Array 16 (2022) 100249

Contents lists available at ScienceDirect

Array

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

Collaborative optimization by shared objective function data

I Gusti Agung Gede Angga [a], Mathias Bellout [a], Per Eirik Strand Bergmo [b], Per Arne Slotte [a], Carl Fredrik Berg [a,*]

[a] Department of Geoscience and Petroleum, Norwegian University of Science and Technology (NTNU), S. P. Andersens veg 15A, 7031, Trondheim, Norway
[b] Department of Petroleum, SINTEF Industry, S. P. Andersens veg 15B, 7031, Trondheim, Norway

ARTICLE INFO

Keywords:
Collaborative optimization algorithms
Multi-task optimization
Simulation-based optimization
Genetic algorithm
Particle swarm optimization
Gradient descent

ABSTRACT

This article presents a collaborative algorithmic framework that is effective for solving a multi-task optimization scenario where the evaluation of their objectives consists of two parts: The first part involves a common computationally heavy function, e.g., a numerical simulation, while the second part further evaluates the objective by performing additional, significantly less computationally-intensive calculations. The ideas behind the collaborative framework are (i) to solve all the optimization problems simultaneously and (ii) at each iteration, to perform a synchronous "collaborative" operation. This distinctive operation entails sharing the outcome of the heavy part between all search processes. The goal is to improve the performance of each individual process by taking advantage of the already-computed heavy part of solution candidates from other searches. Several problem sets are presented. With respect to solution quality, consistency, and convergence speed, we observe that our collaborative algorithms perform better than traditional optimization techniques. Information sharing is most actively exploited during early stages of optimization. Though the collaborative algorithms require additional computing time, the added cost is diminishing with increasing difference between the computational cost of the expensive and light parts.

1. Introduction

There are two common classes of optimization problem: single-objective optimization (SOO) and multi-objective optimization (MOO). SOO aims at determining the best solution for a single objective function only, while the goal of MOO is to find a set of non-dominated (Pareto-optimal) solutions for two or more, often contradicting, objectives. In recent years, a new class of optimization problem called multi-task optimization (MTO) has emerged and attracted attention in research society. The goal of MTO is to solve a set optimization problems, usually referred as "tasks", simultaneously so that the synergies among the tasks can be utilized (i.e., by means of inter-task knowledge transfer) to improve the search process of each task [1,2]. An MTO is made up of N_p different optimization problems, where each problem has a unique objective function f_i defined over search space Ω_i. Without loss of generality, we herein define that all problems are maximization problems. For minimization problems, we can turn them into maximization problems by using negative objective functions. The purpose of MTO is to determine a set of optimal solutions $\{\vec{x}_1^*, \vec{x}_2^*, \ldots, \vec{x}_{N_p}^*\}$ where $\vec{x}_i^* = \text{argmax}_{\vec{x}_i \in \Omega_i} f_i(\vec{x}_i)$. So the end product of MTO is different from MOO. Distinctions between MTO and MOO are further discussed in [1,3,4]. In spite of the differences, Pareto-optimal solutions for an MOO could be obtained by reformulating the MOO as a set of SOO problems (e.g., by methods proposed in [5,6]), and then solving for this set of problems as an MTO scenario.

A straightforward and traditional way to solve an MTO is by solving each problem in an independent manner. This approach however may have limitations as the optimization problems become more complex. In addition, real-world optimization problems usually have high degree of similarity (i.e., commonality in their fitness landscapes or optimal solutions) [1,7,8]. These two aspects have motivated the scientific community to come up with MTO algorithms which aim to solve all the optimization problems simultaneously in a single search process so that the knowledge obtained from the optimization of one problem can be exploited to help addressing other problems. MTO algorithms are usually developed using concepts drawn from evolutionary computation (e.g., [3,7,9]), swarm intelligence (e.g., [10–12]), or Bayesian optimization (e.g., [13]). Some of these MTO algorithms will be briefly explained in the following paragraphs.

Evolutionary MTO (EMTO) algorithms, which adopt search procedures and operators of evolutionary computation, can be classified

* Corresponding author.
E-mail addresses: i.g.a.g.angga@ntnu.no (I G.A.G. Angga), mathias.bellout@gmail.com (M. Bellout), per.bergmo@sintef.no (P.E.S. Bergmo), paslotte@gmail.com (P.A. Slotte), carl.f.berg@ntnu.no (C.F. Berg).

https://doi.org/10.1016/j.array.2022.100249
Received 12 July 2022; Received in revised form 28 August 2022; Accepted 12 September 2022
Available online 17 September 2022
2590-0056/© 2022 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Nomenclature

Acronyms

CO_2	Carbon dioxide
C-GA	Collaborative genetic algorithm
C-GD	Collaborative gradient descent
C-GD-I	Collaborative gradient descent version I
C-GD-II	Collaborative gradient descent version II
C-PSO	Collaborative particle swarm optimization
CCEA	Cooperative co-evolutionary algorithm
DEMTO	Differential evolutionary multi-task optimization
EMTO	Evolutionary multi-task optimization
GA & NC-GA	Traditional non-collaborative genetic algorithm
GD & NC-GD	Traditional non-collaborative gradient descent
MFEA	Multi-factorial evolutionary algorithm
MM	Multi-population based multitasking
MOO	Multi-objective optimization
MPEF	Multi-population evolutionary framework
MTO	Multi-task optimization
NSGA-II	Non-dominated sorting genetic algorithm version II
OFV	Objective function value
P10	10th percentile
P50	50th percentile or median
P90	90th percentile
PSO & NC-PSO	Traditional non-collaborative particle swarm optimization
SM	Single-population based multitasking
SOO	Single-objective optimization
USD	United States Dollar

Symbols

$\circ$	Function composition
$\mathbb{R}$	Set of real numbers
Ω_i	Search space for problem i
Φ_i	Population assigned for tackling problem i
Ψ_i	Collection of promising solution candidates from populations Φ_q, $q \neq i$, to be cloned and introduced into Φ_i
$\vec{\beta}_i$	Global best position for problem i
$\vec{m}_i$	Set of coefficients in the objective function of problem i
$\vec{s}_i$	Set of constants in the objective function of problem i
$\vec{x}_i$	Solution candidate for problem i
$\vec{x}_i^*$	Optimal solution for problem i
$\vec{y}_i$	Outputs of a computationally heavy function, f_h, with respect to $\vec{x}_i$
A	Coefficient in the objective function
$C(\vec{y})$	Operating cost of hydrocarbon production with respect to simulation results $\vec{y}$
f_h	Computationally heavy function
f_i	Objective function for problem i
$f_{l,i}$	Computationally light function for problem i
$H(k)$	Aggregated objective function value with respect to swap scenario k
$M_{CO_2}(\vec{y})$	Quantity of carbon emissions from hydrocarbon production with respect to simulation results $\vec{y}$
N_d	Dimension of optimization problems
$N_{f,c}$	Number of objective function evaluations in collaborative algorithms
$N_{f,nc}$	Number of objective function evaluations in traditional non-collaborative algorithms
N_m	Size of each population
N_p	Number of optimization problems
N_r	Number of swap scenarios
N_t	Number of iterations
$R(\vec{y})$	Revenue from hydrocarbon production with respect to simulation results $\vec{y}$
r_{CO_2}	CO_2 tax rate
T_c	Run time of collaborative algorithms
T_h	Computation time of heavy function, f_h
T_l	Computation time of light functions, $f_{l,i}$
T_{nc}	Run time of traditional non-collaborative algorithms
z_i	Output of a computationally light function for problem i, $f_{l,i}$

into two main families of algorithm; one of them is single-population based multitasking (SM). One early example of SM is multi-factorial evolutionary algorithm (MFEA) proposed by Gupta et al. [3]. The development of MFEA was inspired by the concept of multifactorial inheritance, which explains that the traits among offspring are influenced by "many factors", such as genetic and cultural factors [14,15]. The search process in MFEA is executed using one population that contains solutions for all problems. Each individual in the population has a skill factor indicating the one task on which the individual is assigned to, and two randomly chosen individuals in the population can freely execute a crossover operation if they have the same skill factors. However, if the skill factors are different, the two individuals can undergo crossover only by a given probability.

Another family of EMTO algorithms is multi-population based multitasking (MM). In contrast with SM, MM algorithms employ several populations where each population contains all solution candidates for an optimization problem in the MTO setting. The advantages of MM are twofold: (i) MM can easily adopt any existing population-based optimization algorithms, and (ii) the transfer of information among the populations can be performed effectively [16]. One example of MM algorithms is multi-population evolutionary framework (MPEF) initially proposed by Li et al. [16], and later extended in [17,18]. In MPEF each population has its own probability for exploiting knowledge of other populations, and this probability is adaptively tuned to prevent a phenomenon called negative transfer. This phenomenon might occur in SM due to the crossover of individuals for non-related tasks and eventually could lead to performance degradation. Another example of MM algorithms is differential evolutionary multi-task optimization (DEMTO) by Zheng et al. [7]. There is also an MM algorithm developed based on the island model by Hashimoto et al. [19]. Both DEMTO and the island model implement explicit knowledge transfer, meaning that individuals (complete solutions) are migrated from one population or task to another. One commonality between DEMTO and the island model is that individuals being migrated into a population are randomly chosen from other populations. This can potentially lead to negative transfer. In contrast, our collaborative algorithms have

different procedures of transferring knowledge among the tasks, i.e., by only exploiting or migrating individuals (from other populations) that are most useful for the particular population or task.

Besides evolutionary computation, the idea of transferring knowledge gathered while solving an optimization problem to other problems has also been incorporated with the Bayesian optimization framework by Swersky et al. in multitask Bayesian optimization [13]. An important component of such algorithm is multitask Gaussian Process models (e.g., [20]) which are employed to progressively capture the degree of correlation between different tasks in an MTO. Using this correlation, observations on the other tasks will act as extra observations for the task of interest without any additional function evaluations [13]. If the tasks are related, these extra observations could help the surrogate models (i.e., the Gaussian Process models) to better represent the objective functions implemented in those tasks, meaning that the models become more accurate (closer to the true objective functions) and more precise (having less uncertainties). Response surface of the acquisition function, which is needed for deciding the next sampling point, is then affected by the improvements on the surrogate models [21]. As a result, we might obtain a better sampling point or solution candidate to evaluate. On the other hand, if the tasks are unrelated, the extra observations will not be considered and therefore will not ruin the optimization performance [20].

Herein we present collaborative algorithms that are effective for solving a special MTO case. Like normal MTO, the special MTO case consists of several optimization problems or tasks, where each task employs a unique objective function. The uniqueness of the special MTO case is that all the objective functions must have the following characteristics:

- The evaluation of each objective consists of two stages, $f_i = f_{l,i} \circ f_h$, where $\circ$ indicates a function composition of the heavy f_h and light $f_{l,i}$ functions.
- In the first stage, the evaluation process involves a computationally heavy function, f_h, such as solving a set of non-linear partial differential equations in a simulation.
  Importantly, all the optimization problems have an identical heavy function f_h, e.g., a simulation is performed using the exact same model giving the same type of output.
- In the second stage, a subsequent evaluation of the objective is based on the simulation results, i.e., the outputs of the heavy function f_h, and is a comparatively light in terms of the computation. This light function $f_{l,i}$ is problem-specific, meaning that each optimization problem has its own distinctive light function.

Lastly, all the tasks optimize the same variable types and have the same search space.

One example of such special MTO case in engineering is for optimizing airfoil design. The geometry of an airfoil influences lift and drag forces exerted on the airfoil. Optimization problems include identifying airfoil shapes that either maximize lift force, minimize drag force, or maximize lift-to-drag ratio (e.g., [22–24]). Engineers however may want to find and compare all optimal airfoil designs corresponding to these different objective functions. This means that they need to solve several optimization problems. Again the main interest here is to find different optimal solutions for different objective functions, and not to identify the trade-off between the objective functions as in MOO. Note that, to evaluate these problem objectives, we need to perform computational fluid dynamics simulations (i.e., the heavy function f_h) to determine spatial distributions of fluid pressure and velocity around the airfoil. Results from these simulations are then used to compute the objective function value using the corresponding light function $f_{l,i}$. Clearly, for all these problems, the simulation effort is the computationally demanding part compared to the subsequent computation of the objective function value. The two stages of objective function evaluation here reflect the uniqueness of the special MTO case.

Another example of the special MTO case appears in the development phase of a hydrocarbon field. Some variables to optimize include well placement, well completion, and/or well production strategy [25]. The majority of optimization studies in the literature discuss optimization problems that either maximize profit, maximize field production, or minimize energy consumption [26–28]. Like before, one may want to identify and compare all optimal development scenarios for different objective functions, meaning that a set of optimization problems must be solved. There are also interests to investigate how the variations of market prices, e.g., oil and steel prices, alter the optimal development scenario. For such sensitivity analysis, we must solve multiple optimization problems, with each problem using a different market price. A commonality of such sets of optimization problems is that some of the parameters, e.g., market prices or particular choice of objective function, do not enter into the simulation (i.e., the heavy function f_h) and require only light calculations using the simulation outputs (i.e., the light function $f_{l,i}$). For example, the final net present value calculation consists of a single function evaluation. On the other hand, the full simulation, e.g., for fluid displacement in porous media, relies on computationally costly solutions to a set of non-linear partial differential equations.

The easiest approach to deal with the aforementioned special MTO cases is by solving each optimization problem independently using traditional optimization algorithms, such as gradient descent (GD) [29], pattern search [30], genetic algorithm (GA) [31], particle swarm optimization (PSO) [32,33], or Bayesian optimization [34,35]. Better optimal solutions can obtained by employing MTO algorithms discussed earlier which have the capabilities to transfer and exploit the knowledge gathered while solving the optimization problems simultaneously. However, the MTO algorithms are sub-optimal for tackling our special MTO cases because the results of heavy function f_h will be used only once. For example in EMTO algorithms, an offspring will be evaluated only for one problem (depending on the skill factor or population of the offspring). Similarly in multitask Bayesian optimization, the next data point (that is obtained from the maximization of acquisition function) will be observed only for the relevant problem. This practice is inefficient as the results of heavy function f_h can actually be utilized to compute other light functions $f_{l,i}$ and those computed objective values are potentially helpful for the overall optimization processes. For instance in multitask Bayesian optimization, additional observed data points (coming from the reuse of heavy function outputs) can completely eliminate uncertainties, instead of just reducing them.

Further, MTO algorithms rely heavily on the similarity of optimization problems being considered [36–38]. They work by assuming that the optimization problems have commonality in their fitness landscapes or optimal solutions which can be utilized to facilitate the overall search processes [7]. However, if some of the optimization tasks are unrelated and the transfer of knowledge occurs between these unrelated tasks (referred as negative transfer in [1]), MTO algorithms could experience performance downturns [38].

With the above rationale, we herein present collaborative optimization algorithms that are effective for solving the special MTO cases, consisting of a common heavy and an individual light part in their objective function calculation. In our proposed collaborative algorithms, all defined problems are solved simultaneously and communicating, instead of being solved independently without inter-communication. Additionally, by considering the advantages of MM mentioned earlier, the proposed collaborative algorithms adopt the multi-population based framework and implement the explicit knowledge transfer strategy. The communication is materialized through a "collaborative" operation in every iteration, thus a differentiating feature of the proposed algorithms. The "collaborative" operation implies that results of heavy function f_h are distributed to all problems. By evaluating a corresponding light function $f_{l,i}$ on the shared result data, each optimization problem is able to evaluate additional solution candidates. Therefore, the presented collaborative algorithms have the potential to (i) produce

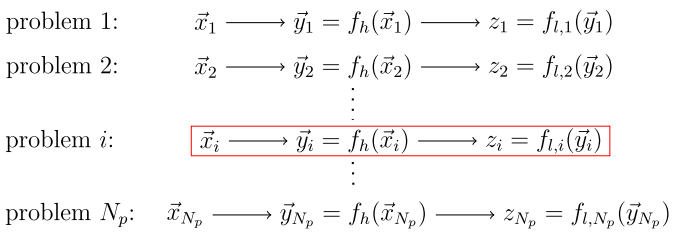


Fig. 1. Calculation for objective function value, z_i, in a set of N_p optimization problems, without any collaboration between the different problems. $\vec{x}_i$: a solution candidate for problem i to evaluate. $\vec{y}_i$: outputs of a computationally heavy function, f_h, with respect to $\vec{x}_i$. z_i: output of a computationally light function for problem i, $f_{l,i}$.

better solutions for their corresponding optimization problems and (ii) have faster convergences. The additional light function calculations add to the computing time, but when the problems' heavy function is computationally much more demanding than the light functions, this additional cost becomes insignificant. Based upon our literature review, we cannot identify existing algorithms that take advantage of the heavy/light-function structural characteristics in the objective function evaluation of the aforementioned special MTO cases. Also note that our proposed collaborative algorithms are not multi-fidelity optimization; the collaborative algorithms exploit the two levels of objective function calculation, i.e., the heavy and light parts, and do not use or create any low-fidelity models. These problem characteristics and the features of the collaborative algorithms will be discussed deeper in this article.

This article is structured as follows. Section 2 defines the optimization problems at which the collaborative algorithms are developed for. Section 3 describes two pivotal and distinctive features in the collaborative algorithms and presents the adoption of these features in several traditional optimization techniques. Section 4 provides descriptions of four problem sets that we use for testing out the collaborative algorithms. Optimization results, including performance comparison against the traditional optimization methods and the MFEA, are presented and discussed in Section 5. To avoid confusion with the naming of some existing algorithms, Section 6 further clarifies the use of the term "collaborative". Lastly, concluding remarks and future work are given in Sections 7 and 8, respectively.

2. Characteristics of optimization problems

Our collaborative algorithms target resolving a special MTO case with attributes as described in Section 1. In this MTO case, we have a collection of N_p optimization problems having different objective functions. Every problem aims at optimizing a set of variables, $\vec{x}$, with respect to its objective. A crucial characteristic is that the search space for $\vec{x}$ is identical for all the problems under consideration.

Another important problem characteristic is that the objective evaluation process for each problem i consists of two steps; $f_i = f_{l,i} \circ f_h$. The individual evaluation of each objective function is illustrated in Fig. 1. In this figure, the red box indicates how the objective function value for the ith problem, $z_i = f_{l,i}(\vec{y}_i) = f_{l,i}(f_h(\vec{x}_i))$, is obtained for a solution candidate $\vec{x}_i$. The first step of each objective evaluation process involves resolving a computationally heavy function, f_h, e.g., the numerical solutions of a real system. Note that the heavy function in this first step, f_h, is exactly the same for all the optimization problems i, and thus that the simulation outputs, $\vec{y}_i = f_h(\vec{x}_i)$, contain the same components. The second step involves resolving a computationally light function, $f_{l,i}$, with input the simulation results, $\vec{y}_i$, computed in the first step. A main distinction is that, unlike the heavy function, the light function is different for each optimization problem i. That is, $f_{l,i}$ denotes a light function that uniquely defines problem i.

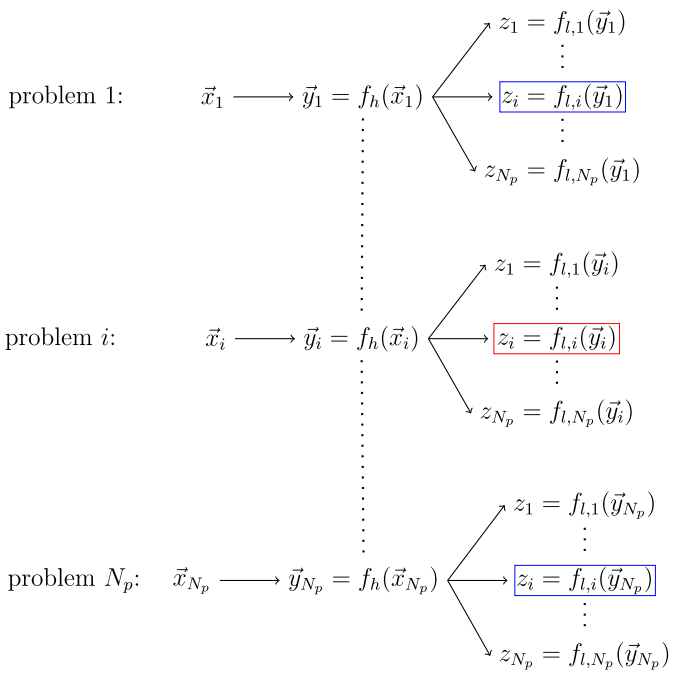


Fig. 2. Calculation for objective function value, z_i, in our collaborative optimization algorithms.

3. Collaborative optimization algorithms

3.1. Main concepts

In many traditional optimization techniques, all solution candidates that have been evaluated in the current or previous iterations provide information for generating new solution candidates to test. This information has pivotal influence on how the optimization routine progresses. For GA, individuals in the current population will pass on their genes to their offsprings, with the inherited genes determining the fitness of the new population. For PSO, location of the global best particle (which is updated from the current population) will affect the velocities (i.e., magnitudes plus directions) of the other population members. For GD, search direction or gradient is determined at the current solution.

The collaborative algorithms presented in this article improve the search process by enhancing the information available for the development of the next solution candidates. The algorithms achieve this goal using two distinctive features. First, all optimization problems specified in Section 2 are solved concurrently. Here, the term "concurrent" implies that the search process for each problem (generation and evaluation of new solution candidates) advances for every iteration of the algorithms. This differs from a straightforward (sequential) approach where each optimization problem is solved completely before moving onto the next one.

Second, a collaborative operation is carried out in every single iteration, where all problems share the outputs of the heavy function evaluations, $\vec{y}_i$. When using traditional approaches (see Fig. 1), $\vec{y}_i$ is used only for quantifying the light function of problem i, $z_i = f_{l,i}(\vec{y}_i)$. The collaborative operation extends the utilization of $\vec{y}_i$, where it is also used to evaluate the light functions associated with the other problems, i.e., $z_1 = f_{l,1}(\vec{y}_i); \ldots; z_{i-1} = f_{l,i-1}(\vec{y}_i); z_{i+1} = f_{l,i+1}(\vec{y}_i); \ldots; z_{N_p} = f_{l,N_p}(\vec{y}_i)$ (see Fig. 2). A clear consequence is that any single optimization problem is able to evaluate many additional solution candidates (in every iteration), thus enhancing the information available for deciding the next solution candidates. For example, the next solution candidates of problem i are traditionally determined based upon $z_i = f_{l,i}(\vec{y}_i)$ alone

(see the red box in Fig. 2). This contrasts with what occurs in the collaborative algorithms where we decide the next solution candidates not only from $z_i = f_{l,i}(\vec{y}_i)$ but also from the current solution candidates of the other problems, i.e., $z_i = f_{l,i}(\vec{y}_1); \ldots; z_i = f_{l,i}(\vec{y}_{i-1}); z_i = f_{l,i}(\vec{y}_{i+1}); z_i = f_{l,i}(\vec{y}_{N_p})$ (see the blue boxes in Fig. 2).

In population-based optimization methods like GA or PSO, the additional information may steer the overall search towards more promising territories. In local-search methods like GD or PS, the additional information may prevent us from getting stuck in local optima. Enhancement on the information thus enables: (i) the improvement of final solution for any optimization problem, and (ii) the faster convergence. These two aspects are investigated in this article. In the next three subsections, we present the implementations of the collaborative notion on three traditional optimization techniques, i.e., GA, PSO, and GD. Note, however, that the collaborative approach depends solely on the computational characteristics of the problems, and can thus be implemented on other search methods.

3.2. Collaborative genetic algorithm (C-GA)

Algorithm 1: Collaborative genetic algorithm (C-GA)

```
1  t ← 0 // Iteration counter.
   /* Initialize all populations and their members. */
2  for i ← 1 to N_p do
3      Φ_i ← {x⃗_i^j | j = 1, 2, ..., N_m}
   /* Perform evaluations of a computationally heavy function, f_h. */
4  for i ← 1 to N_p do
5      for j ← 1 to N_m do
6          y⃗_i^j ← f_h(x⃗_i^j)
7  while termination criteria is not met do
8      t ← t + 1 // Iteration counter.
9      /* Perform a collaborative operation. */
10     for i ← 1 to N_p do
11         Ψ_i ← determine_candidates_to_clone(i, N_c)
12     for i ← 1 to N_p do
13         Φ_i ← update_population_members(Φ_i, Ψ_i, i, N_m)
14     /* Perform basic GA operations for each population. */
15     for i ← 1 to N_p do
16         Φ_i ← run_selection_operation(Φ_i, i) // This operation employs the
               computationally light function embedded in problem i, f_{l,i}, for
               sorting population members.
17         Θ_i ← run_crossover_operation(Φ_i) // Where Θ_i = {θ⃗_i^j | j = 1, 2, ..., N_m}.
18         Θ_i ← run_mutation_operation(Θ_i)
19         for j ← 1 to N_m do
20             η⃗_i^j ← f_h(θ⃗_i^j)
21         Φ_i ← run_replacement_operation(Φ_i, Θ_i, i) // This operation also employs
               f_{l,i} for comparing offsprings and their parents.
22     evaluate_termination_criteria()
```

Φ_i: a population assigned for tackling problem i.
N_m: number of solution candidates in Φ_i (size of each population).
$\vec{x}_i^j$: chromosome of individual j that belongs to Φ_i (solution candidate j for problem i).
$\vec{y}_i^j$: outputs of a computationally heavy function, f_h, with respect to $\vec{x}_i^j$.
Ψ_i: a collection of promising solution candidates from populations Φ_q, $q \neq i$, to be cloned and introduced into Φ_i.
N_c: number of promising solution candidates in Ψ_i.
Θ_i: a set of offsprings generated from a crossover operation on the selected parents Φ_i.
$\vec{\theta}_i^j$: chromosome of offspring j that belongs to Θ_i (new solution candidate j for problem i).
$\vec{\eta}_i^j$: outputs of f_h with respect to $\vec{\theta}_i^j$.

Algorithm 2: Specific functions in C-GA

```
Function 1: Ψ_i ← determine_candidates_to_clone(i, N_c)
    Ψ_i ← ∅
    for q ← 1 to N_p do
        if q ≠ i then
            Copy all members of Φ_q, and append to Ψ_i.
    Sort members of Ψ_i based upon f_{l,i}.
    Keep the best N_c solution candidates among Ψ_i, and eliminate the remaining.
    return Ψ_i

Function 2: Φ_i ← update_population_members(Φ_i, Ψ_i, i, N_m)
    Clone and introduce members of Ψ_i into Φ_i. // So Φ_i now has N_m + N_c members.
    Sort members of Φ_i based upon f_{l,i}.
    Keep the best N_m solution candidates among Φ_i, and eliminate the remaining.
    return Φ_i
```

Genetic algorithm (GA) is a well known class of search methods with many implementations reported in the literature [39–41]. This algorithm was initially developed by Holland [31] back in the 70s, and is stochastic and population-based. The success of GA may stem from its simple and lucid concept of mimicking the natural selection process in biological evolution. To realize this abstraction, GA entails four fundamental operations, i.e., selection, crossover, mutation, and replacement:

1. The selection process chooses some quality individuals from a population.
2. The selected individuals (parents) undergo the crossover (mating) operation and produce new individuals (offsprings). These offsprings have genes (traits) inherited from the selected parents.
3. To prevent the population from having a premature convergence, a mutation procedure is put in place to introduce random perturbations on the genes.
4. One generation (iteration) will end with the replacement operation, where the offsprings are compared against their parents. Individuals having better fitness will survive, while the rests will vanish.

In this subsection we present a collaborative genetic algorithm (C-GA), and portray its principal differences from the conventional GA. As the name reveals, C-GA is constructed on the basis of the genetic algorithm, meaning that all standard GA operators will still be involved. There are many variants of GA. In this work we adopt the one suggested by Chuang et al. [42]. We expect that other versions are equally suitable for incorporating the collaborative part, and will not alter findings of this study.

A pseudocode of C-GA is given in Algorithm 1. The algorithm begins with initializing several populations. Each population is responsible for finding the optimal solution of one particular problem. For example, Φ_i reflects a population (a group of solution candidates) that targets resolving problem i. After initializing all solution candidates, we evaluate the computationally heavy function, f_h, for each candidate (line 4–6 in Algorithm 1). Henceforward a solution candidate, $\vec{x}$, and its corresponding outputs of f_h, $\vec{y}$, are deemed as an entity (object). This implies that copying $\vec{x}$ from one population to another will also copy the corresponding $\vec{y}$. A "while" loop in Algorithm 1 (line 7–22) indicates the optimization iterative process. Referring to this loop, it is evident that all specified problems are being solved simultaneously in C-GA, meaning that in one iteration we update the members (solution candidates) of each population. This is the first distinctive feature of C-GA.

From the underlying idea of GA, we surmise that a population having better genetic information will produce more robust offsprings and eventually improve optimization performances, e.g., quality of the optimal solution found and convergence speed. With the intention to improve the quality of all populations, we perform a collaborative operation (line 9–13 in Algorithm 1) in each iteration. This operation is another exclusive attribute of G-GA. If one disregards the collaborative operation, the code becomes equivalent to the conventional GA. The essence of the operation is to replace some members of a population with better individuals cloned from other populations. The collaborative operation in C-GA consists of two functions (see Algorithm 2). The first function will define a set of promising individuals, Ψ_i, to be cloned and introduced into population Φ_i. These promising individuals must originate from any population other than Φ_i. In addition, the promising individuals in Ψ_i are selected based upon the light function of problem i, $f_{l,i}$. The second function in Algorithm 2 will replace some of the worst members of Φ_i with the promising individuals. One condition for the replacement is that the promising individuals have better fitness than the individuals being replaced. There could be a case where none of the promising individuals in Ψ_i have better fitness than the existing members of Φ_i. In this case, none of Φ_i's members will be replaced (no cloning). Finally, note that the collaborative operation keeps the population size constant.

3.3. Collaborative particle swarm optimization (C-PSO)

Algorithm 3: Collaborative particle swarm optimization (C-PSO)

1 $t \leftarrow 0$ // Iteration counter.
/* Initialize all populations and their members. */
2 **for** $i \leftarrow 1$ **to** N_p **do**
3 $\quad \Phi_i \leftarrow \{(\vec{x}_i^j, \vec{v}_i^j) | j = 1, 2, \ldots, N_m\}$
/* Perform evaluations of a computationally heavy function, f_h. */
4 **for** $i \leftarrow 1$ **to** N_p **do**
5 $\quad$ **for** $j \leftarrow 1$ **to** N_m **do**
6 $\quad\quad \vec{y}_i^j \leftarrow f_h(\vec{x}_i^j)$
/* Initialize particle and global best positions. */
7 **for** $i \leftarrow 1$ **to** N_p **do**
8 $\quad$ **for** $j \leftarrow 1$ **to** N_m **do**
9 $\quad\quad \vec{a}_i^j \leftarrow \vec{x}_i^j$
10 $\quad \vec{\beta}_i \leftarrow \vec{x}_i^b | f_{l,i}(\vec{y}_i^b) \geq f_{l,i}(\vec{y}_i^j),\ j = 1, 2, \ldots, N_m$ // Note $\vec{x}_i^b$ is the best member of Φ_i and $\vec{y}_i^b$ is the outputs of f_h with respect to $\vec{x}_i^b$.
11 **while** *termination criteria is not met* **do**
12 $\quad t \leftarrow t + 1$ // Iteration counter.
$\quad$ /* Perform basic PSO operations for each population. */
13 $\quad$ **for** $i \leftarrow 1$ **to** N_p **do**
14 $\quad\quad$ **for** $j \leftarrow 1$ **to** N_m **do**
15 $\quad\quad\quad \vec{a}_i^j \leftarrow$ **update_particle_best_position**$(\vec{a}_i^j, \vec{x}_i^j, i)$ // This operation compares $\vec{a}_i^j$ and $\vec{x}_i^j$. Note this operation is based upon $f_{l,i}$ only.
16 $\quad\quad \vec{\beta}_i \leftarrow$ **update_global_best_position**$(\vec{\beta}_i, \Phi_i, i)$ // This operation compares $\vec{\beta}_i$ and all members of Φ_i. Note this operation is based upon $f_{l,i}$ only.
17 $\quad$ /* Perform a collaborative operation. */
18 $\quad$ **for** $i \leftarrow 1$ **to** N_p **do**
19 $\quad\quad \vec{\beta}_i \leftarrow$ **enhance_global_best_position**$(\vec{\beta}_i, i)$
20 $\quad$ /* Resume other basic PSO operations for each population. */
21 $\quad$ **for** $i \leftarrow 1$ **to** N_p **do**
22 $\quad\quad$ **for** $j \leftarrow 1$ **to** N_m **do**
23 $\quad\quad\quad \vec{v}_i^j \leftarrow$ **update_particle_velocity**$(\vec{v}_i^j, \vec{x}_i^j, \vec{a}_i^j, \vec{\beta}_i)$
24 $\quad\quad\quad \vec{x}_i^j \leftarrow$ **update_particle_position**$(\vec{v}_i^j, \vec{x}_i^j)$
25 $\quad\quad\quad \vec{y}_i^j \leftarrow f_h(\vec{x}_i^j)$
26 $\quad$ **evaluate_termination_criteria**()

Φ_i: a population assigned for tackling problem i.
N_m: number of solution candidates in Φ_i (size of each population).
$\vec{x}_i^j$: position of particle j that belongs to Φ_i (solution candidate j for problem i).
$\vec{v}_i^j$: velocity of particle j that belongs to Φ_i.
$\vec{y}_i^j$: outputs of a computationally heavy function, f_h, with respect to $\vec{x}_i^j$.
$\vec{a}_i^j$: historical best position of particle j that belongs to Φ_i.
$\vec{\beta}_i$: historical global best position for problem i.

Algorithm 4: A specific function in C-PSO

1 **Function 1:** $\vec{\beta}_i \leftarrow$ **enhance_global_best_position**$(\vec{\beta}_i, i)$
2 $\quad Y_i \leftarrow \emptyset$
3 $\quad$ **for** $q \leftarrow 1$ **to** N_p **do**
4 $\quad\quad$ **if** $q \neq i$ **then**
5 $\quad\quad\quad$ Copy all members of Φ_q, and append to Y_i.
6 $\quad \vec{\gamma}_i \leftarrow$ Choose the best member of Y_i based upon $f_{l,i}$.
7 $\quad \vec{\beta}_i \leftarrow$ Compare $\vec{\beta}_i$ and $\vec{\gamma}_i$ based upon $f_{l,i}$, and keep the best one.
8 $\quad$ **return** $\vec{\beta}_i$

Particle swarm optimization (PSO) is another type of stochastic population-based optimization techniques. This algorithm was first proposed by Kennedy and Eberhart [32] and since then has gained a lot of attention. A large number of modification proposals [43–45] and applications [46–49] of PSO exist in the literature. The search mechanism in PSO is inspired by the social behavior of animals, particularly ones displaying swarm behavior like birds or fish. The animal swarm has its own set of operations to locate a food source as a group. Each individual in the swarm (in PSO, individuals are referred to as particles) continuously updates its search direction (velocity) depending on the learning experiences of its own and other members in the swarm. As reported in [50], PSO has four basic operations:

1. Determine the best position that a particle has encountered; the "particle best position".
2. Determine the best position that the swarm has found so far; the "global best position".
3. Update the velocity of each particle in the swarm. The new velocity is controlled by three factors, i.e., the current velocity, the particle best position, and the global best position.
4. Update the position of each particle based on the current position and the updated velocity.

One key to swarm behavior in PSO is the global best position, which is shared among particles. In a multi-swarm context, a natural extension for a collaborative algorithm is to enhance this global best by sharing information across swarms. One could also introduce cloning or swapping of particles between swarms, similar to the C-GA algorithm in the previous subsection. Here we will introduce a collaborative particle swarm optimization (C-PSO) where PSO has only been extended by a shared global best position. A pseudocode of C-PSO is provided in Algorithm 3. Like C-GA, the algorithm starts with initializing swarms, where each swarm solves for one particular problem in the problem set. The algorithm then continues with completing an iterative routine indicated by the "while" loop statement in Algorithm 3 (line 11–26). As reflected in this loop, we solve all defined problems concurrently, i.e., for each iteration we update the position of every individual in all swarms. C-PSO employs the four standard PSO operations for updating particle positions. Several velocity update formulas can be found in the literature [51–55]. In this work we use the velocity update formula suggested by Shi and Eberhart [33] because it is often deemed as the standard PSO formula [56], but any other formulas for updating velocity can be implemented in C-PSO.

The proposed C-PSO algorithm entails a collaborative operation within the iterative process (line 17–19 in Algorithm 3). This extra operation aims at enhancing the global best position by sharing the objective function data. To better understand this operation, let us concentrate on the improvement of $\vec{\beta}_i$ which represents the global best position for problem i. The collaborative operation is specifically carried out through a function detailed in Algorithm 4. This function replaces the swarm global best position with the best position found by any particle in any swarm. Omitting this collaborative operation will make the C-PSO code identical to the regular PSO.

3.4. Collaborative gradient descent (C-GD)

Algorithm 5: Collaborative gradient descent version I (C-GD-I)

1 $t \leftarrow 0$ // Iteration counter.
/* Initialize all points. */
2 **for** $i \leftarrow 1$ **to** N_p **do**
3 $\quad \vec{x}_i \leftarrow$ **specify_initial_position**(i)
/* Perform evaluations of a computationally heavy function, f_h. */
4 **for** $i \leftarrow 1$ **to** N_p **do**
5 $\quad \vec{y}_i \leftarrow f_h(\vec{x}_i)$
6 **while** *termination criteria is not met* **do**
7 $\quad t \leftarrow t + 1$ // Iteration counter.
8 $\quad$ /* Perform a collaborative operation. */
9 $\quad R \leftarrow$ **determine_replacement_scenario**() // Where $R = \{\vec{r}_i | i = 1, 2, \ldots, N_p\}$ and $\vec{r}_i$ is the replacement candidate for $\vec{x}_i$.
10 $\quad$ **for** $i \leftarrow 1$ **to** N_p **do**
11 $\quad\quad (\vec{x}_i, \vec{y}_i) \leftarrow (\vec{r}_i, \vec{s}_i)$ // Here we implement the replacement scenario. Note that $\vec{s}_i = f_h(\vec{r}_i)$ has already been calculated, so we do not need any new heavy calculations.
12 $\quad$ /* Perform basic GD operations for each point. */
13 $\quad$ **for** $i \leftarrow 1$ **to** N_p **do**
14 $\quad\quad \vec{g}_i \leftarrow \nabla f_i(\vec{x}_i)$ // Where $f_i = f_{l,i} \circ f_h$ and $f_{l,i}$ denotes the computationally light function embedded in problem i.
15 $\quad\quad \alpha_i \leftarrow$ **decide_step_length**$(\vec{x}_i, \vec{g}_i, f_i)$
16 $\quad\quad \vec{x}_i \leftarrow$ **update_position**$(\vec{x}_i, \vec{g}_i, \alpha_i)$
17 $\quad\quad \vec{y}_i \leftarrow f_h(\vec{x}_i)$
18 $\quad$ **evaluate_termination_criteria**()

$\vec{x}_i$: a point dedicated for tackling problem i (solution candidate for problem i).
$\vec{y}_i$: outputs of a computationally heavy function, f_h, with respect to $\vec{x}_i$.

Algorithm 6: The replacement function for C-GD-I

1 **Function 1:** $R \leftarrow$ **determine_replacement_scenario**()
2 $\quad R \leftarrow \emptyset$
3 $\quad$ **for** $i \leftarrow 1$ **to** N_p **do**
4 $\quad\quad$ Choose $\vec{r}_i \in \{\vec{x}_q | q = 1, 2, \ldots, N_p\}$ that maximizes $f_{l,i}(\vec{s}_i)$. Without losing generality, here we take a maximization problem as the example. Note that $\vec{s}_i = f_h(\vec{r}_i)$ has already been calculated, so we do not need any new heavy calculations.
5 $\quad\quad$ Append $\vec{r}_i$ to R.
6 $\quad$ **return** R

Algorithm 7: The replacement function for C-GD-II

```
1 Function 1: R ← determine_replacement_scenario()
2     Define all swap scenarios. These scenarios come from the permutation of
      {x⃗_q | q = 1, 2, ..., N_p}, and hence we have a total N_r = P(N_p, N_p) = N_p! scenarios.
3     for k ← 1 to N_r do
4         Evaluate H(k) which computes the aggregate objective value of a particular swap
          scenario k. The function is expressed as H(k) = Σ_{i=1}^{N_p} f_i(x⃗_{i,k}), where f_i = f_{l,i} ∘ f_h
          and x⃗_{i,k} denotes a new solution candidate for problem i under swap scenario k.
          Note that H(k) is precisely computed as H(k) = Σ_{i=1}^{N_p} f_{l,i}(y⃗_{i,k}), where the heavy
          computation of x⃗_{i,k} in y⃗_{i,k} = f_h(x⃗_{i,k}) is conducted only once.
5     R ← Choose a swap scenario that maximizes H(k).
6     return R
```

Gradient descent (GD) is one of the simplest algorithms in the field of numerical optimization. The algorithm is classified as a local search method since it returns a local optima of a differentiable objective function. The idea behind GD is to repeatedly make a move within the given search space, where the movement itself is guided by the slope of the response surface. GD enforces two principal operations:

1. Evaluate the gradient of the objective function.
2. Update the position based on the predetermined gradient direction.

This subsection presents a collaborative gradient descent (C-GD) built upon the conventional GD algorithm. Even though GD is seldom employed in simulation-based optimization as it can only find a local optima, we herein present C-GD to exemplify the applicability of collaborative concept, not only on population-based but also on gradient-based optimization methods. A pseudocode of the first version of C-GD, abbreviated as C-GD-I, is given in Algorithm 5. The optimization procedure starts by defining a set of starting points. Each of these points reflects an initial solution for a specific member of the problem set, e.g., $\vec{x}_i$ denotes a solution candidate for problem i. After initialization, the optimization iterative routine commences (line 6–18 in Algorithm 5). In contrast to traditional approaches, C-GD seeks for the optimal solutions of all defined problems simultaneously, in the sense that we advance all the points toward better locations on every single iteration. When updating the point location, we remain using the traditional GD operations.

Although GD is fast and efficient, the solution is often affected by the choice of starting location. Especially for an optimization problem having a multi-modal objective function, the search process using GD might get stuck in a local optima. The presented C-GD-I algorithm extends the traditional GD by the collaborative operation (line 8–11 in Algorithm 5). This operation helps preventing the search process from getting trapped at local optima, particularly in the early iterations, and thus provides an opportunity to improve the search performance. In essence, through the utilization of shared objective function data, the collaborative operation will upgrade the current solution, $\vec{x}_i$, with a better replacement candidate. The procedure for deciding the best replacement candidates is elaborated in Algorithm 6. Again, the C-GD-I algorithm is equivalent to the traditional GD when we exclude the collaborative operation.

A possible downside of the replacement scenario yielded from Algorithm 6 is the fact that some of the replacement candidates could be identical. This implies that the replacement candidates have lower diversity, which may cause premature convergence. With the goal of maintaining the diversity of the replacement candidates, and thus the degree of exploration within the search space, we introduce and examine another collaborative version of GD called C-GD-II. This algorithm is similar to C-GD-I, except for the strategy in deciding the replacement candidates (detailed in Algorithm 7). In this replacement strategy we only swap the current solution of one problem with the current solution of another problem in such a way that the swap scenario will maximize an aggregated objective function, $H(k)$. Maximizing $H(k)$ is the most straightforward approach for deciding the swap scenario, however other procedures, e.g., which add some randomness, could be more effective. Referring to Algorithm 7, we implement an exhaustive search in order to find the best swap scenario. Note that the search for the best swap scenario in Algorithm 7 can become a computationally demanding task, since the number of swap scenarios to evaluate grows factorially with the number of problems, i.e., $N_r = N_p!$. It is possible to treat the search task as a combinatorial problem analogous to the traveling salesman problem, and then use algorithms that can solve such a combinatorial problem efficiently, like dynamic programming [57,58], branch-and-bound [59,60], or branch-and-cut [61] algorithms. Such efforts are beyond the scope of this article.

3.5. Number of function evaluations & computational cost

With reference to Fig. 1, the number of objective function evaluations in traditional non-collaborative approaches, $N_{f,nc}$, is expressed as:

$$N_{f,nc} = N_t \cdot N_m \cdot N_p \tag{1}$$

where N_t and N_p indicate the number of iterations and the number of problems, respectively. For GA and PSO, $N_m \geq 1$ indicates the number of solution candidates in each population, while for GD, $N_m = 1$ because there is only one solution candidate to evaluate for each problem in every iteration. The equation above could also reflect the number of objective evaluations that occur in MTO algorithms. Further, referring to Fig. 2, the number of objective function evaluations in collaborative algorithms, $N_{f,c}$, is formulated as:

$$N_{f,c} = N_t \cdot N_m \cdot (N_p)^2 \tag{2}$$

The second-order exponent for N_p comes from the fact that in collaborative algorithms the heavy function (simulation) outcomes are shared and used for computing N_p different light functions embedded in the problems. The ratio between $N_{f,c}$ and $N_{f,nc}$ then becomes:

$$\frac{N_{f,c}}{N_{f,nc}} = N_p \tag{3}$$

This ratio implies that the search conducted by the collaborative algorithms is guided by a larger number of solution candidates compared to the sampling performed by either the non-collaborative methods or the MTO algorithms. We therefore can expect that the collaborative algorithms will outperform the non-collaborative methods as well as the MTO algorithms.

On the one hand, run time for the traditional non-collaborative techniques, T_{nc}, can be estimated using:

$$T_{nc} = N_t \cdot N_m \cdot N_p \cdot (T_h + T_l) \tag{4}$$

where T_h and T_l symbolize the computation time of the heavy and light functions, respectively. This equation is also valid for the MTO algorithms. On the other hand, run time of the collaborative algorithms, T_c, is approximately given by:

$$T_c = N_t \cdot N_m \cdot N_p \cdot (T_h + N_p \cdot T_l) \tag{5}$$

The multiplier for T_l again arises from the extended utilization of the results of heavy function evaluation. Dividing T_c by T_{nc} gives us:

$$\frac{T_c}{T_{nc}} = \frac{T_h + N_p \cdot T_l}{T_h + T_l} = 1 + \frac{(N_p - 1) \cdot T_l}{T_h + T_l} = 1 + \frac{N_p - 1}{\left(\frac{T_h}{T_l} + 1\right)} \tag{6}$$

The relation between $\frac{T_h}{T_l}$ and $\frac{T_c}{T_{nc}}$ is illustrated in Fig. 3. According to the relation, $\frac{T_c}{T_{nc}}$ monotonically decreases as $\frac{T_h}{T_l}$ increases, and it is asymptotic to one when $\frac{T_h}{T_l} \to +\infty$. In other words, if the heavy function is much more expensive to calculate than the light function, the additional computing time needed by the collaborative algorithms is negligible. Despite the limited increase in computational cost, the increase in function evaluations, as given by Eq. (3), can be large. Thus, for large $\frac{T_h}{T_l}$ the collaborative feature enables a larger sampling size at a similar computational cost as the non-collaborative approaches or the MTO algorithms.

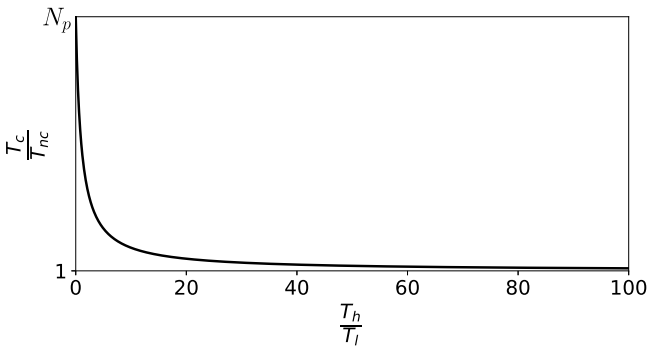


Fig. 3. Relation between $\frac{T_h}{T_l}$ and $\frac{T_c}{T_{nc}}$, as given by Eq. (6).

4. Descriptions of problem sets

Four problem sets are considered for comparing the collaborative algorithms against both traditional optimization techniques (i.e., GA, PSO, and GD) and an existing MTO algorithm (i.e., MFEA). These problem sets are intended to showcase the benefits of our proposed algorithms, without having the complexity and the computational cost where our proposed algorithms will be most beneficial. Even though the presented problem sets do not necessarily have computationally heavy function f_h, we want to compare the efficiency of optimization process with respect to the same number of f_h evaluations, as the proposed algorithms are intended for problems sets where f_h is dominating the computational time.

For the first three problem sets, analytical objective functions are defined and used. These first three problem sets do not actually involve any heavy simulation, and are intended for illustrating the difference between the traditional and collaborative algorithms. Note that the exclusion of heavy function evaluation has no influence on the performance of the collaborative algorithms in terms of final solution and objective function value. Thus, it is still possible to compare the performance of the collaborative and non-collaborative approaches along these lines for these simple problem sets. For the fourth problem set, we have an example of the special MTO case with characteristics as described in Section 2. The goal of this last problem set is to demonstrate a more realistic application of the collaborative algorithms.

4.1. Problem Sets #1, #2, and #3

In these problem sets, we aim at finding a solution candidate for problem i, $\vec{x}_i = \left(x_{i,1}; x_{i,2}; \ldots; x_{i,N_d}\right) \in \mathbb{R}^{N_d}$, that maximizes the following objective function:

$$f_{l,i}(\vec{x}_i) = -A \cdot N_d + \sum_{k=1}^{N_d} \left(A \cdot \cos\left(2\pi \left(x_{i,k} + s_{i,k}\right)\right) - \left(x_{i,k} + s_{i,k}\right)^2 \right) + \sum_{k=1}^{N_d} \left(m_{i,k} \left(x_{i,k} + s_{i,k}\right)\right) \tag{7}$$

where $\vec{s}_i = \left(s_{i,1}; s_{i,2}; \ldots; s_{i,N_d}\right) \in \mathbb{R}^{N_d}$ and $\vec{m}_i = \left(m_{i,1}; m_{i,2}; \ldots; m_{i,N_d}\right) \in \mathbb{R}^{N_d}$ are sets of constants and coefficients, respectively, which are specific for problem i.

In Problem Set #1 we define nine two-dimensional optimization problems, thus $N_p = 9$ and $N_d = 2$. Values of $\vec{s}_i$, $\vec{m}_i$, and A are given in Table 1. The objective function above is constructed based on the Rastrigin function [62]. Some modifications of the original function include (i) the addition of a linear plane with slope $\vec{m}_i$ which is represented by the last term of Eq. (7), and (ii) a shift of the entire response surface through the set of constants $\vec{s}_i$. Furthermore, all the optimization problems have an identical search space, i.e., $\vec{x}_i \in [-5.12, 5.12]^{N_d}$.

Table 1
Constants and coefficients for objective functions in Problem Set #1.

Problem i	N_d	A	$\vec{s}_i$	$\vec{m}_i$
1	2	10	[0, 0]	[10, 10]
2	2	10	[0.125, 0.125]	[10, 7.5]
3	2	10	[0.25, 0.25]	[10, 5]
4	2	10	[0.375, 0.375]	[10, 2.5]
5	2	10	[0.5, 0.5]	[10, 0]
6	2	10	[0.625, 0.625]	[10, −2.5]
7	2	10	[0.75, 0.75]	[10, −5]
8	2	10	[0.875, 0.875]	[10, −7.5]
9	2	10	[1, 1]	[10, −10]

Table 2
Constants and coefficients for objective functions in Problem Set #2.

Problem i	N_d	A	$\vec{s}_i$	$\vec{m}_i$
1	3	10	[0, 0, 0]	[10, 10, 10]
2	3	10	[0.125, 0.125, 0.125]	[10, 7.5, 7.5]
3	3	10	[0.25, 0.25, 0.25]	[10, 5, 5]
4	3	10	[0.375, 0.375, 0.375]	[10, 2.5, 2.5]
5	3	10	[0.5, 0.5, 0.5]	[10, 0, 0]
6	3	10	[0.625, 0.625, 0.625]	[10, −2.5, −2.5]
7	3	10	[0.75, 0.75, 0.75]	[10, −5, −5]
8	3	10	[0.875, 0.875, 0.875]	[10, −7.5, −7.5]
9	3	10	[1, 1, 1]	[10, −10, −10]

Imposing unique $\vec{s}_i$ and $\vec{m}_i$ for each optimization problem means that all the problems have different objective functions. Response surfaces of three (out of nine) optimization problems under study are visualized in Fig. 4. The black, blue, and red squares in the figure indicate different optimal solutions for different problems. Besides, we can observe the nature of the objective functions which have many local optima.

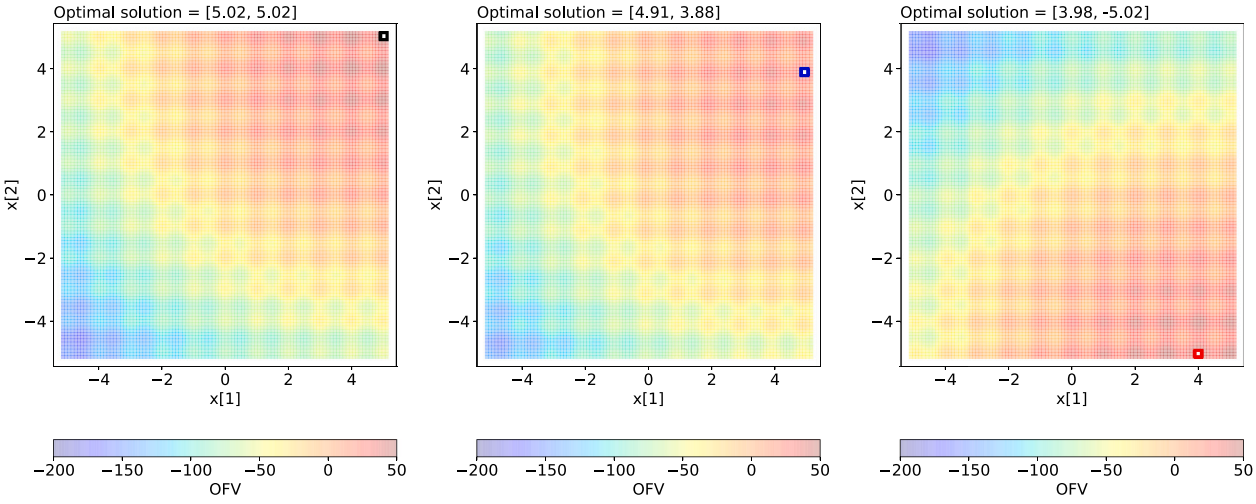
Problem Set #2 is similar to Problem Set #1, except for the number of variables to optimize is increased to $N_d = 3$. The goal is to assess and compare the performance of our cooperative algorithms on a more complex setting. Values of $\vec{s}_i$, $\vec{m}_i$, and A are detailed in Table 2.

Problem Set #3 is included for investigating the significance of our collaborative algorithms on tackling optimization problems with smooth and uni-modal objective functions. To establish these kind of functions, we simply set the coefficient A in Eq. (7) to zero, while keeping the other components as in Problem Set #1.

Several parameters needed by our collaborative algorithms and the traditional optimization techniques for solving Problem Set #1 to #3 are listed in Table 3 (for GA and C-GA), Table 4 (for PSO and C-PSO), and Table 5 (for GD and C-GD). Besides, we employ the basic MFEA presented in [3,36,38] for solving Problem Set #1 to #3. Parameters for the MFEA are given in Table 6. The initial solution candidates are selected using the Latin Hypercube Sampling method [63], aiming to better distribute them across the entire search space. Moreover, the same set of initial solution candidates is used for both the collaborative algorithms and the traditional methods as well as the MFEA to provide a fair comparison. We implement a reflective approach to deal with the boundary constraints, where a newly-generated solution candidate is reflected away from the boundary if it is going outside the defined search space. Lastly, we repeat the optimization processes with 1000 different sets of initial solution candidates in order to have a solid and unbiased comparison.

4.2. Problem Set #4

For this problem set, we take an example of optimization in the field of petroleum engineering, specifically from the work of Angga et al. [64]. The hydrocarbon recovery process is usually aided with an injection of external fluids, like water or gas, into the reservoir. The injection scenario, i.e., the injection rates and pressures implemented over a given production period, affects not only the volume



(a) Problem 1. (b) Problem 2. (c) Problem 9.

Fig. 4. Response surfaces of three optimization problems in Problem Set #1.

Table 3
Parameter setting for GA and C-GA.

Parameter	Value	Description
N_t	10	Number of iterations
N_m	$2 \cdot N_d$	Size of each population
p	$\frac{1}{N_m}$	Proportional parameter for selection operation [42]
λ	0.1	Probability threshold for conducting crossover operation [42]
ϕ_0	0.25	Bound for random perturbations during mutation operation [42]
b	4	Parameter that controls mutation step size [42]
N_c	1	Number of promising individuals to be cloned

Table 4
Parameter setting for PSO and C-PSO.

Parameter	Value	Description
N_t	10	Number of iterations
N_m	$2 \cdot N_d$	Size of each population
ω	0.5	Inertia weight [50]
c_p	2	Cognitive factor [50]
c_g	2	Social factor [50]

Table 5
Parameter setting for GD and C-GD.

Parameter	Value	Description
N_t	10	Number of iterations

Table 6
Parameter setting for MFEA.

Parameter	Value	Description
N_t	10	Number of iterations
N_m	$2 \cdot N_d$	Number of individuals (in the single population) that have the same skill factor
p_c	1	Probability for the simulated binary crossover (SBX) operator [36]
η_c	15	Distribution index for the SBX operator [36]
p_m	$\frac{1}{N_d}$	Probability for the polynomial mutation (PM) operator [36]
η_m	15	Distribution index for the PM operator [36]
rmp	0.3	Random mating probability [36]

of hydrocarbon produced, but also the amount of CO_2 gas emitted. The optimization goal is thus to find an injection scenario that maximizes the revenue from production minus the operational cost and carbon tax. The carbon tax itself is a product of the CO_2 tax rate, r_{CO_2}, and the amount of carbon emissions.

Angga et al. conduct a sensitivity analysis to identify the influence of r_{CO_2} on the optimal injection scenario [64]. In that work, eleven optimization problems with different r_{CO_2} values are defined (the r_{CO_2} values are uniformly sampled between 0 and 525 USD/ton CO_2). In these optimization problems, the calculation of each objective function comprises two stages as discussed in Section 2. The first one is to predict the reservoir performance under a particular injection scenario, $\vec{x}$, by means of a numerical simulation, f_h. The simulation results, $\vec{y} = f_h(\vec{x})$, are then used to estimate the revenue, $R(\vec{y})$, operating cost, $C(\vec{y})$, and quantity of carbon emissions, $M_{CO_2}(\vec{y})$; which all constitute the objective function expressed as follows.

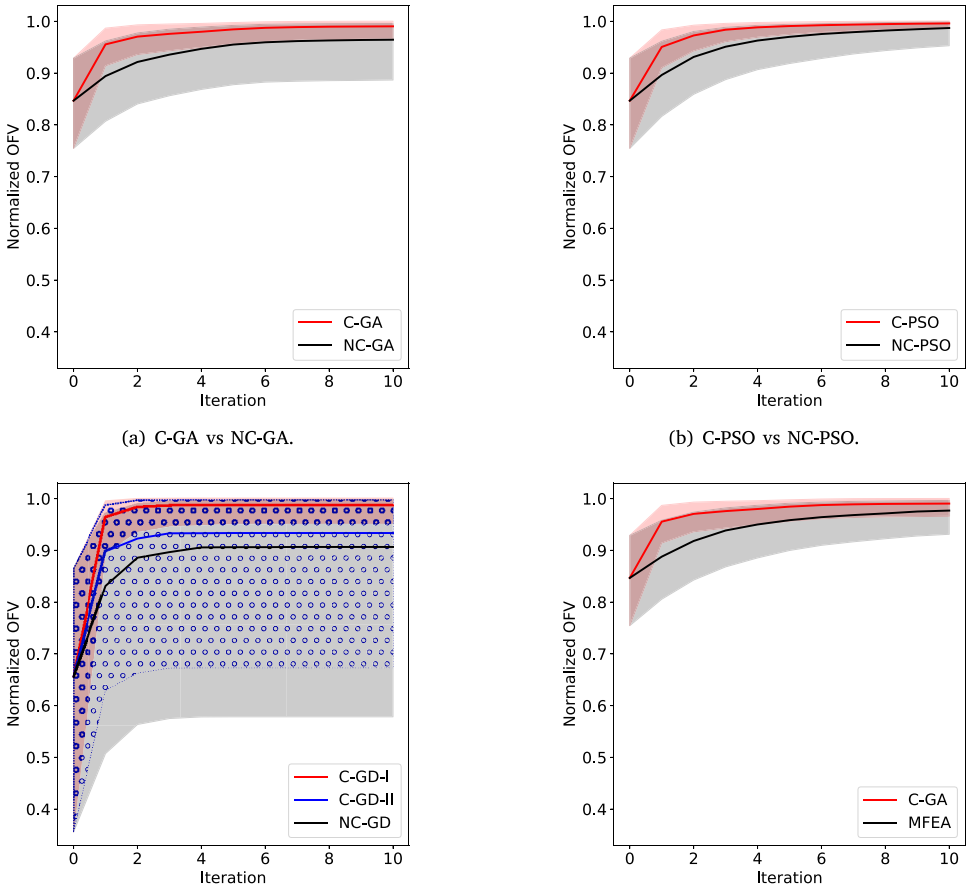
$$f_{l,i}\left(\vec{y}\right) = R\left(\vec{y}\right) - C\left(\vec{y}\right) - r_{CO_2,i} \cdot M_{CO_2}\left(\vec{y}\right) \tag{8}$$

where $r_{CO_2,i}$ represents the CO_2 tax rate embedded in problem i. The numerical reservoir simulation, f_h, is the computationally expensive part of the objective function evaluation; one simulation finishes in around four seconds, while the latter function $f_{l,i}$ completes in around one tenth of a second. Other details about the optimization problems, including a complete description of the objective function, are presented in [64].

5. Results and discussions

5.1. Problem Set #1

This subsection presents and discusses the performance of optimization algorithms while solving Problem Set #1. The first point of discussion is about the progression of the objective function value (OFV) displayed in Fig. 5. The first three subfigures intend to compare the performance of our collaborative algorithms (prefix "C-") against the traditional non-collaborative optimization algorithms (prefix "NC-"), whereas the last subfigure compares our C-GA against the MFEA as they both adopt the principles of natural evolution. As a reminder, we have nine optimization problems in Problem Set #1 and run the optimization algorithms 1000 times (under different sets of initial solution candidates). We therefore have 9000 "iteration vs OFV" curves for every optimization algorithm examined. To draw a solid line in Fig. 5, we first normalize the "iteration vs OFV" curves with the minimum



(a) C-GA vs NC-GA.

(b) C-PSO vs NC-PSO.

(c) C-GD vs NC-GD.

(d) C-GA vs MFEA.

Fig. 5. Progression of objective function value (OFV) for Problem Set #1.

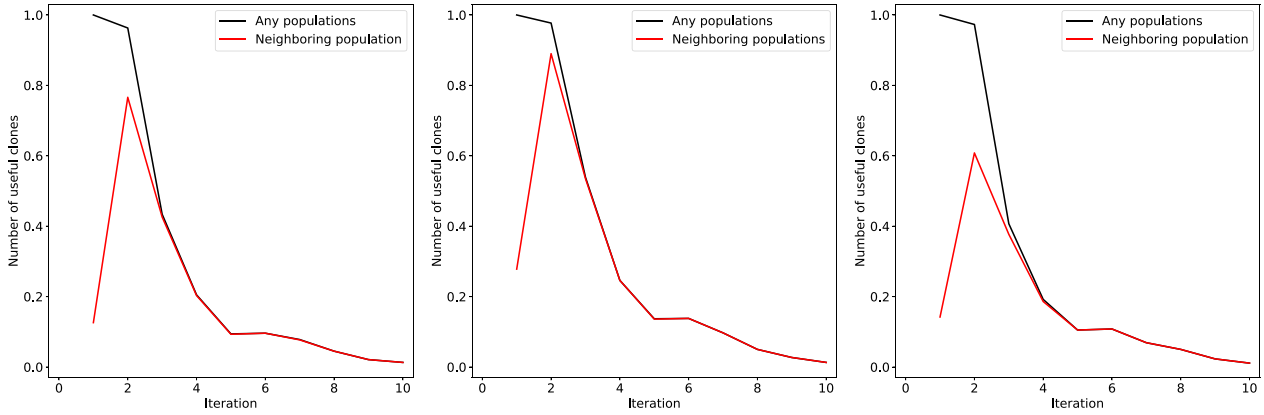
and maximum OFVs of the corresponding optimization problem. The solid line is then drawn representing the median (P50) of those 9000 normalized curves, while the shaded (or the pattern-filled) area is constructed based on the P10 and P90 lines. We depict these parameters instead of the arithmetic average and standard deviation because we have skewed distributions. The steps above are repeated to process the results of other optimization algorithms.

Referring to Fig. 5, we can safely say that our collaborative algorithms outperform not only the non-collaborative optimization methods but also the MFEA. One clear advantage is that the collaborative algorithms yield optimal solutions with higher OFVs. We also notice that the variations in OFVs, particularly at the final iteration, are smaller for the collaborative algorithms. This implies that the collaborative algorithms produce more consistent solutions and are less dependent on the initial solution candidates being used. Furthermore, the collaborative algorithms have faster convergence; fewer iterations are required before reaching a stable condition or plateau where the search process does not longer significantly improve the current solutions. All these gains originate from the collaborative operation (see Section 3.1). The extended utilization of the simulation results enables the collaborative algorithms to enhance their sampling of the search space, and thus offering more information when generating new solution candidates.

Fig. 5(c) shows that the performance of C-GD-II is worse than C-GD-I. As explained in Section 3.4, C-GD-II employs a specific procedure that is intended to preserve the diversity among the solution candidates when deciding a replacement scenario (see Algorithm 7). This was included to maintain the degree of exploration with the aim of improving the optimization performance. Apparently, this does not hold for this particular example, as C-GD-II performs worse than C-GD-I. The procedure in fact restricts the movement during the collaborative operation, and thus decreases the performance gain. This highlights that the performance gain due to collaborative operation is affected by the way the simulation results are shared.

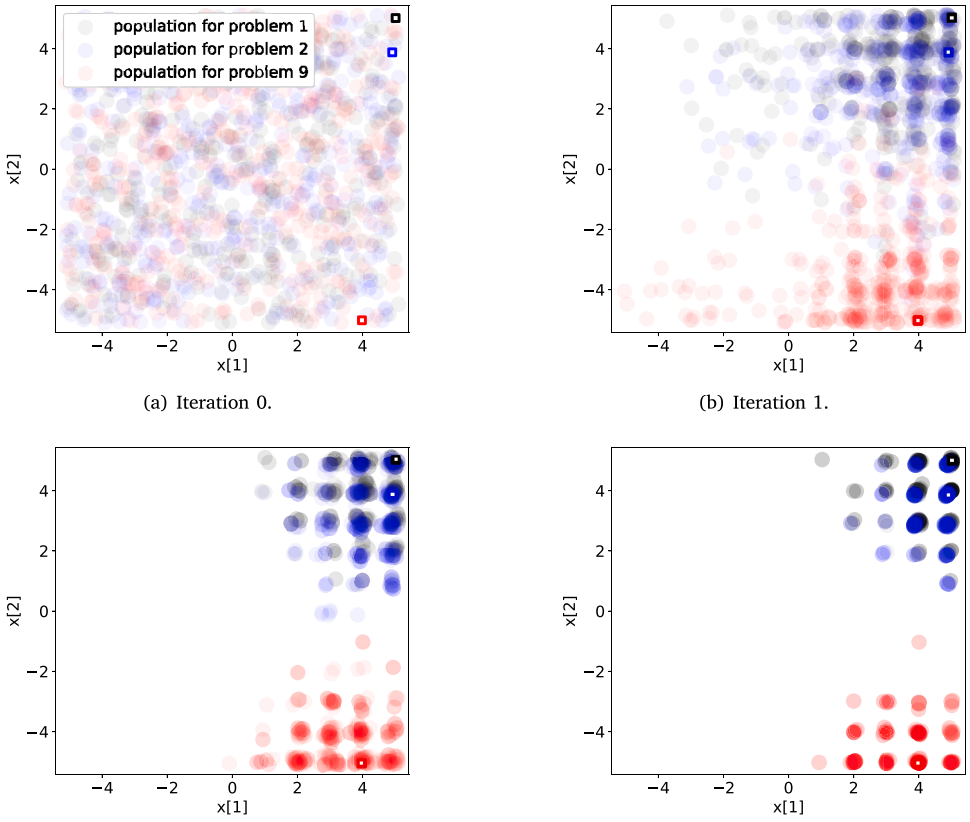
The origin of useful information is another interesting observation. Here, the term "useful information" refers to individuals (or solution candidates) of other populations which will influence the creation of new solution candidates for a particular problem. For C-GA, this term refers to the clones that improve the population Φ_i (see Algorithm 1). For C-PSO, this term refers to the particles that enhance the global best position $\vec{\beta}_i$ (see Algorithm 3). While in this discussion we only consider C-GA, the same argumentation holds for C-PSO too. Fig. 6 depicts the source of useful clones in C-GA while solving Problem Set #1. The black lines indicate the number of useful clones taken from any other populations, while the red lines denote the number of useful clones coming from populations for the "neighboring" problems. The term "neighboring" refers to optimization problems which have adjacent optimal solutions. Typically, "neighboring" problems have similar objective functions, i.e., relatively small differences in objective function parameters (e.g., coefficients and constants). In Problem Set #1, Problem 1 and 2 are deemed "neighboring" since they have adjacent solutions (see Fig. 4), while the neighbors of Problem 2 are Problem 1 and 3. Note that the number of useful clones is in fact an integer; however, the black and red lines in Fig. 6 are pointing to decimal values since they represent the arithmetic averages of 1000 runs.

Referring to the black lines in Fig. 6, the number of useful clones generally declines as the iteration count increases. We also notice that, in the early stage of optimization, the useful clones are taken



(a) Problem 1. (b) Problem 2. (c) Problem 9.

Fig. 6. Source of useful clones in C-GA while solving Problem Set #1.



(a) Iteration 0. (b) Iteration 1.

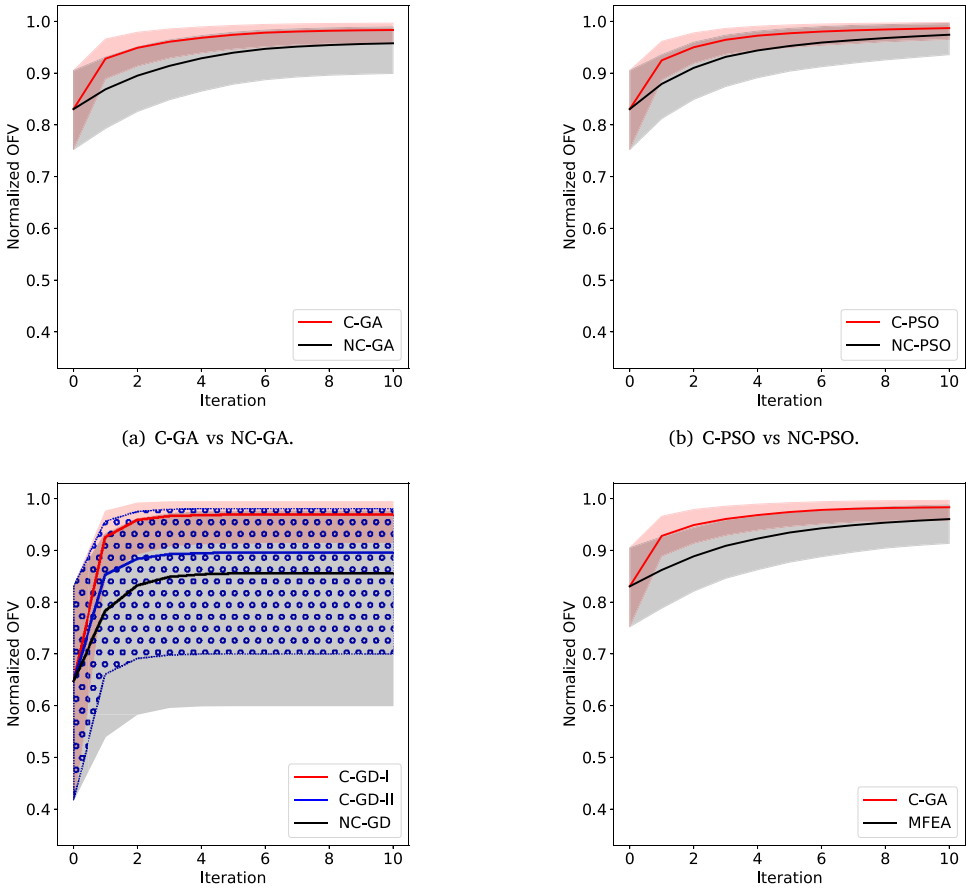
(c) Iteration 2. (d) Iteration 10.

Fig. 7. Overlapping search path in C-GA while solving Problem Set #1.

from any population irrespective of objective function similarity. As the optimization progresses, on the other hand, the useful clones are chiefly taken from populations corresponding to the "neighboring" problems. This behavior can be explained with Fig. 7, which visualizes the overlapping search path in C-GA. Circles in this figure represent the solution candidates for different problems (distinguished by their colors), while the squares denote the "true" optimal solutions for the problems (as shown in Fig. 4). In the early stage of optimization the solution candidates for Problem 9 still spread across the search space, so they can be helpful for Problem 1 despite the relatively long distance between the optimal solutions of Problems 1 and 9 (see Figs. 7(a) and 7(b)). Conversely, later in the optimization, the solution candidates for a particular problem center around the problem's optimal solution. Therefore, the solution candidates for the "neighboring" problems, in this case Problem 2, are the most useful clones for Problem 1 (see Figs. 7(c) and 7(d)). The same behavior is also observed for C-PSO.

5.2. Problem Set #2 and #3

Problem Set #2 is a higher dimensional version of Problem Set #1. The progression of OFV for Problem Set #2 is shown in Fig. 8. We have similar interpretations of this figure as in Section 5.1. A minor



(a) C-GA vs NC-GA. (b) C-PSO vs NC-PSO.

(c) C-GD vs NC-GD. (d) C-GA vs MFEA.

Fig. 8. Progression of objective function value (OFV) for Problem Set #2.

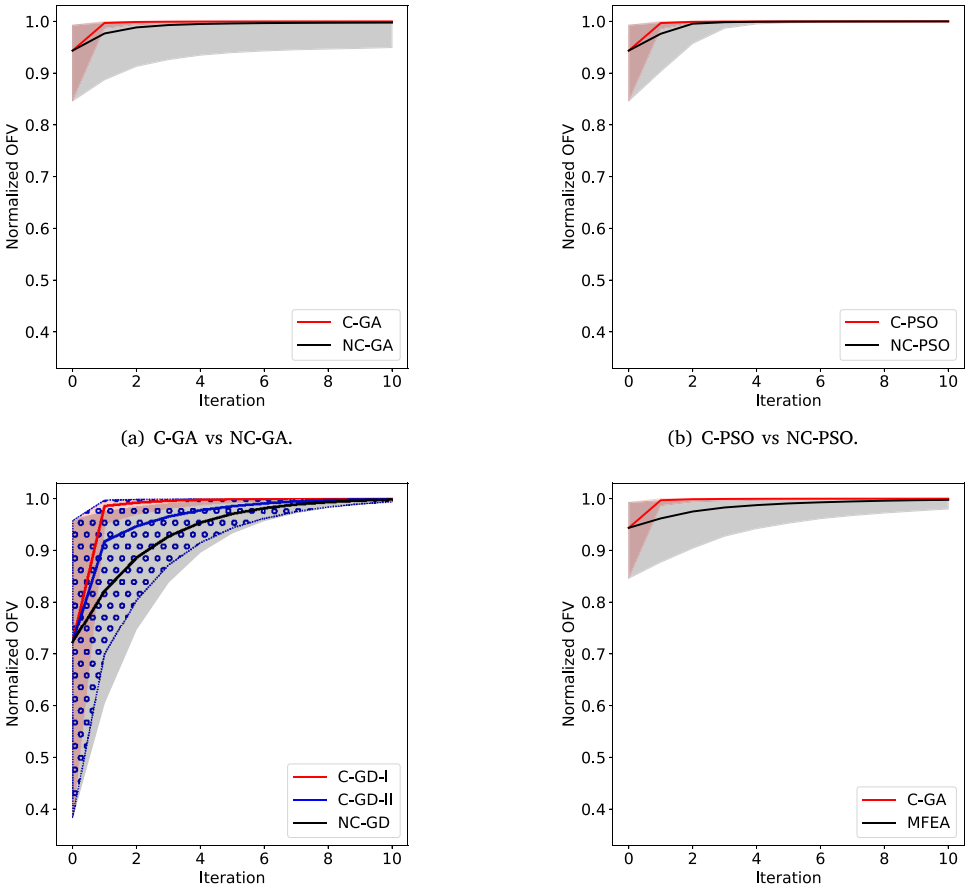
distinction is that all the optimization algorithms require more iterations to converge due to the higher number of decision variables. While the first two problem sets employ multi-modal objective functions, the third problem set consists of smooth and uni-modal functions. The progression of OFV for Problem Set #3 is provided in Fig. 9. Most of our observations in Section 5.1 also apply for this figure. Even though the optimal solutions found by our collaborative algorithms are similar to the ones obtained from the traditional non-collaborative methods, the collaborative algorithms converge faster. The similarity between collaborative and non-collaborative final solutions is likely due to the simpler objective function characteristics of this problem set, enabling all methods to identify the optimal solutions.

Problem Set #1, #2, and #3 do not include the heavy computation part in their objective function evaluation, meaning that $T_h = 0$, while the light computation parts in these problem sets entail fractions of a second to complete. By inserting $T_h = 0$ into Eq. (6), we have $\frac{T_c}{T_{nc}} = N_p$. Since each of the problem sets has nine optimization problems, $N_p = 9$, we therefore have $\frac{T_c}{T_{nc}} = 9$. This implies that the run times of the collaborative algorithms are around nine times of the traditional methods when solving for Problem Set #1, #2, and #3. Now imagine that we include a heavy function that maps $\vec{x}_i$ to $\vec{x}_i$ in Problem Set #1, #2, and #3, but that function involves a time delay to represent a heavy and complex calculation in real-world problems. The inclusion of such heavy function will increase the optimization run time and lower the ratio of $\frac{T_c}{T_{nc}}$. It however will not influence the performance of the collaborative algorithms, including the quality of solutions, the consistency, and the convergence speed, shown in Figs. 5, 8, and 9.

5.3. Problem Set #4

In this problem set, the objective function calculation involves a numerical simulation (i.e., the heavy function), which prolongs the overall optimization run time. We therefore run the optimization algorithms, particularly C-PSO and the traditional PSO, for only five times with different initial particle populations. Note that the numerical simulation involved in this problem set is rather simple where each simulation lasts for around four seconds, $T_h = 4$ s. In real-world problems, numerical simulations can last in order of minutes, hours, or even days. For such heavy and complex simulation, the ratio $\frac{T_c}{T_{nc}}$ given in Eq. (6) will be approximately equal to one, meaning that the addition of computation time due to the collaborative operation will be insignificant compared to the total run time of traditional non-collaborative optimization methods. Using a 9-core workstation, each C-PSO run in this problem set entails around 57 h to finish, while the traditional PSO needs around 44 h (11 optimization problems × 4 h for solving each problem). As the numerical simulation in this problem set is rather simple, the total run time of C-PSO is significantly higher than the total run time of the traditional PSO. We have $T_c/T_{nc} \simeq 57/44 \simeq 1.3$, which is quite close to an estimate of 1.25 obtained from Eq. (6) with $N_p = 11$, $T_h = 4$ s, and $T_l = 0.1$ s.

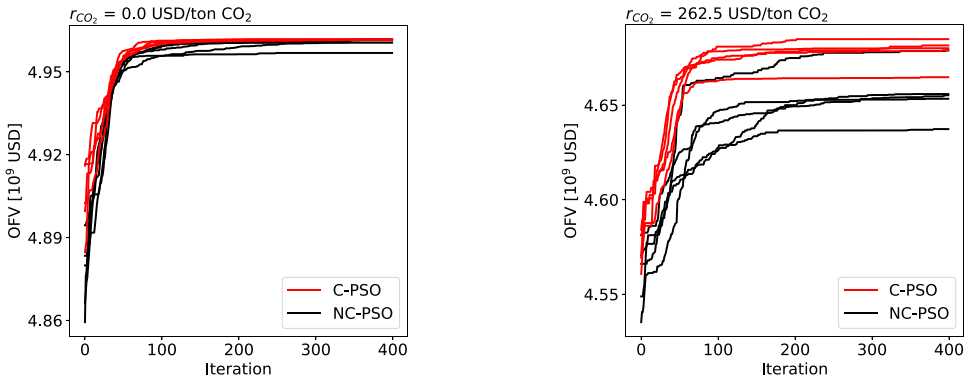
The OFV progressions for two problems are presented in Fig. 10. Despite the limited increase in computational cost, Fig. 10 shows that C-PSO runs have overall higher OFVs and also quicker convergence compared to runs performed with the traditional PSO. This improved optimization performance comes from the fact that C-PSO enlarges the sampling size by a factor of eleven as given by Eq. (3). Referring to Fig. 10, we observe that relative solution improvements are not as



(a) C-GA vs NC-GA. (b) C-PSO vs NC-PSO.

(c) C-GD vs NC-GD. (d) C-GA vs MFEA.

Fig. 9. Progression of objective function value (OFV) for Problem Set #3.



(a) Problem 1. (b) Problem 6.

Fig. 10. Progression of objective function value (OFV) for Problem Set #4.

significant for Problem 1 (see Fig. 10(a)) as they are for Problem 6 (see Fig. 10(b)). One possible explanation is that Problem 1 is considerably different from the other problems in the set, and therefore has fewer and a longer distance to its neighboring problems which can assist the search process. By contrast, the middle problem, i.e., Problem 6, may have many and close neighboring problems, from which the search process can get substantial contributions. Note that the OFV scales in Figs. 10(a) and 10(b) are different because Problem 1 and Problem 6 involve different objective functions, in particular, they enforce different CO_2 tax rates, r_{CO_2}.

6. Clarifications on the term "collaborative"

In the literature there are optimization algorithms which put the "cooperative" word in their names. The term is somewhat similar in meaning as the "collaborative" term we use, and the distinction therefore needs some clarifications. One of the algorithms is the cooperative co-evolutionary algorithm (CCEA), which was coined by Potter and Jong [65] in 1994. Since then many developments and implementations of CCEA have been reported, e.g., in [66–69]. CCEA is beneficial particularly for solving a high dimensional SOO problem, meaning that

the problem includes a large number of variables to optimize. Based on a divide-and-conquer strategy, CCEA decomposes a high dimensional problem into several subproblems with fewer variables to optimize, and resolves these subproblems individually. Solutions from different subproblems are then recombined to update all subproblems as the iteration progresses and to eventually form an overall solution for the high dimensional problem. This is probably the reason CCEA has the "cooperative" term in its name. Again, the nature of problem tackled by CCEA and the collaborative algorithms is different; CCEA solves just one SOO problem, while the collaborative algorithms solve a collection of SOO problems simultaneously.

7. Conclusions

We propose a collaborative optimization framework that is effective for solving a special multi-task optimization (MTO) case consisting of several single-objective optimization problems. In this special MTO case, each problem has a specific objective function and the evaluation of this objective function consists of two steps. The first step involves a computationally heavy function, like running a numerical simulation. The second step is a computationally light function, e.g., the simulation results enter as input into a simple equation, for example a net present value calculation. This light function is unique for every problem in the MTO case, e.g., a unique discount rate in the net present value, while the heavy function is identical across all problems.

The proposed collaborative algorithms have two distinctive features, i.e., (i) they solve all the optimization problems simultaneously, and (ii) they perform a "collaborative" operation in every iteration. In this collaborative operation, the simulation results are shared across all problems, enhancing the sampling capacity of the algorithms. The added information improves the quality of new solution candidates and thus enables a more efficient search.

The algorithms have been tested using four different problem sets. The results suggest that the collaborative algorithms have characteristics to improve solutions, consistency, and convergence with the same number of heavy function evaluations f_h compared to traditional search methods and an MTO algorithm. The information sharing contributes the most during early stages of the optimization when objective function sampling for the various problems is spread across the search space. At later stages, when the different search processes start converging towards their individual best solutions, information sharing from similar problems ("neighboring" problems) becomes most helpful. The collaborative feature entails additional computing time. However, this additional run time is negligible when the heavy function is much more expensive to compute than the light functions.

8. Future work

The collaborative optimization framework presented in this article could have many potential applications and underlie some algorithm developments or improvements. In this section we mention several potential studies with the basis of collaborative optimization algorithms. Firstly, special MTO cases with characteristics described in Section 2 may appear in various disciplines, including engineering, business, and healthcare, as simulation-based optimization technique is common for these disciplines. Some low hanging fruits for future studies are to have some real-world applications of the proposed collaborative algorithms. Performance comparisons of the collaborative algorithms against traditional optimization methods or MTO algorithms when solving for the real-world problems are relevant for future studies.

Secondly, future studies can consider the developments of new collaborative algorithms (besides the one presented in this article). For example by adopting the collaborative concepts into other traditional optimization methods, like pattern search or Bayesian optimization. Thirdly, future studies can explore and investigate some potential improvements for the collaborative optimization framework. One of the improvements could come from an idea of selective-and-adaptive collaboration. The term "selective" means that the transfer of information is carried out only between similar tasks or problems because the information sharing between similar problems is found most helpful, particularly at later stages. The term "adaptive" implies to keep updating the similarity of the tasks along the search processes. For this we can utilize the source of "useful information" data (as in Fig. 6) or other measures explained in [8,37]. Next, as discussed in Section 5, the way of sharing simulation results (e.g., C-GD-I versus C-GD-II) affects the performance of collaborative algorithms. The sharing mechanism can thus be considered as room for improvements. Making the collaborative operation more efficient is also an interesting topic for future studies. For instance, we can transform the operation for deciding the best swap scenario in C-GD-II (see Algorithm 7) into a combinatorial problem. Moreover, extensions into asynchronous versions of the collaborative methods can speed up the optimization if the heavy calculations have significantly different run times.

Lastly, besides the non-dominated sorting genetic algorithm (NSGA-II) [70], one approach for finding Pareto-optimal solutions of an MOO problem is by converting the MOO problem into a set of SOO problems (e.g., by methods proposed in [5,6]), and then solving those problems one at a time. In case the SOO problems have properties as described in Section 2, the collaborative algorithms may serve as an alternative to the NSGA-II. Comparisons between this alternative approach against the NSGA-II or other MTO algorithms are interesting to look into in future studies.

CRediT authorship contribution statement

I Gusti Agung Gede Angga: Methodology, Investigation, Software, Visualization, Writing – original draft. **Mathias Bellout:** Validation, Formal analysis, Writing – original draft. **Per Eirik Strand Bergmo:** Formal analysis, Writing – review & editing, Project administration, Funding acquisition. **Per Arne Slotte:** Validation, Formal analysis, Writing – review & editing. **Carl Fredrik Berg:** Conceptualization, Writing – original draft, Supervision.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: I Gusti Agung Gede Angga reports financial support was provided by the Research Council of Norway.

Acknowledgment

The first author, I Gusti Agung Gede Angga, was supported by the Research Council of Norway through its Research Centre for Petroleum (PETROSENTER) program, project number 296207, LowEmission.

Appendix A. Code availability

Python scripts of the collaborative algorithms presented in this article are available at the GitHub page of Petroleum Cybernetics Group NTNU.[1]

[1] https://github.com/PetroleumCyberneticsGroup/Materials.

References

[1] Osaba E, Martinez AD, Del Ser J. Evolutionary multitask optimization: A methodological overview, challenges and future research directions. 2021.

[2] Gupta A, Ong Y-S. Genetic transfer or population diversification? Deciphering the secret ingredients of evolutionary multitask optimization. In: IEEE symposium series on computational intelligence (SSCI). 2016, p. 1–7.

[3] Gupta A, Ong Y-S, Feng L. Multifactorial evolution: Toward evolutionary multitasking. IEEE Trans Evol Comput 2016;20(3):343–57.

[4] Gupta A, Ong Y-S, Feng L, Tan KC. Multiobjective multifactorial optimization in evolutionary multitasking. IEEE Trans Cybern 2017;47(7):1652–65.

[5] Das I, Dennis JE. Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems. SIAM J Optim 1998;8(3):631–57.

[6] Utyuzhnikov SV, Fantini P, Guenov MD. A method for generating a well-distributed Pareto set in nonlinear multiobjective optimization. J Comput Appl Math 2009;223(2):820–41.

[7] Zheng X, Lei Y, Qin AK, Zhou D, Shi J, Gong M. Differential evolutionary multitask optimization. In: IEEE congress on evolutionary computation (CEC). 2019, p. 1914–21.

[8] Gupta A, Ong YS, Da B, Feng L, Handoko SD. Measuring complementarity between function landscapes in evolutionary multitasking. In: IEEE congress on evolutionary computation (CEC). 2016.

[9] Tang Z, Gong M, Jiang F, Li H, Wu Y. Multipopulation optimization for multitask optimization. In: IEEE congress on evolutionary computation (CEC). 2019, p. 1906–13.

[10] Cheng M-Y, Gupta A, Ong Y-S, Ni Z-W. Coevolutionary multitasking for concurrent global optimization: With case studies in complex engineering design. Eng Appl Artif Intell 2017;64:13–24.

[11] Feng L, Zhou W, Zhou L, Jiang SW, Zhong JH, Da BS, Zhu ZX, Wang Y. An empirical study of multifactorial PSO and multifactorial DE. In: IEEE congress on evolutionary computation (CEC). 2017, p. 921–8.

[12] Song H, Qin AK, Tsai P-W, Liang JJ. Multitasking multi-swarm optimization. In: IEEE congress on evolutionary computation (CEC). 2019, p. 1937–44.

[13] Swersky K, Snoek J, Adams RP. Multi-task Bayesian optimization. In: Advances in neural information processing systems (NIPS), Vol. 26. 2013.

[14] Rice J, Cloninger CR, Reich T. Multifactorial inheritance with cultural transmission and assortative mating. I. Description and basic properties of the unitary models. Am J Human Genet 1978;30(6):618–43.

[15] Cloninger CR, Rice J, Reich T. Multifactorial inheritance with cultural transmission and assortative mating. II. A general model of combined polygenic and cultural inheritance. Am J Human Genet 1979;31(2):176–98.

[16] Li G, Zhang Q, Gao W. Multipopulation evolution framework for multifactorial optimization. In: Genetic and evolutionary computation conference. 2018, p. 215—216.

[17] Li G, Lin Q, Gao W. Multifactorial optimization via explicit multipopulation evolutionary framework. Inform Sci 2020;512:1555–70.

[18] Li X, Wang L, Jiang Q. Multipopulation-based multi-tasking evolutionary algorithm. Appl Intell 2022.

[19] Hashimoto R, Ishibuchi H, Masuyama N, Nojima Y. Analysis of evolutionary multi-tasking as an island model. In: Genetic and evolutionary computation conference. 2018, p. 1894—1897.

[20] Bonilla EV, Chai K, Williams C. Multi-task Gaussian process prediction. In: Advances in neural information processing systems (NIPS), Vol. 20. 2007.

[21] Shahriari B, Swersky K, Wang Z, Adams RP, de Freitas N. Taking the human out of the loop: A review of Bayesian optimization. Proc IEEE 2016;104(1):148–75.

[22] Han Z, Xu C, Zhang L, Zhang Y, Zhang K, Song W. Efficient aerodynamic shape optimization using variable-fidelity surrogate models and multilevel computational grids. Chin J Aeronaut 2020;33(1):31–47.

[23] Koziel S, Leifsson L. Multi-level CFD-based airfoil shape optimization with automated low-fidelity model selection. Procedia Comput Sci 2013;18:889–98.

[24] Anitha D, Shamili GK, Ravi Kumar P, Sabari Vihar R. Air foil shape optimization using CFD and parametrization methods. Mater Today: Proc 2018;5(2):5364–73.

[25] Baumann EJM, Dale SI, Bellout MC. FieldOpt: A powerful and effective programming framework tailored for field development optimization. Comput Geosci 2020;135:104379.

[26] Bellout MC, Echeverría Ciaurri D, Durlofsky LJ, Foss B, Kleppe J. Joint optimization of oil well placement and controls. Comput Geosci 2012;16(4):1061–79.

[27] Emerick AA, Portella RCM. Production optimization with intelligent wells. In: Latin American & caribbean petroleum engineering conference. 2007.

[28] Farajzadeh R, Kahrobaei SS, Zwart AHd, Boersma DM. Life-cycle production optimization of hydrocarbon fields: Thermoeconomics perspective. Sustain Energy Fuels 2019;3(11):3050–60.

[29] Lemarechal C. Cauchy and the gradient method. Doc Math 2012;251–4.

[30] Hooke R, Jeeves TA. "Direct search" solution of numerical and statistical problems. J ACM 1961;8(2):212–29.

[31] Holland JH. Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press; 1975.

[32] Kennedy J, Eberhart R. Particle swarm optimization. In: IEEE international conference on neural networks, Vol. 4. 1995, p. 1942–8.

[33] Shi Y, Eberhart R. A modified particle swarm optimizer. In: IEEE international conference on evolutionary computation. 1998, p. 69–73.

[34] Kushner HJ. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. J Basic Eng 1964;86(1):97–106.

[35] Jones DR, Schonlau M, Welch WJ. Efficient global optimization of expensive black-box functions. J Global Optim 1998;13(4):455–92.

[36] Bali KK, Ong Y-S, Gupta A, Tan PS. Multifactorial evolutionary algorithm with online transfer parameter estimation: MFEA-II. IEEE Trans Evol Comput 2020;24(1):69–83.

[37] Zhou L, Feng L, Zhong J, Zhu Z, Da B, Wu Z. A study of similarity measure between tasks for multifactorial evolutionary algorithm. In: Genetic and evolutionary computation conference. 2018, p. 229–30.

[38] Da B, Ong Y-S, Feng L, Qin AK, Gupta A, Zhu Z, Ting C-K, Tang K, Yao X. Evolutionary multitasking for single-objective continuous optimization: Benchmark problems, performance metric, and baseline results. 2017.

[39] Man K, Tang K, Kwong S. Genetic algorithms: Concepts and applications [in engineering design]. IEEE Trans Ind Electron 1996;43(5):519–34.

[40] Katoch S, Chauhan SS, Kumar V. A review on genetic algorithm: Past, present, and future. Multimedia Tools Appl 2021;80(5):8091–126.

[41] Ghaheri A, Shoar S, Naderan M, Hoseini SS. The applications of genetic algorithms in medicine. Oman Med J 2015;30(6):406–16.

[42] Chuang Y-C, Chen C-T, Hwang C. A real-coded genetic algorithm with a direction-based crossover operator. Inform Sci 2015;305:320–48.

[43] Zhan Z-H, Zhang J, Li Y, Shi Y-H. Orthogonal learning particle swarm optimization. IEEE Trans Evol Comput 2011;15(6):832–47.

[44] Park J-B, Jeong Y-W, Shin J-R, Lee KY. An improved particle swarm optimization for nonconvex economic dispatch problems. IEEE Trans Power Syst 2010;25(1):156–66.

[45] Yang C, Simon D. A new particle swarm optimization technique. In: International conference on systems engineering. 2005, p. 164–9.

[46] Onwunalu JE, Durlofsky LJ. Application of a particle swarm optimization algorithm for determining optimum well location and type. Comput Geosci 2010;14(1):183–98.

[47] Babazadeh A, Poorzahedy H, Nikoosokhan S. Application of particle swarm optimization to transportation network design problem. J King Saud Univ - Sci 2011;23(3):293–300.

[48] Zhou Y, Li Z, Zhou H, Li R. The application of PSO in the power grid: A review. In: Chinese control conference. 2016, p. 10061–6.

[49] Ajbar W, Parrales A, Cruz-Jacobo U, Conde-Gutiérrez RA, Bassam A, Jaramillo OA, Hernández JA. The multivariable inverse artificial neural network combined with GA and PSO to improve the performance of solar parabolic trough collector. Appl Therm Eng 2021;189:116651.

[50] Wang D, Tan D, Liu L. Particle swarm optimization algorithm: An overview. Soft Comput 2018;22(2):387–408.

[51] Clerc M, Kennedy J. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput 2002;6(1):58–73.

[52] Li C, Yang S, Nguyen TT. A self-learning particle swarm optimizer for global optimization problems. IEEE Trans Syst Man Cybern B 2012;42(3):627–46.

[53] Peram T, Veeramachaneni K, Mohan C. Fitness-distance-ratio based particle swarm optimization. In: IEEE swarm intelligence symposium. 2003, p. 174–81.

[54] Ardizzon G, Cavazzini G, Pavesi G. Adaptive acceleration coefficients for a new search diversification strategy in particle swarm optimization algorithms. Inform Sci 2015;299:337–78.

[55] Roy R, Ghoshal SP. A novel crazy swarm optimized economic load dispatch for various types of cost functions. Int J Electr Power Energy Syst 2008;30(4):242–53.

[56] Freitas D, Lopes LG, Morgado-Dias F. Particle swarm optimisation: A historical review up to the current developments. Entropy 2020;22(3):362.

[57] Bellman R. Dynamic programming treatment of the travelling salesman problem. J ACM 1962;9(1):61–3.

[58] Held M, Karp RM. A dynamic programming approach to sequencing problems. J Soc Ind Appl Math 1962;10(1):196–210.

[59] Volgenant T, Jonker R. A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. European J Oper Res 1982;9(1):83–9.

[60] Carpaneto G, Dell'Amico M, Toth P. Exact solution of large-scale, asymmetric traveling salesman problems. ACM Trans Math Software 1995;21(4):394–409.

[61] Padberg M, Rinaldi G. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM Rev 1991;33(1):60–100.

[62] Yang X-S. Test problems in optimization. In: Yang X-S, editor. Engineering optimization: an introduction with metaheuristic applications. John Wiley & Sons; 2010.

[63] McKay MD, Beckman RJ, Conover WJ. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 1979;21(2):239–45.

[64] Angga IGAG, Bellout M, Kristoffersen BS, Bergmo PES, Slotte PA, Berg CF. Effect of CO_2 tax on energy use in oil production: Waterflooding optimization under different emission costs. 2022, (submitted for publication).

[65] Potter MA, De Jong KA. A cooperative coevolutionary approach to function optimization. In: Davidor Y, Schwefel H-P, Männer R, editors. International conference on parallel problem solving from nature. 1994, p. 249–57.
[66] Yang Z, Tang K, Yao X. Large scale evolutionary optimization using cooperative coevolution. Inform Sci 2008;178(15):2985–99.
[67] van den Bergh F, Engelbrecht A. A cooperative approach to particle swarm optimization. IEEE Trans Evol Comput 2004;8(3):225–39.
[68] Li X, Yao X. Cooperatively coevolving particle swarms for large scale optimization. IEEE Trans Evol Comput 2012;16(2):210–24.
[69] Parsopoulos KE. Cooperative micro-particle swarm optimization. In: ACM/SIGEVO summit on genetic and evolutionary computation. 2009, p. 467–74.
[70] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 2002;6(2):182–97.