UKAEA-CCFE-PR(23)142

E. J. Muttio, W. G. Dettmera, J. Clarke, D. Peric, Z. Ren and L. Fletcher

A Supervised Parallel Optimisation Framework for Metaheuristic Algorithms

A Supervised Parallel Optimisation Framework for Metaheuristic Algorithms

E. J. Muttio, W. G. Dettmera, J. Clarke, D. Peric, Z. Ren and L. Fletcher

This is a preprint of a paper submitted for publication in Advances in Engineering Software

Highlights

A Supervised Parallel Optimisation Framework for Metaheuristic Algorithms

Eugenio J. Muttio,Wulf G. Dettmer,Jac Clarke,Djordje Perić,Zhaoxin Ren,Lloyd Fletcher

- A novel Supervised Parallel Optimisation (SPO) balances exploration and exploitation of distinct optimisers to solve problems with diverse characteristics.
- The proposed SPO efficiently ensembles four optimisation algorithms (PSO, GA, CMAES, MCS), however, it can be easily extended to any optimisation algorithm.
- The supervised strategy outperforms isolated algorithms, finding reproducible, optimal solutions to a complex path finding problem with numerous local minima.
- The generalised framework of the proposed strategy reduces the necessity of tedious hyperparameter fine tuning of independent optimisers by incorporating a reduced number of supervisor's parameters.

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] and 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, four optimisation algorithms are included in the ensemble: Particle Swarm Optimisation (PSO), Genetic Algorithm (GA), Covariance Matrix Adaption - Evolution Strategy (CMA-ES), and Modified Cuckoo Search (MCS). A 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. The good solution generated by SPO is 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 many more. In addition, a thriving interest in optimisation has been observed in the last two decades due to the advances in machine learning, where the *training* stage of most of these methods involve searching for an optimal solution. Hence, the optimisation field is not static, but actively changing according to emerging technology. A traditional optimisation approach takes into account the gradient of the objective function to determine a possible direction of the solution. However, real-life 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* have emerged since the mid-late last century but exponentially increased in the last few decades due to their success. In general, a meta-heuristic algorithm is characterised by initialising a random population of agents which develop through generations to find a better position in the solution space. The selection process is based on each agent's fitness (function evaluation), and, may contain operations like 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], and more recently, cuckoo search (CS) [6]. However, the list keeps growing since novel strategies and variations of them are being developed continuously. Challenges to be addressed include the problem dependent suitability and performance of meta-heuristic, premature convergence [7–9], 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 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) [10, 11], a simulated annealing and PSO hybrid approach [12, 13], cuckoo search (CS) inspired by PSO [14–16], a CS-PSO hybrid with DE for global search [17], a DE and PSO combination [18–20], and many more. An alternative strategy to combine the special features of algorithms is by running them independently but including merging or seeding processes of their populations. Such strategies are commonly referred to as *Ensemble strategies*, see for instance [21–26]. A single-optimiser ensemble strategy is introduced in [27, 28] by including a behaviour pool. Due to the high computational effort required by real-life problems, parallel optimisation is undoubtedly needed. Numerous studies on

*Corresponding author
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)
ORCID(s): 0000-0002-7555-9023 (E.J. Muttio); 0000-0003-0799-4645 (W.G. Dettmer); 0009-0009-0624-8991 (J. Clarke); 0000-0002-1112-301X (D. Perić); 0000-0002-6305-9515 (Z. Ren); 0000-0002-6305-9515 (L. Fletcher)

communication in a parallel setting for optimisation are found in literature, including the efficiency between processors [29], the correlation of variables in objective functions [30], parallel architectures [31, 32], among others.

The objective of this work is the development of a novel generalised strategy for real-life optimisation problems. The strategy is capable of coupling multiple independent optimisation algorithms executed in a supervised manner by using parallel computation, therefore, it is named Supervised Parallel Optimisation (SPO). A geometric path finding problem is employed to demonstrate the main features and capabilities of the proposed strategy. The objective is to minimise the path length subject to avoiding the penetration of any of the large number of obstacles. While the implementation in this work is based on Python, the algorithmic structure described is easily extended to any programming language. Although this work includes four optimisers only, Python facilitates the inclusion of various meta-heuristics. Hence, an established Python multi-objective optimisation library (Pymoo) [33] has been utilised to incorporate a genetic algorithm (GA), a particle swarm optimisation (PSO) and a covariance matrix adaptation evolution strategy (CMA-ES). A Python version of the modified cuckoo search (MCS) is adapted from [34] due to the outstanding performance exhibited. It is important to note that the strategy proposed in this article is not meant to compete with any specific evolutionary optimisation procedure, but is designed to solve or, at least, to solve more efficiently large and challenging problems.

This article is organised as follows: The four optimisation algorithms included in this ensemble approach are described in Section 2, which include PSO, GA, CMA-ES and MCS. The proposed supervised parallel optimisation strategy is introduced in Section 3, where the general structure and the two crucial mechanisms of SPO are fully described. In Section 4 the path finding optimisation problem is defined, the performance of the proposed methodology is tested, and, a comparison exercise is carried out by contrasting the results obtained by the included algorithms. Finally, conclusions are summarised in Section 5.

2. Meta-heuristic Algorithms

2.1. Particle Swarm Optimisation (PSO)

Particle Swarm Optimisation (PSO) was first introduced in [3], and is considered a reference among the so-called *swarm intelligence* methods due to its simplicity and speed. This method was inspired by the behaviour of swarming creatures in nature, such as bird flocking and fish schooling. In PSO, each member of the population, or “particle”, has a position that lies within the specified design space and represents a potential solution. This position has an associated fitness, or “cost”, which is defined by the objective function. The population is first initialised randomly, providing each particle with a starting position in the design space. Then, each particle’s position is updated iteratively until a termination criterion is reached, such as a predefined maximum number of generations. The swarm converges towards the best region of the design space 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 particles inertia. The velocities V_d of the particles are updated by

$$V_d^{(i)} = \omega V_d^{(i)} + c_1 r_1 (P_d^{(i)} - X_d^{(i)}) + c_2 r_2 (G_d^{(i)} - X_d^{(i)}) \quad (1)$$

where P_d is the particle’s local best position, G_d is the swarm global best position, X_d 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. These weighting coefficients can be selected to control the behaviour of the swarm, with respect to the previously described set of influences. They can be used to enhance the local or global exploitation of the algorithm, by increasing c_1 and c_2 or they can be used to encourage exploration within the swarm by increasing ω. Following the calculation of the velocity from Equation (1), the position X_d of the particles is updated by

$$X_d^{(i)} = X_d^{(i)} + V_d^{(i)} \quad (2)$$

2.2. Genetic Algorithm (GA)

The Genetic Algorithm (GA) is the most widely used and known evolutionary algorithm, taking inspiration from the theory of natural selection and evolution by Charles Darwin. The algorithm was first introduced in the 1960s and 1970s by Professor John Holland of the University of Michigan and his collaborators [1]. The essential characteristics of GA include the representation of individuals as chromosomes, the manipulation of these by genetic operators, and the 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 increase the diversity; and a *selection* that allows propagating the best solutions on to next generations. A desired behaviour presented in GA is that, as the process evolves, multiple offspring can explore diverse regions of the search space alleviating premature convergence problems. Numerous GA variants have been presented since its introduction, focused especially on the improvement of the genetic operators.

2.3. CMA-ES Algorithm

Evolution strategies (ES) were created in the 1960s and further developed by Rechenberg and Schwefel in the 1970s, and are algorithms based on the use of mutation and selection mechanisms. In 1996, Hansen and Ostermeier proposed a new formulation named covariance matrix adaptation evolution strategy (CMA-ES) [4]. 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. This leads to a similar approximation of the inverse Hessian matrix in the classical quasi-Newton optimisation method. This method has several desirable *invariance properties* including order transformation of the objective function and angle preserving transformations of the search space, both of which imply uniform behaviour on classes of functions. In addition, 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 already be solved with a small number of function evaluations. However, as indicated in [35], CMA-ES has disadvantages such as premature stagnation when solving large-scale optimisation problems.

2.4. Modified Cuckoo Search (MCS)

The standard cuckoo search (CS) algorithm was introduced in [6], inspired by the brood parasitism of certain cuckoo bird species and by the foraging and flight behaviour exhibited by many animals such as birds and insects. 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 allows for a fraction of the poorer quality eggs to be removed from nests after a generation, making room for new eggs to be laid. The simplest approach is to consider that each nest has only a single egg, which represents an individual containing a position in the design space. This algorithm combines local and global random walks, where the latter is carried out by the so-called Lévy flights i.e.

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha \oplus \text{Lévy}(\lambda) \tag{3}$$

where $\alpha > 0$ 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) was introduced to improve the performance of the original algorithm [34]. A number of modifications were made, including a decreasing α coefficient, which enhances exploitation as the agents evolves toward a potentially better solution and a crossover mechanism between the current solutions. MCS has been shown to outperform standard CS and exhibits a significantly better convergence rate than PSO in many applications.

3. Supervised Parallel Framework

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 or MCS. Each worker starts an isolated optimisation algorithm, i.e. runs a standalone 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.

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 Figure 1 and are further explained in the following sections.

3.2. Stopping Criteria

The workers report regularly their best cost and solution to the supervisor at each checkpoint (every N_{gen} generations). The supervisor monitors the current solution sent by each worker and keeps the history of the previously sent 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

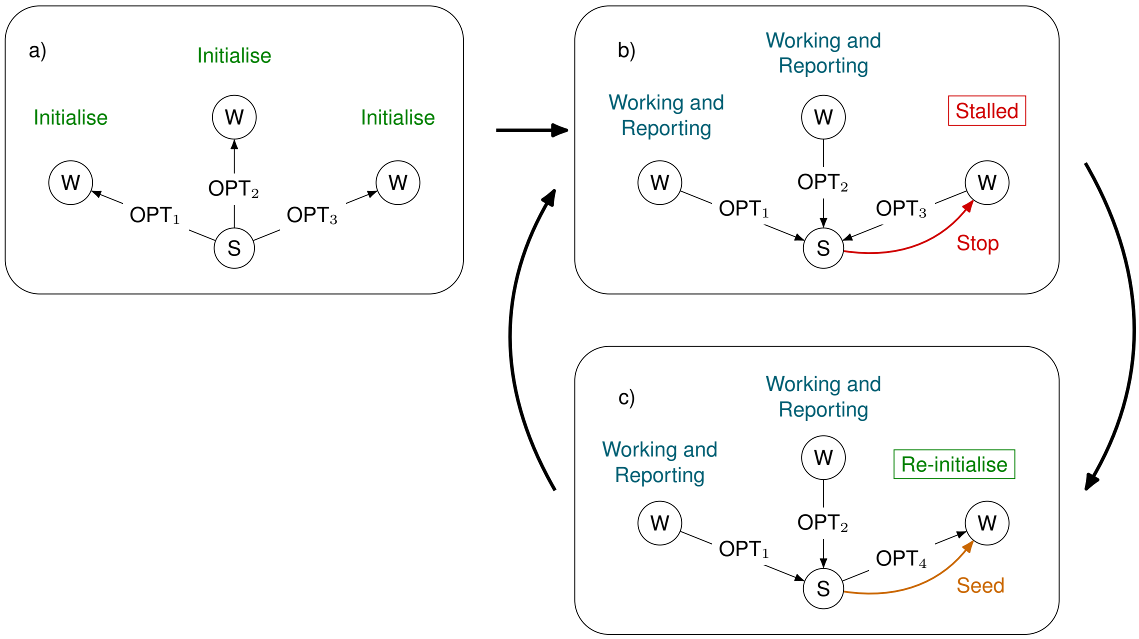

Figure 1: Supervised parallel structure and roles of the processors in the proposed strategy. Three stages are depicted: a) processor initialisation by the supervisor (S), b) workers (W) report their performance to the supervisor (S) and supervisor stops stalled workers, and, c) the supervisor re-initialises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

have been completed. The criterion used by the supervisor to detect stall can be written as

$$\frac{\epsilon_m}{\epsilon_{m-N_{\text{stall}}}} > 1 - tolerance \qquad \Rightarrow \qquad \text{Optimisation has stalled.} \tag{4}$$

where ϵ_m is the m-th cost reported to the supervisor by the corresponding worker. The critical number of checkpoints reached without sufficient improvement N_{stall} is calculated from

$$N_{\text{stall}} = \bar{N}_{\text{stall}} \left(\frac{\bar{\epsilon}}{\epsilon_m} \right)^p \tag{5}$$

where $\bar{N}_{\text{stall}}$ is an initial number of stalled solutions allowed. The exponent p may be chosen as 1, 2 or 3 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 optimisers.

$$\bar{\epsilon} = \frac{1}{N_{alg}} \sum_{i=1}^{N_{alg}} \epsilon_m^i \tag{6}$$

where N_{alg} is the number of different optimisation algorithms run by the workers.

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 checkpoints allowed will increase as stated by Equation (5). Algorithm 1 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 and arranges them according to the cost and stores them in a *seed* repository of size N_{rep}, taking precaution to avoid duplicates of the gathered solutions. The seeding procedure can happen only after the first worker has been declared

Algorithm 1 Stopping Criteria.

$\epsilon_m \leftarrow$ Worker cost ▷ 1. The worker sends the cost of its best solution
ϵ_{his}.append(ϵ_m) ▷ 2. Store cost history per worker
while $\left(\frac{\epsilon_m}{\epsilon_{m-N_{\text{stall}}}} < 1 - tolerance\right)$ **do** ▷ Verification of stalled worker by Equation 4
    remove(ϵ_{his}.first) ▷ 3. Remove the first cost received
if $\bar{\epsilon}$ **is set then**
    $N_{\text{stall}} \leftarrow \bar{N}_{\text{stall}} \left(\frac{\bar{\epsilon}}{\epsilon_m}\right)^p$ ▷ 4. Compute a new number of stalled messages allowed.
if Size(ϵ_{his}) $> N_{\text{stall}}$ **then** ▷ 5. Verify if a worker is stalled
    StallWorker $\leftarrow$ **True** ▷ Worker is declared stalled
    Optim.StallCounter += 1 ▷ Stall counter per each optimisation algorithm
    if (All) Optim.StallCounter $> N_{\bar{\epsilon}}$ **then**
        OptimFlag $\leftarrow$ **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

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 uniform 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 can be seen in Algorithm 2.

4. Illustrative Example: Path Finding Problem

4.1. Problem Definition

To test the efficiency of the proposed strategy, a model problem is defined as follows. 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 Figure 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 x-direction. Hence, the set of design variables reduces to an N_p-dimensional array $y = y_1, y_2, ..., y_{N_p}$ 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

$$cost = l(y) + k\, p(y) \tag{7}$$

where, in the remainder of this work, the penalty factor is set to $k = 1$. The length of the path is computed from

$$l(y) = \sum_{i=1}^{N_p - 1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{8}$$

Algorithm 2 Seeding Procedure.

Pop $\leftarrow$ RU(PopSize) $\triangleright$ Initialise population using a random distribution RU
if RepExists **then**
  if RandNum $< \nu$ **then** $\triangleright$ Verify probability ν of seeding a population
    MaxSeeds = $\phi \times$ Pop $\triangleright$ Maximum number of seeds constrained by percentage ϕ
    SeedsFromRep $\leftarrow$ random(0, MaxSeeds) $\triangleright$ Number of seeds is a random number
    for pi $\leftarrow$ 1 to size(SeedsFromRep) **do**:
      RandSeed $\leftarrow$ random(0, RepSize)
      RandPop $\leftarrow$ random(0, PopSize)
      Pop[RandPop] $\leftarrow$ Repository[RandSeed] $\triangleright$ A random particle from the population is replaced by a random seed from the repository

while the penetration can be evaluated from

$$p(y) = \sum_{i=1}^{N_p-1} \sum_{j=1}^{N_c} \max \left(0, R_j - \sqrt{(X_j - x_i)^2 + (Y_j - y_i)^2}\right) \tag{9}$$

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 Figure 3. Recall that the coordinates x_i are known from the equal spacing of the points in x-direction.

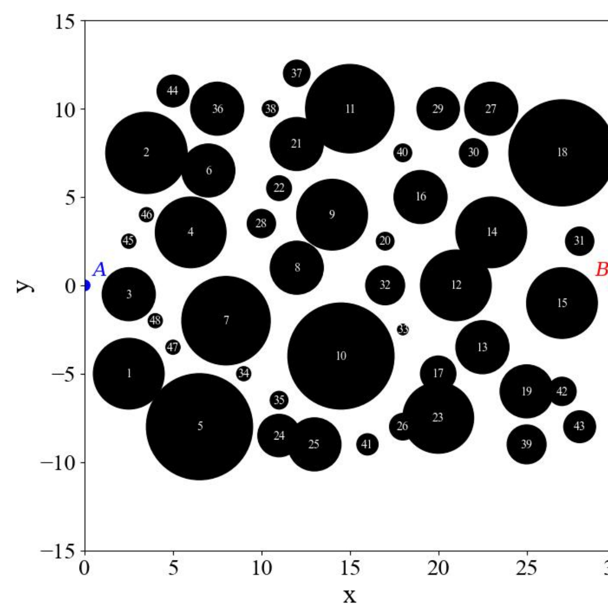

Figure 2: Path finding problem domain and obstacles imposed.

Considering the large number of obstacles shown in Figure 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.

4.2. 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 and MCS, as introduced in Section 2. The recommended parameters, detailed in Appendix B, have been used to set up each optimiser, i.e. without parameter experimentation phase done a priori. In addition, an *explorer* and *exploiter* version 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 hours of computation. The convergence behaviour of the proposed methodology has been presented in Figure 4 where all the individual convergence plots are 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, which are allowed to continue if they are able to sufficiently decrease their cost, 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 Figure 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

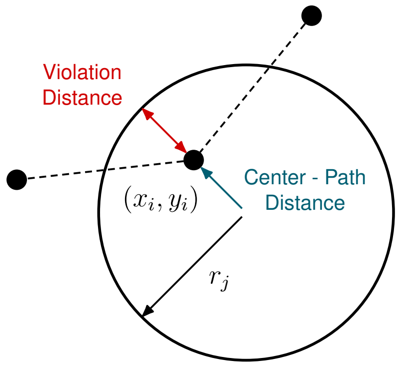

Figure 3: Definition of the obstacle penetration.

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
MCS	32.8160	66.8578	108.5957	14.7948	65.4331
SPO	31.4619	34.3488	41.0430	4.3824	31.4748

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

start having their entire population seeded. As expected, it is less likely that one algorithm remains as 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. Figure 4 b) shows the convergence behaviour over time exhibiting that optimisers with exploitation capabilities take over and refine the solution after the first 2.5 hours of exploration. To maintain diversification, new explorers are continuously initialised in the remaining time. The exploiter PSO is the most effective optimiser in the corresponding refinement region shown in Figure 4 c), while other optimisers with insufficient improvement are stopped. Figure 4 d) illustrates the seeding process, as different optimisers take over the best solution. In this specific problem, a GA optimiser seeds a MCS while in turn seeds a PSO that refines the solution. The latter two optimisers, MCS and PSO, share the best solution in the remaining time demonstrating they are the most suitable algorithms in SPO for the refinement process.

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 best solution achieved, the mean and standard deviation are presented in Figure 5, in which the vertical axis is the objective function while the horizontal is the computation time, with a maximum of 15 hours 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. Figure 6 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 clearly 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 PSO, provides straight segments in between the obstacles, proving to be a balanced approach between exploration and exploitation. Although the closest competitor is MCS, its best solution is crossing through an obstacle, suggesting that this optimiser has not converged in the imposed time constraint, but, it could refine the solution if continue 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.

5. 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 hyper parameters, which is otherwise generally crucial. The strategy has been applied to 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 procedure 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. A notable application, where the proposed supervised parallel optimisation strategy has recently shown promising results, is the training of recurrent neural networks, see [36].

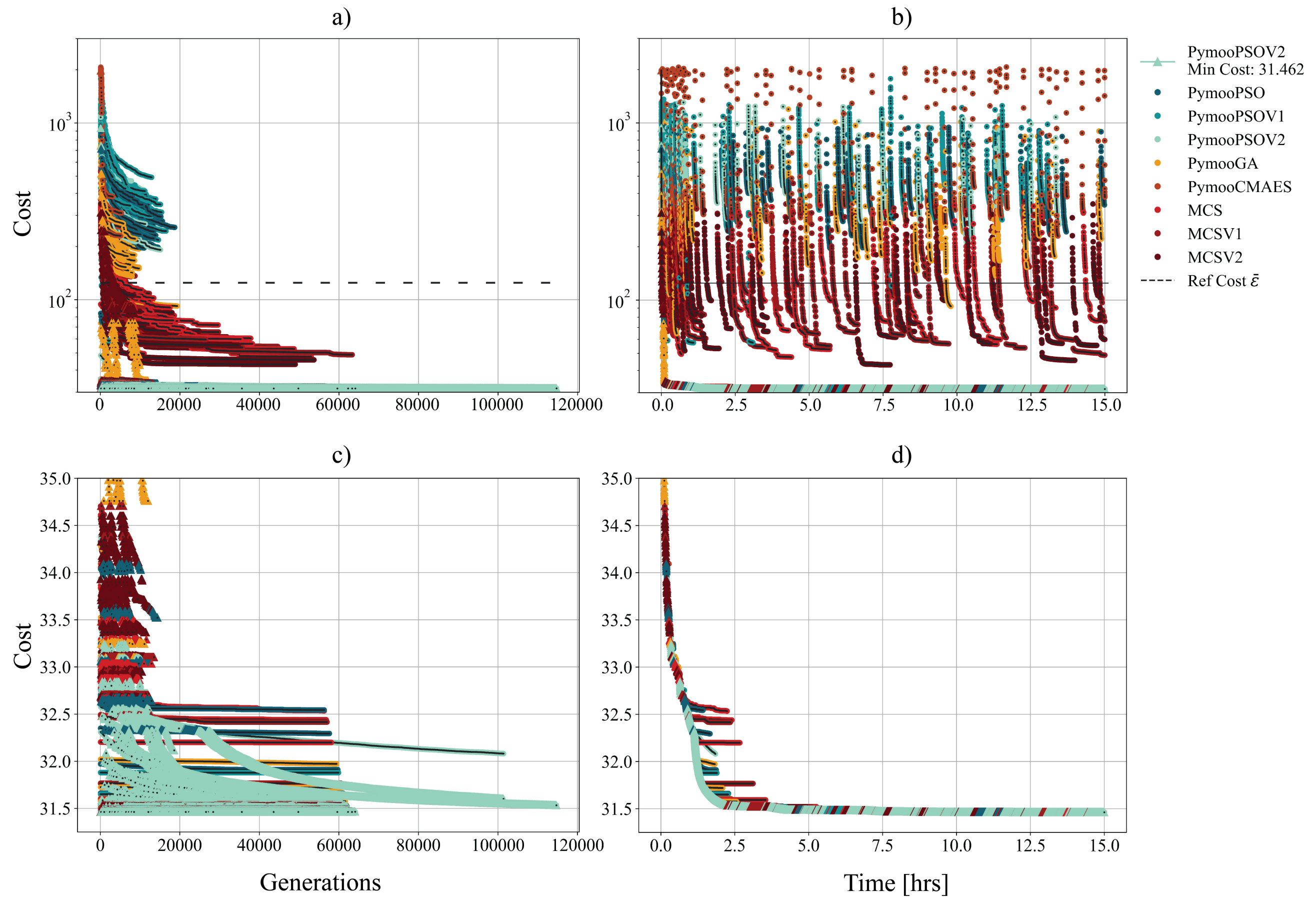

Figure 4: a) Convergence plot of the proposed strategy, b) convergence behaviour over time, c) refinement region extracted from convergence plot a), and d) refinement region of convergence over time extracted from b).

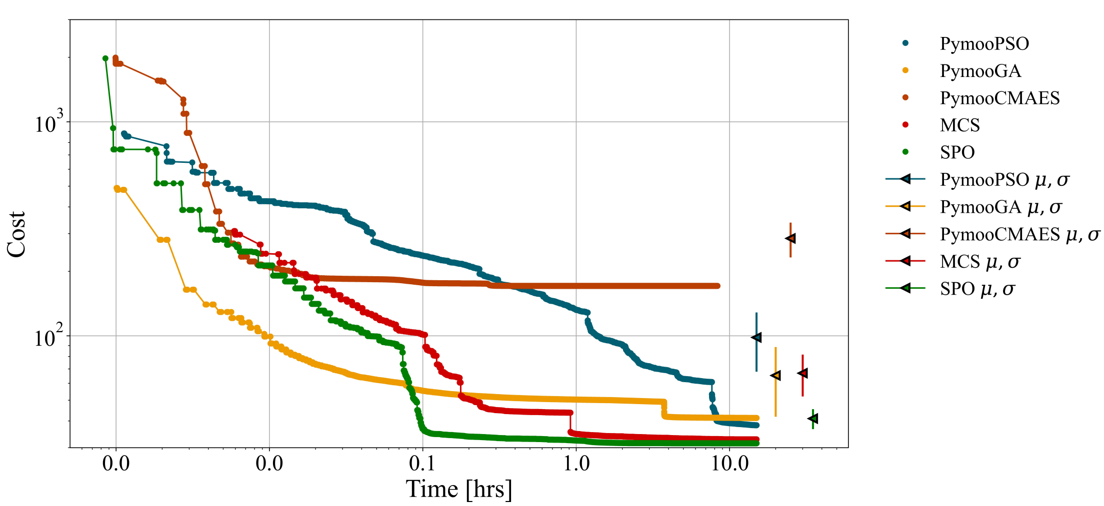

Figure 5: Convergence comparison of the best solution obtained by stand-alone optimisers and the proposed approach. The mean μ and standard deviation σ of the solutions throughout the 10 experiments is computed using the last result obtained.

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].

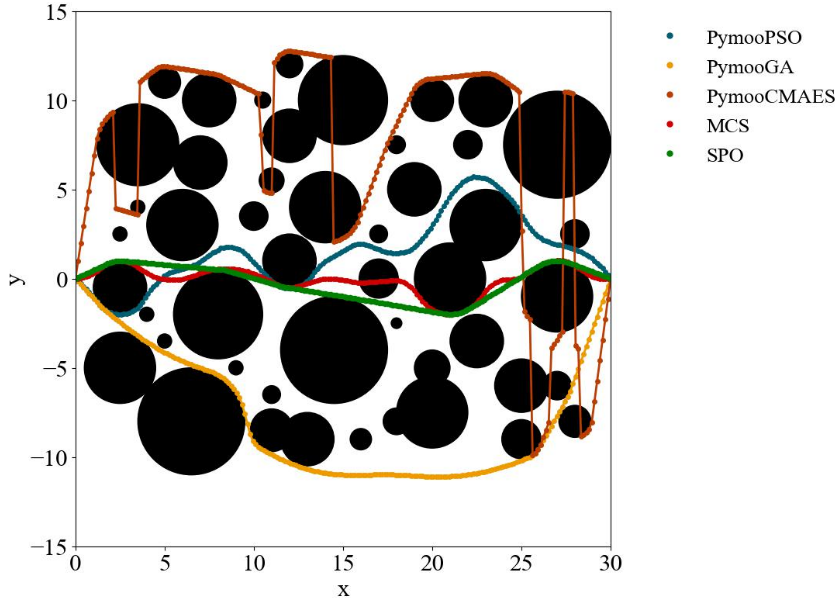

Figure 6: Solution comparison of stand-alone optimisers against the proposed strategy

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

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

Table 2
Circular obstacles location and radii defined within the domain of the problem.

A. 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 2 summarises the parameters to define the obstacles.

B. Optimiser Hyperparameters

Suggested SPO and individual hyperparameters utilised in the solution of the path finding problem of Section 4. Note that for the explorer and exploiter version 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 offsprings: 50
- **Pymoo CMA-ES**
  - Population size: 100
  - Initial standard deviation σ: 0.5
- **Pymoo PSO**
  - Population size: 25
  - 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: 25
  - 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: 25
  - 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]

References

[1] Holland, J.H.. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. MIT Press; 1992.

[2] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.. Optimization by Simulated Annealing. Science 1983;220(4598):671–680. URL: https://www.science.org/doi/10.1126/science.220.4598.671. doi:10.1126/science.220.4598.671.

[3] Kennedy, J., Eberhart, R.. Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks; vol. 4. 1995, p. 1942–1948. doi:10.1109/ICNN.1995.488968.

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

[5] Storn, R., Price, K.. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Journal of Global Optimization 1997;11(4):341–359. URL: https://doi.org/10.1023/A:1008202821328. doi:10.1023/A:1008202821328.

[6] Yang, X., Deb, S.. Cuckoo search via Lévy flights. In: 2009 World Congress on Nature Biologically Inspired Computing (NaBIC). 2009, p. 210–214. doi:10.1109/NABIC.2009.5393690.

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

[8] Vanaret, C., Gotteland, J.B., Durand, N., Alliot, J.M.. Preventing premature convergence and proving the optimality in evolutionary algorithms. In: Legrand, P., Corsini, M.M., Hao, J.K., Monmarché, N., Lutton, E., Schoenauer, M., editors. Artificial Evolution. Springer International Publishing; 2014, p. 29–40.

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

[10] Yang, B., Chen, Y., Zhao, Z.. 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, p. 166–170. doi:10.1109/ICCA.2007.4376340.

[11] Ghamisi, P., Benediktsson, J.A.. Feature Selection Based on Hybridization of Genetic Algorithm and Particle Swarm Optimization. IEEE Geoscience and Remote Sensing Letters 2015;12(2):309–313. doi:10.1109/LGRS.2014.2337320.

[12] Zhao, F., Zhang, Q., Yu, D., Chen, X., Yang, Y.. A Hybrid Algorithm Based on PSO and Simulated Annealing and Its Applications for Partner Selection in Virtual Enterprise. In: Huang, D.S., Zhang, X.P., Huang, G.B., editors. Advances in Intelligent Computing. Lecture Notes in Computer Science; Berlin, Heidelberg: Springer; 2005, p. 380–389. doi:10.1007/11538059_40.

[13] Sadati, N., Amraee, T., Ranjbar, A.M.. A global Particle Swarm-Based-Simulated Annealing Optimization technique for under-voltage load shedding problem. Applied Soft Computing 2009;9(2):652–657. URL: https://www.sciencedirect.com/science/article/pii/S1568494608001269. doi:10.1016/j.asoc.2008.09.005.

[14] Ghodrati, A., Lotfi, S.. A hybrid cs/pso algorithm for global optimization. In: Pan, J.S., Chen, S.M., Nguyen, N.T., editors. Intelligent Information and Database Systems. Lecture Notes in Computer Science; Berlin, Heidelberg: Springer; 2012, p. 89–98. doi:10.1007/978-3-642-28493-9_11.

[15] Chi, R., Su, Y.x., Zhang, D.h., Chi, X.x., Zhang, H.j.. A hybridization of cuckoo search and particle swarm optimization for solving optimization problems. Neural Computing and Applications 2019;31(1):653–670. URL: https://doi.org/10.1007/s00521-017-3012-x. doi:10.1007/s00521-017-3012-x.
[16] Li, X., Yin, M.. A particle swarm inspired cuckoo search algorithm for real parameter optimization. Soft Computing 2016;20(4):1389–1413. URL: https://doi.org/10.1007/s00500-015-1594-8. doi:10.1007/s00500-015-1594-8.
[17] Dash, J., Dam, B., Swain, R.. Optimal design of linear phase multi-band stop filters using improved cuckoo search particle swarm optimization. Applied Soft Computing 2017;52:435–445. URL: https://www.sciencedirect.com/science/article/pii/S1568494616305373. doi:10.1016/j.asoc.2016.10.024.
[18] Hendtlass, T.. 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. IEA/AIE '01; Berlin, Heidelberg: Springer-Verlag; 2001, p. 11–18.
[19] Zhang, W.J., Xie, X.F.. 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, p. 3816–3821 vol.4. doi:10.1109/ICSMC.2003.1244483.
[20] Xu, R., Xu, J., Wunsch, D.C.. Clustering with differential evolution particle swarm optimization. In: IEEE Congress on Evolutionary Computation. 2010, p. 1–8. doi:10.1109/CEC.2010.5586257.
[21] Robinson, J., Sinton, S., Rahmat-Samii, Y.. Particle swarm, genetic algorithm, and their hybrids: optimization of a profiled corrugated horn antenna. In: IEEE Antennas and Propagation Society International Symposium (IEEE Cat. No.02CH37313); vol. 1. 2002, p. 314–317 vol.1. doi:10.1109/APS.2002.1016311.
[22] Zhang, J., Pan, T.S., Pan, J.S.. A parallel hybrid evolutionary particle filter for nonlinear state estimation. In: 2011 First International Conference on Robot, Vision and Signal Processing. 2011, p. 308–312. doi:10.1109/RVSP.2011.77.
[23] Nik, A.A., Nejad, F.M., Zakeri, H.. Hybrid PSO and GA approach for optimizing surveyed asphalt pavement inspection units in massive network. Automation in Construction 2016;71:325–345. URL: https://www.sciencedirect.com/science/article/pii/S0926580516301571. doi:10.1016/j.autcon.2016.08.004.
[24] Garg, H.. A hybrid PSO-GA algorithm for constrained optimization problems. Applied Mathematics and Computation 2016;274:292–305. URL: https://www.sciencedirect.com/science/article/pii/S0096300315014630. doi:10.1016/j.amc.2015.11.001.
[25] Wansasueb, K., Bureerat, S., Kumar, S.. Ensemble of four metaheuristic using a weighted sum technique for aircraft wing design. Engineering and Applied Science Research 2021;48(4):385–396. URL: https://ph01.tci-thaijo.org/index.php/easr/article/view/242706.
[26] Singh, P., Kottath, R.. An ensemble approach to meta-heuristic algorithms: Comparative analysis and its applications. Computers & Industrial Engineering 2021;162:107739. URL: https://www.sciencedirect.com/science/article/pii/S0360835221006434. doi:10.1016/j.cie.2021.107739.
[27] Engelbrecht, A.P.. Heterogeneous particle swarm optimization. In: Swarm Intelligence; vol. 6234. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010, p. 191–202. URL: http://link.springer.com/10.1007/978-3-642-15461-4_17. doi:10.1007/978-3-642-15461-4_17.
[28] Lynn, N., Suganthan, P.N.. Ensemble particle swarm optimizer. Applied Soft Computing 2017;55:533–548. URL: https://www.sciencedirect.com/science/article/pii/S1568494617300753. doi:10.1016/j.asoc.2017.02.007.
[29] Schutte, J.F., Reinbolt, J.A., Fregly, B.J., Haftka, R.T., George, A.D.. Parallel global optimization with the particle swarm algorithm. International Journal for Numerical Methods in Engineering 2004;61(13):2296–2315. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1149. doi:10.1002/nme.1149.
[30] Chang, J.F., Chu, S.C., Roddick, J., Pan, J.S.. A parallel particle swarm optimization algorithm with communication strategies. Journal of Information Science and Engineering 2005;21:809–818.
[31] Venter, G., Sobieszczanski-Sobieski, J.. Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. Journal of Aerospace Computing, Information, and Communication 2006;3(3):123–137. URL: https://arc.aiaa.org/doi/10.2514/1.17873. doi:10.2514/1.17873.
[32] Waintraub, M., Schirru, R., Pereira, C.M.N.A.. Multiprocessor modeling of parallel Particle Swarm Optimization applied to nuclear engineering problems. Progress in Nuclear Energy 2009;51(6):680–688. URL: https://www.sciencedirect.com/science/article/pii/S014919700900033X. doi:10.1016/j.pnucene.2009.02.004.
[33] Blank, J., Deb, K.. pymoo: Multi-objective optimization in python. IEEE Access 2020;8:89497–89509.
[34] Walton, S., Hassan, O., Morgan, K., Brown, M.. Modified cuckoo search: a new gradient free optimisation algorithm. Chaos, Solitions and Fractals 2011;44:710–718. doi:10.1016/j.chaos.2011.06.004.
[35] Jin, J., Yang, C., Zhang, Y.. An improved cma-es for solving large scale optimization problem. In: Tan, Y., Shi, Y., Tuba, M., editors. Advances in Swarm Intelligence. Springer International Publishing; 2020, p. 386–396.
[36] Dettmer, W.G., Muttio, E.J., Alhayki, R., Perić, D.. A framework for neural network based constitutive modelling of inelastic materials [unpublished] 2023;.