J Heuristics (2017) 23:449–469
DOI 10.1007/s10732-017-9351-z

Constructive cooperative coevolution for large-scale global optimisation

Emile Glorieux[1] · **Bo Svensson**[1] · **Fredrik Danielsson**[1] · **Bengt Lennartson**[2]

Received: 25 March 2016 / Revised: 29 June 2017 / Accepted: 30 June 2017 /
Published online: 17 July 2017

Abstract This paper presents the Constructive Cooperative Coevolutionary (C^3) algorithm, applied to continuous large-scale global optimisation problems. The novelty of C^3 is that it utilises a multi-start architecture and incorporates the Cooperative Coevolutionary algorithm. The considered optimisation problem is decomposed into subproblems. An embedded optimisation algorithm optimises the subproblems separately while exchanging information to co-adapt the solutions for the subproblems. Further, C^3 includes a novel constructive heuristic that generates different feasible solutions for the entire problem and thereby expedites the search. In this work, two different versions of C^3 are evaluated on high-dimensional benchmark problems, including the CEC'2013 test suite for large-scale global optimisation. C^3 is compared with several state-of-the-art algorithms, which shows that C^3 is among the most competitive algorithms. C^3 outperforms the other algorithms for most partially separable functions and overlapping functions. This shows that C^3 is an effective algorithm for large-scale global optimisation. This paper demonstrates the enhanced performance by using constructive heuristics for generating initial feasible solutions for Cooperative Coevolutionary algorithms in a multi-start framework.

Keywords Evolutionary optimisation · Cooperative coevolution · Algorithm design and analysis · Large-scale optimisation

This work was supported in part by Västra Götalandsregionen, Sweden, under the Grant PROSAM 612-0974-14.

✉ Emile Glorieux
emile.glorieux@hv.se

[1] Department of Engineering Science, University West, S-461 86 Trollhättan, Sweden

[2] Department of Signals and Systems, Chalmers University of Technology, S-412 96 Gothenburg, Sweden

1 Introduction

Many practical real-world optimisation problems within engineering can be considered as global optimisation problems. Such problems are also often complex and large-scale (i.e. large number of parameters) which makes them difficulty to solve (Ali et al. 2005; Lozano et al. 2011). Heuristics and metaheuristics are among the existing methods to solve continuous optimisation problems.

Multi-start methods are one type of metaheuristics. These methods usually embed other optimisation algorithms, such as local search and neighbourhood search, which lack the required diversification to explore the search space globally (Grendreau and Potvin 2010). The multi-start method starts the embedded algorithm from multiple different initial solutions, often obtained by random sampling. A well-known multi-start method, the Continuous Greedy Randomised Adaptive Search Procedure (CGRASP) (Hirsch et al. 2010), is relevant for the work presented in this paper.

Another metaheuristic, that is adopted in this work, is the Cooperative Coevolutionary (CC) algorithm (Potter and De Jong 1994). This algorithm requires that the optimisation problem is decomposed into subproblems. Then, each subproblem has a group of corresponding parameters. These subproblems are optimised separately, but there is cooperation between the optimisations. The cooperation is necessary to ensure that the solutions for the subproblems are still optimal when combined into a solution for the entire problem. CC has a good performance especially on large-scale optimisation problems. Though, other works (Li and Yao 2009; Omidvar et al. 2014, 2010; Potter and De Jong 1994; Ray and Yao 2009; Yang et al. 2008) found that the CC struggles to solve non-separable problems (i.e. interactions/interdependencies between parameters). Those works propose different versions of CC to improve the performance on non-separable problems. These different versions of CC mainly focus on regrouping the parameters into different subproblems. The aim with this is to gather interacting parameters in the same subproblem and optimise them together.

This work considers a different approach to improve CC's performance, but can still be used together with earlier proposed regrouping strategies. The focus is on extending CC using the GRASP multi-start architecture. The algorithm proposed in this work is called the Constructive Cooperative Coevolutionary (C^3) algorithm. A constructive heuristic is incorporated in C^3 to efficiently find good feasible solutions. These feasible solutions are used as initial solution for CC. When the search of CC stagnates, C^3 restarts the constructive heuristic to get a new initial solution. C^3's aim is to increase the performance of CC, specifically on non-separable large-scale optimisation problems.

The main contribution of this paper is the evaluation of the performance and the robustness of the C^3 algorithm and to get insight into its behaviour towards specific characteristics of large-scale optimisation problems (modality, separability, etc.). Compared to previous work (Glorieux et al. 2014, 2015), C^3 is improved in order to make it scalable in number of subproblems. Thereby, it can be applied on a wider range of problems. C^3 is compared with other algorithms, such as the Cooperative Coevolutionary (CC) algorithm (Potter and De Jong 2000; Wiegand et al. 2001; Zamuda et al. 2008), Self-Adaptive Differential Evolution (jDErpo) (Brest et al. 2014) and Particle

Swarm Optimiser (PSO) (Nickabadi et al. 2011). This shows that C^3 is better than the other algorithms, both in terms of performance and robustness.

The remaining of this paper is organised as follows, in Sect. 2 the relevant background information for C^3 is given. In Sect. 3, the details of C^3 are presented. The implementation of the tests performed in this paper is described in Sect. 4. The results of these tests are presented and discussed in Sect. 5 and finally Sect. 6 concludes this work.

2 Background

In earlier work, C^3 is applied on practical optimisation problems concerning control of interacting production stations (Glorieux et al. 2014, 2015). There, it is used within a simulation-based optimisation framework and it is shown that C^3 outperforms other optimisation methods. The previous version of C^3 was not scalable in number of subproblems, which was a limitation of the algorithm. In this work, an improved version of C^3 is proposed for which this limitation is removed. Hence, it can be applied on a wider range of problems. In previous work on C^3 the algorithm has only been tested on problems that has up to 100 dimensions. In this work, C^3 is tested on a wide range of large-scale problems with up to 1000 dimensions. This section provides the relevant background for the design of C^3.

2.1 Greedy randomised adaptive search procedure

A well-known multi-start method (Martí et al. 2010) is the Continuous Greedy Randomised Adaptive Search Procedure (CGRASP) (Hirsch et al. 2007, 2010), which is based on the discrete GRASP (Feo and Resende 1995). For each *start* (also referred to as *iteration*), two phases are executed: a constructive phase and a local improvement phase. The constructive phase builds a feasible solution. This is done by performing a line search separately in each search direction while keeping the parameters for the other directions fixed to a random initial value. This solution is then used as an initial solution for the optimisation algorithm used in the local improvement phase. The local improvement phase terminates when the search reaches a local optimum.

Hirsch et al. (2007, 2010) evaluated the performance of CGRASP on a set of standard benchmark functions and a real world continuous optimisation problems. For some of the benchmark functions, CGRASP's performance was not as good compared to other optimisation methods (Simulated Annealing and Tabu Search). Later, an improved version of DC-GRASP was proposed by Araújo et al. (2015) that has an improved performance, especially on high-dimensional problems.

C^3 adopts CGRASP's multi-start architecture with a constructive and improvement phase for each start. The constructive phase of C^3 is different in that the subproblems are stepwise optimised (instead of a single parameter) and only the previously optimised subproblems are kept fixed and the other subproblems are not considered during the function evaluations. Another difference is that in C^3's improvement phase CC is incorporated.

2.2 Cooperative coevolutionary algorithm

The Cooperative Coevolutionary (CC) algorithm for continuous global function optimisation was first proposed by Potter and De Jong (1994). It requires that the problem is decomposed into subproblems. Typically, a natural decomposition is used that groups the D parameters into n sets, one set for each subproblem. For each subproblem, a subpopulation is initialised and optimised separately. In order to evaluate the cost of a member of a subpopulation, collaborator solutions are selected from the other subpopulations in order to form a complete solution. The combination of these collaborators is called the *context solution*. These collaborators are updated at specific intervals.

It has been proposed to use multiple collaborators from each subpopulation (Potter and De Jong 1994; Wiegand et al. 2001). When multiple collaborators are used, different combinations of these collaborators are evaluated to calculate the cost for a given subpopulation member. The function evaluation results of these different combinations are then combined into a single cost value for the subpopulation member. One of the questions when using CC is how to select the collaborators for the context solution and how many from each subpopulation.Wiegand et al. (2001) investigate the choice of the collaborator solutions, more specifically the collaborator selection pressure, the number of collaborators for a given function evaluation, and the credit assignment when using multiple collaborators. It was shown that how to select the collaborators depends on specific characteristics of the problem, especially the separability. Moreover, the selection pressure of collaborators is of less importance than the number of collaborators. Although, increasing the number of collaborators is not always preferred becomes the cost calculation because computationally more expensive.

The decomposition or grouping of the parameters influences the performance of the cooperative coevolutionary algorithm. *Random grouping* has been proposed to increase the performance on non-separable high dimensional problems (Omidvar et al. 2010; Yang et al. 2008). With random grouping, the parameters are frequently regrouped to increase the chance of having interacting parameters in the same subproblem. Omidvar et al. (2014) propose *differential grouping* to automatically uncover the underlying substructures of the problem for grouping the parameters. The parameter groups are then determined so that the interactions between the subproblems is kept to a minimum. Results show that this increases the performance significantly for non-separable problems. Ray and Yao (2009) propose to group the parameters according to their observed correlation during the optimisation. With this approach, a population of solutions for the entire problem is generated, prior to the optimisation, to determine the initial grouping. Thus, it requires an additional computational effort. Decomposition strategies for C^3 are not investigated in the work presented in this paper but is considered a topic for future studies.

CC could have the tendency to limit its search to a single neighbourhood instead of exploring the search space more (Grendreau and Potvin 2010). This behaviour has been observed especially when best solution in the subpopulation is used as collaborator. The search then convergence towards a local optimum.

Shi et al. (2005) propose using Differential Evolution (DE) instead of a genetic algorithm for the subproblem optimisations in the cooperative coevolutionary algorithm. Furthermore, an alternative static decomposition scheme is proposed in which

a subproblem takes half of the parameters. Typically, a problem is decomposed so that there is a subproblem for each parameter. The proposed algorithm (CCDE) and decomposition scheme showed an improved performance compared to DE and compared to the typical decomposition scheme. Though, the proposed algorithm was not tested on large scale non-separable problems.

Using the Particle Swarm Optimiser (PSO) for the subproblem optimisation in CC has been proposed by Bergh and Engelbrecht (2004). CCPSO has an increased performance and robustness compared to PSO, especially when the optimisation problem's dimension increases. Though, it was also noticed that the chance of converging to a suboptimum increases. Li and Yao (2012) propose a more advanced version of CCPSO that incorporates a new PSO model and a random parameter regrouping scheme with dynamic size for large scale problems up to 2000 parameters. The results showed an increased performance compared to PSO and other state-of-the-art optimisation algorithms.

3 Constructive cooperative coevolutionary algorithm

The details of the Constructive Cooperative Coevolutionary (C^3) algorithm are described in this section. In Algorithm 1 the pseudo code of C^3 is presented. C^3 is based on CGRASP's construction and adopts the multi-start architecture. Furthermore, C^3 also incorporates the CC algorithm. Hence, each *iteration* (or *start*) of C^3 includes a constructive heuristic (Phase I) and CC (Phase II).

The optimisation problem is decomposed into n subproblems. This is done by partitioning the D-dimensional set of search dimensions $G = \{1, 2, \ldots, D\}$ into n sets $G_1, \ldots, G_n$ (line 1 in Algorithm 1). The decomposition or partitioning $\mathcal{G}$ can have an influence on the performance of C^3 and is thus important. This is not investigated further in this work, but suggested as future work. The problems in this work are always decomposed randomly in equal sized subproblems.

In Phase I, the constructive heuristic builds up a feasible solution for the entire optimisation problem ($\mathbf{x}_{it,constr}$ on line 4 in Algorithm 1). A *feasible solution* is a solution that does not violate the constraints of the optimisation problem. Next, $\mathbf{x}_{it,constr}$ is used in Phase II as initial context solution for CC that further improves this solution (line 5 in Algorithm 1). Phase II is terminated when CC's search stagnates. At the next iteration, $it + 1$, the constructive heuristic (Phase I) is restarted to build up a new feasible solution. This is repeated for all iterations until the termination criteria are met. An example of a termination criterion is to limit the maximum number of function evaluations. The best solution $\mathbf{x}^*$ found over all iterations is recorded and presented as the result when the optimisation ends.

In both Phase I and II, the subproblems are optimised separately during n steps, by an embedded optimisation algorithm. To calculate the cost of the members during Phase I, the subproblems' trial solutions are assembled in a partial solution. A *partial solution* $\mathbf{p}^i$ from Step i, is a solution for the first i subproblems, with $1 \leq i \leq n$, and neglects Subproblem $i + 1$ to Subproblem n. During the function evaluation, the neglected subproblems are then also not considered. Note that calculating the cost of a partial solution is equivalent to calculating the cost of a solution for a smaller problem that only considers the parameters that are included in the partial solution.

During Phase II, to calculate the cost of a member of a subpopulation, they are assembled in a *context solution* (a solution for the entire problem) in the same way as with CC. As mentioned earlier, the context solution consists of collaborators, one from each of the other subpopulations. In this work, different collaborators are randomly chosen for each function evaluation.

The embedded optimisation algorithm must be a population-based algorithm. When it is deployed for optimising a subproblem, it initialises a (sub)population for the corresponding subproblem's parameters and evolves this subpopulation according to the procedure of the used population-based algorithm. After the optimisation, the best partial solutions in this subpopulations are used while the others are discarded. Hence, in general terms, C^3 can be combined with any suitable population-based optimisation algorithm. This is demonstrated in this work by using both an evolution algorithm (i.e. Differential Evolution) and a swarm-based algorithm (i.e. Particle Swarm Optimiser).

1: $\mathcal{G} = \{G_1, \ldots, G_n\} \leftarrow$ grouping(n)
2: $it \leftarrow 0$
3: **while** termination criteria false **do**
4: $\quad \mathbf{x}_{it,constr} \leftarrow PhaseI(\mathcal{G}, it)$
5: $\quad PhaseII(\mathcal{G}, \mathbf{x}_{it,constr})$
6: $\quad it \leftarrow it + 1$
7: **end while**

Algorithm 1: Pseudo code for Constructive Cooperative Coevolutionary (C^3) algorithm

3.1 Phase I: constructive heuristic

In Phase I, a constructive heuristic builds up a feasible solution $\mathbf{x}_{it,constr}$ for the optimisation problem in a stepwise manner, without backtracking. It includes up to n steps, one for each subproblem. The subproblem is optimised by the embedded optimisation algorithm. In Algorithm 2, the pseudo code of Phase I is presented.

In the first iteration, $it = 0$, Step 1 starts with an empty solution $\emptyset$ (line 1–5 in Algorithm 2) and optimises only Subproblem 1 starting from a randomly initialised subpopulation pop_1 ($SubOpt$ on lines 2-3 in Algorithm 2). During Step 1, the function evaluations are done on only Subproblem 1. The next subproblem then is added in the next step to initialise and optimise its subpopulation pop_i (lines 9–10 in Algorithm 2). During Step i, Subproblem 1 to Subproblem i are included, and Subproblem $i + 1$ to Subproblem n are neglected. Hence, the current parameter vector only contains the parameters of the first i subproblems and the function evaluations only take into account those subproblems. For the optimisation in each step ($SubOpt$ on line 10 in Algorithm 2), only the parameters related to the most recently added subproblem are optimised (Subproblem i in Step i). All the other included parameters are kept fixed to values of partial solution selected in the previous step. This is illustrated in Fig. 1 for the second and the third step.

Require: $\mathcal{G} = \{G_1, \ldots, G_n\}, it$
1: **if** $it = 0$ **then**
2: $pop_1 \leftarrow initialise()$
3: $\{\mathbf{p}_1^1, \ldots, \mathbf{p}_k^1\} \leftarrow SubOpt(G_1, pop_1, \emptyset)$
4: $\{\mathbf{X}^p\} \leftarrow \{\mathbf{p}_1^1, \ldots, \mathbf{p}_k^1\}$
5: **end if**
6: $(\mathbf{p}_{j_i}^i, i) \leftarrow BestUnexplored(\{\mathbf{X}^p\})$
7: **while** $i < n$ **do**
8: $i \leftarrow i + 1$
9: $pop_i \leftarrow initialise()$
10: $\{\mathbf{p}_1^i, \ldots, \mathbf{p}_k^i\} \leftarrow SubOpt(G_i, pop_i, \mathbf{p}_{j_{i-1}}^{i-1})$
11: $\{\mathbf{X}^p\} \leftarrow \{\mathbf{X}^p\} \cup \{\mathbf{p}_1^i, \ldots, \mathbf{p}_k^i\}$
12: $\mathbf{p}_{j_i}^i \leftarrow Choose(\{\mathbf{p}_1^i, \ldots, \mathbf{p}_k^i\})$
13: **end while**
14: **return** $\mathbf{p}_{j_n}^n$

Algorithm 2: Pseudo code for Phase I of C^3

At the end of the optimisation of subproblem i, the k best (partial) solutions in subpopulation pop_i are stored in $\{\mathbf{X}^p\}$ (line 11 in Algorithm 2). The purpose of the stored partially constructed solutions in $\{\mathbf{X}^p\}$ is to be further constructive in Phase I of next iterations of C^3. For the next step, one of the k stored partial solutions is randomly chosen (line 12 in Algorithm 2). In the next Step $i + 1$ of the current iteration, the parameters of Subproblem i are now kept fixed to the randomly chosen partial solution. Then, the parameters of Subproblem $i + 1$ are optimised in the same way as Subproblem i in Step i. Finally, in the last step, Step n, the found partial solution is now a solution for the entire problem since all subproblems have been added. The best one, $\mathbf{p}_{j_n}^n$, is then used as initial context solution for CC in Phase II.

When the constructive heuristic is restarted in the next iterations, it does not start constructing a new solution from scratch. Instead, it starts with the best unexplored partial solution in $\{\mathbf{X}^p\}$ (line 6 in Algorithm 2). That partial solution is then further constructed in the same way as in the first iteration. Since all stored partial solutions in $\{\mathbf{X}^p\}$ are unique and different, the constructed solutions will also all be different.

Note that the cost value of all stored partial solutions in $\{\mathbf{X}^p\}$, even though some include more parameters (subproblems) than others, is compared to selected the best one. If all subproblems have the same optimal cost value, this is possible. In the other cases, a scale factor or a heuristic estimate that compensates for the differences in the cost value between stored partial solutions from different steps can be introduced.

A constructive heuristic typically creates better feasible solutions, with the same effort (i.e. in the same number of cost calculations), compared to random sampling (Grendreau and Potvin 2010). Obviously, using better solutions as initial context solution for CC is beneficial for its convergence. The role of the constructive heuristic of Phase I is to construct a feasible solution in a greedy fashion. The greediness of the constructive heuristic comes from the fact that a single partial solution (one of the k best) is further constructed in each step. The constructive heuristic also avoids redundancy and guarantees that, in each iteration, a different feasible solution is constructed. This forces CC in Phase II to search in unexplored areas.

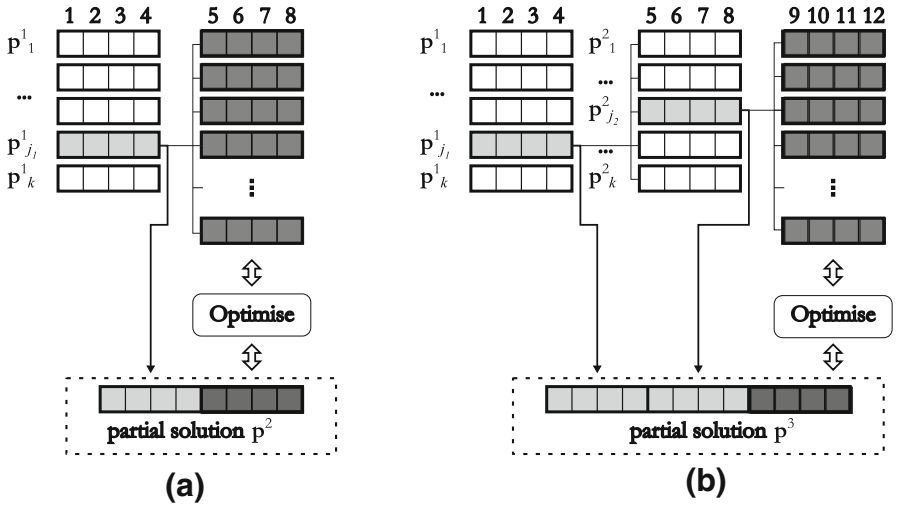

Fig. 1 Illustration of the second (**a**) and third step (**b**) of the constructive heuristic in Phase I

3.2 Phase II: cooperative coevolution

Phase II starts from the constructed feasible solution $\mathbf{x}_{it,constr}$ and searches for better solutions using CC. The pseudo code of Phase II is presented in Algorithm 3. The optimisation in Phase II is organised in cycles. In one cycle, the same subproblems as in Phase I are optimised stepwise, in a round-robin fashion. Consequently, a cycle includes n steps, one for each subproblem.

Require: $\mathcal{G} = \{G_1, \ldots, G_n\}, \mathbf{x}_{it,constr}$
$l \leftarrow 0$
repeat
  $l \leftarrow l + 1$
  $i \leftarrow 1$
  while $i < n$ **do**
    if $l = 1$ **then**
      $pop_i \leftarrow Initialise()$
      $pop_i \leftarrow SubOpt(G_i, pop_i, \mathbf{x}_{it,constr})$
    else
      $coll \leftarrow SelectCollaborators(pop_1, \ldots, pop_{i-1}, pop_{i+1}, \ldots, pop_n)$
      $pop_i \leftarrow SubOpt(G_i, pop_i, coll)$
    end if
    $i \leftarrow i + 1$
  end while
until $\frac{\left|f(\mathbf{b}^{l-1*}) - f(\mathbf{b}^{l*})\right|}{\left|f(\mathbf{b}^{l*})\right|} \leq \varepsilon$
return

Algorithm 3: Pseudo code for Phase II of C^3

Subpopulation pop_i is optimised in the corresponding Step i by the embedded optimisation algorithm (line 11 in Algorithm 3). To evaluate the cost, an individual of the subpopulation is assembled in a context solution to form a complete solution. This context solution consists of collaborator solutions that are randomly chosen from the other subpopulations (line 10 in Algorithm 3). For each function evaluation, different collaborators are randomly selected as proposed by Wiegand et al. (2001).

During the first cycle of Phase II the context solution is initially the constructed solution $\mathbf{x}_{it,constr}$ instead of collaborators from the other subpopulations (lines 6–9 in Algorithm 3). A collaborator from subpopulation pop_i of Subproblem i is used only after Step i has been completed. In other words, in Step i, the collaborators for Subproblem $i+1$ until Subproblem n are taken from $x_{it,constr}$. Only in the first cycle, the subpopulations are randomly initialised at the start of the subproblem optimisation (line 7 in Algorithm 3).

Phase II is terminated when the search stagnates because it is likely that then a local optimum is reached. When the relative difference between the best solution found during the current cycle and the best solution from the previous cycle is less than ε, Phase II is terminated. This is shown in line 15 in Algorithm 3, where $\mathbf{b}^{l*}$ is the best solution found in cycle l.

Because CC optimises the smaller subproblems separately, it is well-suited for large-scale problems. The context solution ensures that a subproblem is co-adaptively optimised, as a part of the complete problem, and not as an isolated optimisation problem. By using different collaborators in the context solution for each cost calculation, a subproblem is optimised to collaborate with the individuals of the other subpopulations and not with just a single specific context solution. On the other hand, using the constructed solution from Phase I as context solution in the first cycle ensures that the CC starts search in the region of the search space specific by this constructed solution. In each iteration, the constructed solution directs CC to search a different region of the search space.

4 Implementation

Two version of C^3, C^3jDErpo and C^3PSO, are compared with 4 other algorithms. The 6 different optimisation algorithms compared in this work are: C^3jDErpo, CCjDErpo, jDErpo, C^3PSO, CCPSO, PSO. Here, C^3jDErpo refers to C^3 where jDErpo is used as embedded algorithm to optimise the subproblems, and in the same way for CCjDErpo, C^3PSO and CCPSO. To evaluate the performance and robustness of C^3, 51 tests on large-scale benchmark functions were done for both versions of C^3. Of which, 36 are based on 12 benchmark functions (see Table 1 and Appendix 1) and each is tested with 3 different number of dimensions ($D = 100$, $D = 500$, $D = 1000$). Additionally, tests are done on the test suite of the CEC'2013 special session on Large-Scale Global Optimisation (LSGO) (Li et al. 2013).

The jDErpo algorithm used as an embedded algorithm in C^3jDErpo for the subproblem optimisation is proposed by Brest et al. (2014), and has a self-adaptive mechanism to tune the control parameters, i.e. the mutation scale factor (F) and the crossover parameter (CR). The PSO algorithm used as an embedded algorithm in C^3PSO for

Table 1 Specifications of the used benchmark functions (Jamil and Yang 2013; Li et al. 2013)

	Separable	Modality	Domain
f_{Ackley}	Yes	Multi	$[-35; 35]$
f_{Eliptic}	Yes	Uni	$[-100; 100]$
$f_{\text{Rastrigin}}$	Yes	Multi	$[-5.12; 5.12]$
f_{Sphere}	Yes	Uni	$[-10; 10]$
$f_{\text{SumOfSquares}}$	Yes	Uni	$[-10; 10]$
$f_{\text{W/Wavy}}$	Yes	Multi	$[-\pi; \pi]$
$f_{\text{Dixon\&Price}}$	No	Uni	$[-10; 10]$
$f_{\text{Rot.Ackley}}$	No	Uni	$[-35; 35]$
$f_{\text{Rot.Rastrigin}}$	No	Uni	$[-5.12; 5.12]$
$f_{\text{Rosenbrock}}$	No	Uni	$[-10; 10]$
f_{Schwefel}	No	Uni	$[-10; 10]$
f_{Griewank}	No	Multi	$[-5; 5]$

the subproblem optimisation is proposed by Nickabadi et al. (2011), and has a dynamic inertia weight to progressively increase the greediness of the search as this is beneficial for large-scale optimisation (Schutte and Groenwold 2005). The used CC algorithm for the comparison is based on Wiegand et al. (2001) and uses self-adaptation based on Zamuda et al. (2008).

The population size of standalone jDErpo is set to $NP = 100$ and of standalone PSO to $NP = 75$. For C^{3}jDErpo and CCjDErpo, the population size is set to $NP = 100$ and for C^{3}PSO and CCPSO, the population size is set to $NP = 30$. All tests are repeated 25 times to obtain reliable mean results. All repetitions are repeated independently, with random start values.

For all tests with CC and C^3, the problem decomposed randomly into 10 equal sized different subproblems ($n = 10$). During the steps of Phase I, only the parameters of the subproblems that are included so far, are used to calculate the cost. For example for $D = 100$, in the first step, the dimension for the function evaluation is 10, in the second step it becomes 20, in the third step 30, and so on until in the last step it finally becomes 100.

The termination criteria for the optimisation is 3E+6 function evaluations. The stop criterion for a subproblem optimisation in a step of C^3 and CC is 60,000 function evaluations. Consequently, in C^3 this allows up to five iterations ($it \leq 5$), depending on how many cycles before Phase II stagnates in each iteration. The predefined value ε, that is used to detect when the search of Phase II stagnates to start the next iteration, was set to ($\varepsilon = 1\text{E}{-6}$). The number of stored partial solutions during each step of Phase I was set to 15 ($k = 15$), to be able to construct more than enough different solutions.

The 3 different versions (i.e. C^3, CC, and stand-alone) are pairwise compared for each specific benchmark function and dimension using the Wilcoxon signed-rank test, which is a non-parametric test as recommended by García et al. (2009). The null hypothesis is that there is no significant difference (i.e. belong to same distribution) and is rejected when the p-value is smaller than the significance level of $\alpha = 0.05$.

5 Results and discussion

In this section, the results of the performed tests in this work are presented and the relevant aspects of the results are highlighted and discussed. For simplicity, in this section C^3 is used as a collective name for C^3jDErpo and C^3PSO, and CC for CCjDErpo and CCPSO, and the "stand-alone algorithms" for jDErpo and PSO.

5.1 Convergence analysis

The evaluation of C^3's convergence performance is presented in this section. The main indicator for this is the cost of the best solution found after all $3E + 6$ function evaluations. The results are shown in Table 2. These are the mean of 25 independent repetitions. When there is a significant difference between the algorithms according to the pairwise comparison using the Wilcoxon signed-rank test, the best result(s) are highlighted in bold font in Table 2.

It can be seen that C^3 has a better convergence for the majority of the tests compared to CC and the stand-alone algorithms. Considering the statistically significant differences, C^3jDErpo is the best algorithm or among the best in 28 of the 36 tests compared to CCjDErpo and jDErpo, and C^3PSO in 23 of the 36 tests compared to CCPSO and PSO. Furthermore, the pairwise comparison showed that C^3jDErpo performs better than CCjDErpo in 20 of the 36 tests, and similar in 10. C^3PSO performs better in 20 of the 36 tests, and similar in 5 tests. There is also no drastic deterioration in convergence performance when the number of dimensions increases. It can be concluded that in general there is a benefit of using C^3 instead of CC because it either performs better or at least similar. It must be noted that C^3's convergence performance is better than CC on the non-separable functions, except for $f_{\text{Rosenbrock}}$. This indicates that C^3 struggles less with this type of optimisation problems.

The pairwise comparison between C^3 and the standalone algorithms showed that C^3jDErpo performs significantly better in 33 of the 36 tests compared with jDErpo, and C^3PSO performs significantly better in 31 of the 36 tests. It can be concluded that C^3 performs better on these large-scale problems compared to the stand-alone algorithms.

It can be concluded that in general C^3jDErpo shows the best convergence performance compared with C^3PSO. The same is true for CCjDErpo and CCPSO, and also when comparing jDErpo and PSO. Note that the embedded optimisation algorithm for the subproblem optimisations has a significant influence on the convergence of C^3. If the subproblems have very different characteristics, it might be valuable to even consider different optimisation algorithms for specific subproblems.

5.2 Computational effort

The difference in computation effort of C^3, CC and a stand-alone algorithm is analysed. This was done by recording the optimisation time on the Rosenbrock benchmark function ($f_{\text{Rosenbrock}}$) and with jDErpo. The results of this are presented in Table 3. Each test was repeated 25 times on the same computer. The specific time values differ

Table 2 Performance results of different algorithms on the 12 benchmark functions (when there is a significant difference, the best result is highlighted in bold)

	D	C^{3}jDErpo	CCjDErpo	jDErpo	C^{3}PSO	CCPSO	PSO
f_{Ackley}	100	**6.5E−19**	3.4E−15	1.7E−1	**1.0E−13**	9.4E+0	1.4E−1
	500	**6.6E−15**	2.9E+0	5.4E+0	**1.4E+0**	1.8E+1	1.9E+1
	1000	**2.4E−10**	6.6E+0	8.4E+0	**4.9E+0**	1.8E+1	1.9E+1
f_{Elliptic}	100	**0.0E+0**	**0.0E+0**	1.8E−22	**3.9E−25**	2.7E−22	1.2E−11
	500	2.3E−22	**3.4E−23**	4.5E−3	**9.2E−20**	2.9E−18	1.8E+5
	1000	4.3E−14	**6.2E−19**	3.9E+7	**1.3E−9**	2.8E−9	2.2E+7
$f_{\text{Rastrigin}}$	100	**0.0E+0**	**0.0E+0**	5.4E+1	**4.8E+1**	7.8E+1	2.5E+2
	500	**7.8E+1**	8.6E+1	6.1E+2	**1.4E+3**	1.5E+4	**1.4E+3**
	1000	**4.7E+2**	5.2E+2	1.4E+3	3.6E+3	**3.1E+3**	3.9E+3
f_{Sphere}	100	**0.0E+0**	**0.0E+0**	4.1E−29	**4.4E−31**	3.1E−29	1.1E−17
	500	**2.3E−28**	**1.0E−29**	7.2E−9	4.0E−16	**5.4E−18**	3.8E+0
	1000	2.6E−18	**1.3E−25**	8.9E+1	1.5E−3	**4.6E−15**	1.3E+3
$f_{\text{SumOfSquares}}$	100	**0.0E+0**	**0.0E+0**	2.1E−27	**8.5E−30**	2.1E−27	1.9E−14
	500	**1.0E−26**	**4.5E−27**	6.6E−6	**8.7E−13**	**1.1E−16**	3.7E+1
	1000	1.1E−13	**1.4E−22**	3.1E+4	3.1E−1	**4.4E−8**	2.9E+4
$f_{\text{W/Wavy}}$	100	**0.0E+0**	5.1E−17	6.3E−2	**3.2E−2**	7.4E−2	1.6E−1
	500	**1.5E−2**	2.0E−2	2.2E−1	1.7E−1	1.7E−1	**1.3E−1**
	1000	**5.1E−2**	6.5E−2	2.8E−1	**1.5E−1**	**1.5E−1**	2.8E−1
$f_{\text{DixonPrice}}$	100	**1.1E+0**	4.8E+3	**6.7E−1**	1.6E+1	2.2E+4	**1.2E+0**
	500	**2.0E+2**	4.6E+2	6.9E+2	**4.9E+2**	3.9E+4	6.6E+2
	1000	2.5E+3	**2.3E+3**	2.6E+6	5.1E+3	**2.2E+3**	1.4E+9
$f_{\text{Rot.Ackley}}$	100	**0.0E+0**	3.3E+3	4.3E+3	**9.7E+2**	3.0E+3	2.3E+3
	500	**3.2E+3**	1.6E+4	1.7E+4	**1.2E+4**	1.7E+4	1.3E+4
	1000	**8.3E+3**	3.4E+4	3.7E+4	**3.2E+4**	4.6E+4	2.7E+4
$f_{\text{Rot.Rastrigin}}$	100	**1.1E−14**	2.1E+1	2.0E+1	**8.4E−1**	2.1E+1	2.0E+1
	500	**8.8E−5**	2.2E+1	2.1E+1	**2.1E+1**	2.2E+1	2.1E+1
	1000	**1.1E+0**	2.2E+1	2.1E+1	**2.2E+1**	2.2E+1	2.2E+1
$f_{\text{Rosenbrock}}$	100	8.8E+1	1.3E+2	**2.1E+1**	1.2E+2	2.3E+2	**3.5E+1**
	500	**7.5E+2**	9.6E+2	1.2E+3	1.0E+3	**7.5E+2**	1.1E+3
	1000	2.4E+3	**2.1E+3**	1.2E+5	3.2E+3	**1.8E+3**	1.5E+8
$f_{\text{Schwefels}}$	100	**1.5E−13**	8.2E+3	2.3E−7	**3.0E−29**	7.0E+3	1.4E+1
	500	**1.5E+2**	1.4E+5	**1.5E+2**	**5.4E+2**	9.2E+4	9.4E+3
	1000	**2.4E+3**	2.8E+5	7.9E+5	**1.0E+4**	2.9E+5	5.5E+4
f_{Griewank}	100	4.5E−19	4.0E−19	3.0E−4	1.6E−2	**6.0E−15**	3.8E−3
	500	**1.3E−18**	**5.7E−18**	2.6E−10	4.8E−3	**3.1E−18**	6.6E−3
	1000	**4.8E−17**	**4.8E−17**	5.5E−2	2.6E−3	**6.6E−17**	5.0E−2

for different problems but the relations between the times for C^{3}jDErpo, CCjDErpo and jDErpo will remain the same. The results show that C^{3}jDErpo's optimisation time is the shortest, and jDErpo's is the longest. It can be assumed that C^{3}jDErpo's

Table 3 Average CPU times for 3E+6 FEs on $f_{\text{Rosenbrock}}$ ($D = 1000$)

	C^3jDErpo	CCjDErpo	jDErpo
Time (s)	57.12	60.80	74.48

and CCjDErpo's shorter optimisation times, compared to jDErpo, is due to separately optimising smaller subproblems. Furthermore, the difference between C^3jDErpo and CCjDErpo is assumingly due to only considering a subset of subproblems during the steps of Phase I.

5.3 Robustness analysis

The robustness of C^3 has also been analysed and compared with the other optimisation algorithms CC, jDErpo and PSO. In this analysis, the term *robustness* implies that the algorithm succeeds in repeatedly finding a solution that is of a certain expected quality, as by Bergh and Engelbrecht (2004). Hence, a robust algorithm manages to consistently find high quality solutions. The tests were done on the same 12 benchmark functions as before with dimension $D = 100$. The required quality of the solutions was set to a cost of maximum 10^{-9}. In Table 4, the number of successful repetitions is shown for each algorithm. Each test was repeated 25 times.

On the 6 separable functions (f_{Ackley}, f_{Elliptic}, $f_{\text{Rastrigin}}$, f_{Sphere}, $f_{\text{SumOfSquares}}$, $f_{\text{W/Wavy}}$), the robustness of C^3jDErpo and CCjDErpo are similar. Both C^3jDErpo and CCfDErpo are successful for all 25 repetitions on these 6 functions. Whereas jDErpo is less robust because this is the case on only 3 of the 6 separable functions. The same behaviour can also be seen when comparing the results of C^3PSO, CCPSO and PSO. The robustness is very similar for C^3PSO and CCPSO, whereas PSO is less robust.

Table 4 Robustness results for $D = 100$

	C^3jDErpo	CCjDErpo	jDErpo	C^3PSO	CCPSO	PSO
f_{Ackley}	25	25	22	25	0	18
f_{Elliptic}	25	25	25	25	25	23
$f_{\text{Rastrigin}}$	25	25	0	0	0	0
f_{Sphere}	25	25	25	25	25	25
$f_{\text{SumOfSquares}}$	25	25	25	25	25	25
$f_{\text{W/Wavy}}$	25	25	0	0	0	0
$f_{\text{DixonPrice}}$	0	0	0	0	0	0
$f_{\text{Rot.Ackley}}$	25	0	0	0	0	0
$f_{\text{Rot.Rastrigin}}$	25	0	0	23	0	0
$f_{\text{Rosenbrock}}$	0	0	11	0	0	0
$f_{\text{Schwefels}}$	25	0	0	25	0	0
f_{Griewank}	25	25	25	1	25	17

On the 6 non-separable functions ($f_{\text{DixonPrice}}$, $f_{\text{Rot.Ackley}}$, $f_{\text{Rot.Rastrigin}}$, $f_{\text{Rosenbrock}}$, $f_{\text{Schwefels}}$, f_{Griewank}), the results show that C^{3}jDErpo is more robust compared to CCjDErpo, and to jDErpo. C^{3}jDErpo is successful for all 25 repetitions on 4 of the 5 non-separable functions. Whereas CCjDErpo has 25 successful repetitions on just a 1 of these 6 functions and on the other 5 functions, all repetitions are unsuccessful. The difference between C^{3}jDErpo and CCjDErpo is interesting because it is known that CC struggles to optimise non-separable problems, and these results indicate that non-separable problems are less problematic for C^3. For jDErpo, the robustness is less compared to C^{3}jDErpo, and interestingly slightly better compared to CCjDErpo. Again, the same can be seen when comparing the results of C^{3}PSO, CCPSO and PSO on the non-separable functions, although smaller differences.

5.4 Results CEC'2013 LSGO functions

The two version of C^3 was also evaluated on the test suite proposed on CEC'2013 special session on Large-Scale Global Optimisation (LSGO) (Li et al. 2013). This test suite consists of 15 functions, each function has 1000 dimensions ($D = 1000$). The same user-settings for the C^3 were used, i.e. 3E+6 function evaluations, $n = 10$, $\varepsilon =$ 1E−6, $k = 15$, $NP = 100$ for C^{3}jDErpo and $NP = 30$ for C^{3}PSO. Again each tests was repeated 25 times. The results of the tests with C^{3}jDErpo and C^{3}PSO are given in Table 6 in Appendix 6.1.

These results were compared with 9 other large-scale global optimisation algorithms representing the state-of-the-art, next to the previously used CCjDErpo, jDErpo, CCPSO and PSO algorithms. These included the following algorithms: MOS (LaTorre et al. 2013), IHDELS (Molina and Herrera 2015), CC-CMA-ES (Liu and Tang 2013), DECC-G (Yang et al. 2008), VMODE (López et al. 2015), MPS-CMA-ES (Bolufe-Rohler et al. 2015), jDEsps (Brest et al. 2012), FBG-CMA-CC (Liu et al. 2015), DECC-DG (Omidvar et al. 2014). The results for these 9 algorithms were taken from literature (LaTorre et al. 2015; López et al. 2015; Bolufe-Rohler et al. 2015; Liu et al. 2015)

The algorithms were ranked based on their reported mean performance for each one of the 15 benchmark functions in the CEC'2013 LSGO test suite and an overall ranking based on the average rank across the 15 functions was consequently calculated. The results of the ranking are given in Table 5.

Table 5 shows that the one of C^3 algorithms is the highest ranked algorithm for 5 of the 15 benchmark functions (f_4, f_5, f_8, f_9, f_{13}). Furthermore, C^{3}jDErpo is ranked 4th overall and C^{3}PSO is ranked 6th overall. Both C^3 algorithms are thus in the top 6 of the 15 algorithms. This shows that C^3 is a competitive algorithm, with respect to these other algorithms representing the state-of-the-art. It can thus be said that the proposed C^3 algorithm is effective for solving large-scale global optimisation problems.

The C^3 algorithms are high ranked specifically for the partially additively separable functions ($f_4 - f_9$), the overlapping functions ($f_{12} - f_{14}$) and the non-separable function (f_{15}). This indicates that C^3 is effective, in respect to the other algorithms, on all functions except the fully-separable ones. This confirms the conclusion from the tests presented in Sect. 5.1.

Table 5 Algorithm ranking on the 15 benchmark functions of the CEC'2013 test suite for large-scale global optimisation with $D = 1000$ (Li et al. 2013)

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	Overall
C^3jDErpo	10	10	10	2	1	2	2	1	1	12	12	7	1	2	5	4
C^3PSO	12	9	14	1	6	4	5	2	11	14	2	10	4	4	7	6
CCjDErpo	4	2	11	13	11	14	13	13	14	13	13	8	13	13	13	13
CCPSO	8	8	13	14	10	15	14	14	15	15	14	9	14	14	15	14
jDErpo	15	15	12	15	12	12	15	15	13	9	15	15	15	15	14	15
PSO	14	13	15	6	9	13	6	8	12	10	8	13	7	7	8	12
MOS	1	4	3	3	7	8	1	5	7	3	5	1	2	3	1	1
IHDELS	2	6	9	4	8	6	3	4	9	11	4	2	3	1	2	3
CC-CMA-ES	9	7	1	7	15	11	7	10	6	5	6	6	8	8	9	8
DECC-G	6	5	4	10	13	5	11	11	8	7	11	5	12	11	11	10
VMODE	11	12	6	9	14	9	8	9	10	6	7	11	6	5	6	9
MPS-CMA-ES	5	11	7	5	2	3	4	7	3	4	3	4	5	6	3	2
jDEsps	3	1	2	8	4	1	12	6	4	2	10	3	11	10	10	5
FBG-CMA-CC	7	3	5	12	3	10	9	3	2	8	1	12	10	12	12	7
DECC-DG	13	14	8	11	5	7	10	12	5	1	9	14	9	9	4	11

6 Conclusions and future work

The Constructive Cooperative Coevolutionary (C^3) algorithm for global optimisation of bound-constrained large-scale global optimisation problems is presented in this paper. C^3 includes a novel constructive heuristic combined with the Cooperative Coevolutionary (CC) algorithm in a multi-start architecture. For each restart, a new good initial solution is created by the constructive heuristic. The region in the search space around the constructed solution is then explored by using it as initial solution for CC. The constructive heuristic ensures that a different solution is constructed for each restart. Thereby, it drives CC to search specific regions of the search space.

C^3 was compared with state-of-the-art algorithms on a set of large-scale benchmark functions with up to 1000 dimensions, and on the test suite of CEC'2013 competition on large-scale global optimisation (Li et al. 2013). For the latter, 15 algorithms (including two versions of C^3) were compared on the 15 benchmark functions of the CEC'2013 test suite. The latter shows that a C^3 algorithm is highest ranked for 5 of the 15 benchmark functions, outperforming the top algorithms from the most recent CEC'2015 competition on large-scale global optimisation.

Based on the overall ranking across all benchmark function, the two proposed C^3 algorithms are in the top 6 out of 15 algorithms (i.e. C^3jDErpo is 4th and C^3PSO is 6th). The results also showed that C^3 outperforms the other algorithms on the partially separable functions and the overlapping functions. Results also showed that there is no extra computational cost with C^3. It can thus be concluded that C^3 is a competitive effective algorithm for large-scale global optimisation.

It was demonstrated that C^3 can be embedded with different population-based optimisation algorithms for the subproblem optimisation. Results showed that the embedded algorithm can significantly influence C^3's performance. Hence, it is important to select an optimisation algorithm that is well-suited for the specific subproblems at hand.

Future work with C^3 should investigate whether it is rewarding to use automatic decomposition strategies (i.e. parameter grouping), instead of a static decomposition as used in this work. An *adaptive* or *dynamic* decomposition strategy would be preferable in order to adjust the decomposition during the search. This could further improve the performance and abilities of the C^3 algorithm.

Appendix A

6.1 Benchmark functions

Ackley function

$$f_{\text{Ackley}}(\mathbf{x}) = -20\mathrm{e}^{-0.02\sqrt{D^{-1}\sum_{i=1}^{D} x_i^2}} - \mathrm{e}^{D^{-1}\sum_{i=1}^{D}\cos(2\pi x_i)} + 20 + \mathrm{e}$$

where the global minimum is located at $\mathbf{x}^* = f(0,\ldots,0)$, $f(\mathbf{x}^*) = 0$.
Elliptic function

$$f_{\text{Elliptic}}(\mathbf{x}) = \sum_{i=1}^{D} 10^{6\frac{i-1}{D-1}}$$

where the global minimum is located at $\mathbf{x}^* = f(0,\ldots,0)$, $f(\mathbf{x}^*) = 0$.
Rastrigin function

$$f_{\text{Rastrigin}}(\mathbf{x}) = 10D + \sum_{i=1}^{D}\left[x_i^2 - 10\cos(2\pi x_i)\right]$$

where the global minimum is located at $\mathbf{x}^* = f(0,\ldots,0)$, $f(\mathbf{x}^*) = 0$.
Sphere function

$$f_{\text{Sphere}}(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$$

where the global minimum is located at $\mathbf{x}^* = f(0,\ldots,0)$, $f(\mathbf{x}^*) = 0$.
Sum of squares function

$$f_{\text{SumOfSquares}}(\mathbf{x}) = \sum_{i=1}^{D} i x_i^2$$

where the global minimum is located at $\mathbf{x}^* = f(0,\ldots,0)$, $f(\mathbf{x}^*) = 0$.
W/Wavy function

$$f_{\text{W/Wavy}}(\mathbf{x}) = 1 - \frac{1}{D}\sum_{i=1}^{D}\cos(kx_i)\mathrm{e}^{\frac{-x_i^2}{2}}$$

where the global minimum is located at $\mathbf{x}^* = f(0,\ldots,0)$, $f(\mathbf{x}^*) = 0$. The number of local minima is k^D or $(k+1)^D$ for an odd or even value of k, respectively. In this work, k was set to the arbitrary value of 12.

Dixon & Price function

$$f_{\text{Dixon\&Price}}(\mathbf{x}) = (x_1 - 1)^2 + \sum_{i=2}^{D} i \left(2x_i^2 - x_{i-1}\right)^2$$

where the global minimum is located at $\mathbf{x}^* = f(2^{-\frac{2^i-2}{2^i}})$, $f(\mathbf{x}^*) = 0$.
Rotated Ackley function

$$f_{\text{Rot.Ackley}}(\mathbf{z}) = f_{\text{Ackley}}(R\,\mathbf{z})$$

where $\mathbf{z} = \mathbf{x} - \mathbf{x}^{opt}$, $\mathbf{x}^{opt}$ is a random shift vector, R is a random rotation matrix and a transformation function to create smooth local irregularities is applied as presented by Li et al. (2013). The global minimum is located at $\mathbf{x} = \mathbf{x}^{opt}$, $f(\mathbf{x}^{opt}) = 0$.
Rotated Rastrigin function

$$f_{\text{Rot.Rastrigin}}(\mathbf{z}) = f_{\text{Rastrigin}}(R\,\mathbf{z})$$

where $\mathbf{z} = \mathbf{x} - \mathbf{x}^{opt}$, $\mathbf{x}^{opt}$ is a random shift vector, R is a random rotation matrix and a transformation function to create smooth local irregularities is applied as presented by Li et al. (2013). The global minimum is located at $\mathbf{x} = \mathbf{x}^{opt}$, $f(\mathbf{x}^{opt}) = 0$.
Rosenbrock function

$$f_{\text{Rosenbrock}}(\mathbf{x}) = \sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$$

where the global minimum is located at $\mathbf{x}^* = f(1, \ldots, 1)$, $f(\mathbf{x}^*) = 0$.
Schwefel's Problem 1.2

$$f_{\text{Schwefel}}(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j\right)^2$$

where the global minimum is located at $\mathbf{x}^* = f(0, \ldots, 0)$, $f(\mathbf{x}^*) = 0$.
Griewank function

$$f_{\text{Griewank}}(\mathbf{x}) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

where the global minimum is located at $\mathbf{x}^* = f(0, \ldots, 0)$, $f(\mathbf{x}^*) = 0$.

Appendix B

6.2 Results CEC'13 benchmark functions

See Table 6.

Table 6 Results on the functions of the CEC'13 test suite for large-scale global optimisation with $D = 1000$ and 3E+6 function evaluations (Li et al. 2013)

	C^3jDErpo		C^3PSO	
	Mean	SD	Mean	SD
f_1	2.6E−7	7.1E−7	1.2E+3	2.0E+3
f_2	4.9E+3	3.9E+1	4.4E+3	2.2E+2
f_3	2.0E+1	1.2E−2	2.1E+1	1.0E−1
f_4	2.1E+6	6.0E+5	1.7E+6	7.7E+5
f_5	5.2E+2	3.9E+2	5.3E+6	1.6E+6
f_6	2.3E+1	5.3E+0	1.7E+4	5.8E+4
f_7	2.2E+4	6.6E+3	2.7E+5	8.3E+5
f_8	7.7E+7	2.1E+7	7.7E+7	3.6E+7
f_9	3.2E+7	3.5E+7	1.2E+9	2.1E+8
f_{10}	9.2E+7	3.9E+5	9.3E+7	7.1E+5
f_{11}	1.4E+12	1.2E+11	1.5E+6	1.4E+6
f_{12}	2.6E+3	2.6E+2	3.3E+3	3.0E+3
f_{13}	3.1E+6	7.3E+5	6.4E+6	2.3E+6
f_{14}	2.0E+7	2.8E+6	3.3E+7	1.4E+7
f_{15}	7.0E+6	5.4E+5	2.1E+7	1.9E+6

References

Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. J. Glob. Optim. **31**(4), 635–672 (2005)

Araújo, T.M.U.D., Andrade, L.M.M.S., Magno, C., Cabral, L.D.A.F., Nascimento, R.Q.D., Meneses, C.N.: DC-GRASP: directing the search on continuous-GRASP. J. Heuristics **22**(4), 365–382 (2016)

Bolufe-Rohler, A., Fiol-Gonzalez, S., Chen, S.: A minimum population search hybrid for large scale global optimization. In: Evolutionary Computation (CEC), 2015 IEEE Congress on, IEEE, pp. 1958–1965 (2015)

Brest, J., Boskovic, B., Zamuda, A., Fister, I., Maucec, M.: Self-adaptive differential evolution algorithm with a small and varying population size. In: 2012 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2012)

Brest, J., Zamuda, A., Fister, I., Boskovic, B.: Some improvements of the self-adaptive jDE algorithm. In: IEEE Symposium on Differential Evolution (SDE), pp. 1–8 (2014)

Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**(2), 109–133 (1995)

García, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. J. Heuristics **15**(6), 617–644 (2009)

Glorieux, E., Danielsson, F., Svensson, B., Lennartson, B.: Optimisation of interacting production stations using a constructive cooperative coevolutionary approach. In: IEEE International Conference on Automation Science and Engineering (CASE), pp. 322–327 (2014)

Glorieux, E., Danielsson, F., Svensson, B., Lennartson, B.: Constructive cooperative coevolutionary optimisation for interacting production stations. Int. J. Adv. Manuf. Technol. **80**(1–4), 673–688 (2015)

Grendreau, M., Potvin, J.Y. (eds.): Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 146, 2nd edn. Springer, USA (2010)

Hirsch, M.J., Meneses, C.N., Pardalos, P.M., Resende, M.G.C.: Global optimization by continuous grasp. Optim. Lett. **1**(201–212), 2 (2007)

Hirsch, M., Pardalos, P., Resende, M.: Speeding up continuous GRASP. Eur. J. Oper. Res. **205**(3), 507–521 (2010)

Jamil, M., Yang, X.: A literature survey of benchmark functions for global optimisation problems. Int. J. Math. Model. Numer. Optim. **4**(2), 150–194 (2013)

LaTorre, A., Muelas, S., Pena, JM.: Large scale global optimization: experimental results with mos-based hybrid algorithms. In: Evolutionary Computation (CEC), 2013 IEEE Congress on, IEEE, pp. 2742–2749 (2013)

LaTorre, A., Muelas, S., Peña, J.M.: A comprehensive comparison of large scale global optimizers. Inf. Sci. **316**, 517–549 (2015)

Li, X., Yao, X.: Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In: IEEE Congress on Evolutionary Computation, 2009. CEC '09, pp. 1546–1553 (2009)

Li, X., Yao, X.: Cooperatively coevolving particle swarms for large scale optimization. IEEE Trans. Evol. Comput. **16**(2), 210–224 (2012)

Li, X., Tang, K., Omidvar, MN., Yang, Z., Qin, K.: Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization. Technical report, evolutionary computation and machine learning group, RMIT University, Australia (2013)

Liu, H., Guan, S., Liu, F., Wang, Y.: Cooperative co-evolution with formula based grouping and CMA for large scale optimization. In: 2015 11th International Conference on Computational Intelligence and Security (CIS), pp. 282–285 (2015)

Liu, J., Tang, K.: Scaling up covariance matrix adaptation evolution strategy using cooperative coevolution. In: Yin, H., Tang, K., Gao, Y., Klawonn, F., Lee, M., Weise, T., Li, B., Yao, X. (eds.) Intelligent Data Engineering and Automated Learning—IDEAL 2013. Lecture Notes in Computer Science, vol. 8206, pp. 350–357. Springer, Berlin (2013)

López, E.D., Puris, A., Bello, R.R.: VMODE: a hybrid metaheuristic for the solution of large scale optimization problems. Revista Investigacion Operacional **36**(3), 232–239 (2015)

Lozano, M., Molina, D., Herrera, F.: Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. Soft Comput. **15**(11), 2085–2087 (2011)

Martí, R., Moreno-Vega, J.M., Duarte, A.: Advanced multi-start methods. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 146, pp. 265–281. Springer, New York (2010)

Molina, D., Herrera, F.: Iterative hybridization of DE with local search for the CEC'2015 special session on large scale global optimization. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 1974–1978 (2015)

Nickabadi, A., Ebadzadeh, M.M., Safabakhsh, R.: A novel particle swarm optimization algorithm with adaptive inertia weight. Appl. Soft Comput. **11**(4), 3658–3670 (2011)

Omidvar, M., Li, X., Yang, Z., Yao, X.: Cooperative co-evolution for large scale optimization through more frequent random grouping. In: IEEE Congress on Evolutionary Computation, 2010. (CEC'10), pp. 1–8 (2010)

Omidvar, M., Li, X., Mei, Y., Yao, X.: Cooperative co-evolution with differential grouping for large scale optimization. IEEE Trans. Evol. Comput. **18**(3), 378–393 (2014)

Potter, M.A., De Jong, K.A.: A cooperative coevolutionary approach to function optimization. In: Davidor, Y., Schwefel, H.P., Männer, R. (eds.) Parallel Problem Solving from Nature—PPSN III. Lecture Notes in Computer Science, vol. 866, pp. 249–257. Springer, Berlin (1994)

Potter, M.A., De Jong, K.A.: Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol. Comput. **8**(1), 1–29 (2000)

Ray, T., Yao, X.: A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning. In: IEEE Congress on Evolutionary Computation, 2009. CEC '09, pp. 983–989 (2009)

Schutte, J.F., Groenwold, A.A.: A study of global optimization using particle swarms. J. Glob. Optim. **31**(1), 93–108 (2005)

Shi, Y., Teng, H., Li, Z.: Cooperative co-evolutionary differential evolution for function optimization. In: Proc. of International Conference on Natural Computation, pp. 1080–1088 (2005)

Van den Bergh, F., Engelbrecht, A.: A cooperative approach to particle swarm optimization. IEEE Trans. Evol. Comput. **8**(3), 225–239 (2004)

Wiegand, RP., Liles, WC., Jong, KAD.: An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In: Proc. from the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, pp. 1235–1242 (2001)

Yang, Z., Tang, K., Yao, X.: Large scale evolutionary optimization using cooperative coevolution. Inf. Sci. **178**(15), 2985–2999 (2008)

Zamuda, A., Brest, J., Boskovic, B., Zumer, V.: Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution. In: Proc. Congr. Evolutionary Computation (CEC 08), pp. 3718–3725 (2008)