Y. Fan, R.E. Kalaba, J.E. Moore, Arriving on Time, J. Optim. Theory Appl. 127 (3) (2005) 497–513.
[63] Y. Nie, Y. Fan, Arriving-on-time problem: Discrete algorithm that ensures convergence, Transp. Res. Rec. 1964 (1) (2006) 193–200.
[64] M. Niknami, S. Samaranayake, Tractable pathfinding for the stochastic on-time arrival problem, in: Experimental Algorithms, in: Lecture Notes in Computer Science, Springer International Publishing, 2016, pp. 231–245, http://dx.doi.org/10.1007/978-3-319-38851-9_16.
[65] Y. Liu, S. Blandin, S. Samaranayake, Stochastic on-time arrival problem in transit networks, Transp. Res. B 119 (2019) 122–138.
[66] A. Kumar, G. Wu, M.Z. Ali, R. Mallipeddi, P.N. Suganthan, S. Das, A test-suite of non-convex constrained optimization problems from the real-world and some baseline results, Swarm Evol. Comput. 56 (2020) 100693.
[67] Z. Ma, G. Wu, P.N. Suganthan, A. Song, Q. Luo, Performance assessment and exhaustive listing of 500+ nature-inspired metaheuristic algorithms, Swarm Evol. Comput. 77 (2023) 101248.
[68] G. Zhang, Y. Shi, Hybrid sampling evolution strategy for solving single objective bound constrained problems, in: 2018 IEEE Congress on Evolutionary Computation, CEC, IEEE, Rio de Janeiro, 2018, pp. 1–7, http://dx.doi.org/10.1109/CEC.2018.8477908.
[69] A.W. Mohamed, A.A. Hadi, K.M. Jambi, Novel mutation strategy for enhancing SHADE and LSHADE algorithms for global numerical optimization, Swarm Evol. Comput. 50 (2019) 100455.
[70] A. Kumar, P.P. Biswas, P.N. Suganthan, Differential evolution with orthogonal array-based initialization and a novel selection strategy, Swarm Evol. Comput. 68 (2022) 101010.
[71] A. Trivedi, D. Srinivasan, N. Biswas, An improved unified differential evolution algorithm for constrained optimization problems, in: Proceedings of 2018 IEEE Congress on Evolutionary Computation, IEEE, 2018, pp. 1–10.
[72] M. Hellwig, H.-G. Beyer, A matrix adaptation evolution strategy for constrained real-parameter optimization, in: 2018 IEEE Congress on Evolutionary Computation, CEC, 2018, pp. 1–8, http://dx.doi.org/10.1109/CEC.2018.8477950.
[73] Z. Fan, Y. Fang, W. Li, Y. Yuan, Z. Wang, X. Bian, LSHADE44 with an improved constraint-handling method for solving constrained single-objective optimization problems, in: 2018 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2018, pp. 1–8, http://dx.doi.org/10.1109/CEC.2018.8477943.
[74] W.G. Dettmer, E.J. Muttio, R. Alhayki, D. Perić, A framework for neural network based constitutive modelling of inelastic materials, Comput. Methods Appl. Mech. Eng. (2023) http://dx.doi.org/10.1016/j.cma.2023.116672.
[75] E.J. Muttio, Supervised parallel optimisation framework, 2023, URL: https://github.com/EugenioMuttio/SPO.git.
[76] E.J. Muttio, W.G. Dettmer, J. Clarke, D. Perić, Z. Ren, L. Fletcher, Supervised Parallel Optimisation Framework, Zenodo, 2023, http://dx.doi.org/10.5281/zenodo.8005337.