Large-Scale Optimization: are Co-operative Co-evolution and Fitness Inheritance Additive?

Aboubakar Hameed, David Corne
Department of Computer Science
Heriot-Watt University
United Kingdom
aah30@hw.ac.uk, dwcorne@gmail.com

David Morgan, Antony Waldock
Advanced Technology Centre, BAE Systems
Filton, Bristol
United Kingdom
antony.waldock@baesystems.com,
david.morgan11@baesystems.com

***Abstract*— Large-scale optimization - here referring mainly to problems with many design parameters - remains a serious challenge for optimization algorithms. When the problem at hand does not succumb to analytical treatment (an overwhelmingly commonplace situation), the engineering and adaptation of stochastic black box optimization methods tends to be a favoured approach, particularly the use of Evolutionary Algorithms (EAs). In this context, many approaches are currently under investigation for accelerating performance on large-scale problems, and we focus on two of those in this paper. The first is co-operative co-evolution (CC), where the strategy is to successively optimize only subsets of the design parameters at a time, keeping the remainder fixed, with an organized approach to managing and reconciling these 'subspace' optimizations. The second is fitness inheritance (FI), which is essentially a very simple surrogate model strategy, in which, with some probability, the fitness of a solution is simply guessed to be a simple function of the fitnesses of that solution's 'parents'. Both CC and FI have been found successful on nontrivial and multiple test cases, and they use fundamentally distinct strategies. In this article we explore the extent to which employing both of these strategies at once provides additional benefit. Based on experiments with 50D—1000D variants of four test functions, we find 'CCEA-FI' to be highly effective, especially when a random grouping scheme is used in the CC component.**

Keywords—large-scale optimization; fitness inheritance; co-operative co-evolution

I. INTRODUCTION

In recent years, evolutionary algorithms (EA) have been successfully and extensively used to solve many optimization problems [1]. However, we still have immense amounts to learn about how to engineer an EA to do well on a given problem class, and one of the more urgent challenges for EAs is that of *large-scale* problems. By 'large-scale', we mainly refer to optimization problems with a relatively large number of decision parameters [2]. A related challenge is that of problems where the time complexity of evaluating a single solution is high. In both cases (and in cases which combine the two), the challenge is to find strategies for evolutionary search that enable good enough solutions to be found in a smaller number of fitness evaluations than would be needed by a conventional EA. When the time-complexity of the fitness function is high, the need for progress in fewer evaluations is obvious. When the number of parameters is high, the issue tends to be that conventional EAs converge long before achieving a suitably rich exploration of the parameter space, and we must somehow achieve a better level of exploration in the available time.

Several approaches to cope with large-scale optimization problems. In this article we look in particular at two quite different strategies that are both effective. The first of these, currently under investigation within the EA community, is the strategy of co-operative co-evolution (CC). This was first posited by Potter and DeJong [3], who saw it as a 'divide and conquer strategy' which essentially decomposes a many-parameter problem into a number of fewer-parameter subproblems. The essential idea behind their 'CCEA' algorithm was to partition the parameter space into separate subproblems, and separately evolve solutions to each of these, using fixed values for the parameters not included within a particular subproblem. A sensible regime was used to occasionally update these 'fixed' values, using best so far solutions from other subproblems. In brief, a straightforward 'CC' algorithm tends to be configured to evolve subproblems in turn, each time using the best values for the previous subproblem when moving on to the next. In a sensibly configured 'CCEA', the interaction between subproblems tends to more creatively explore algorithm design space – for example, subproblems might be evolved in parallel, and there might then be brief spells of 'whole-problem' evolution, or hillclimbing, starting from the best parameters found from the subproblems so far. If we compare the convergence speed of basic EAs and basic CC, EAs may often converges faster than CC, as shown for example by on the Ackley and Sphere problems, however, sensible combinations of CC and EAs clearly prove proved more effective than individual EAs in tackling large scale problems [4].

Meanwhile, fitness inheritance (FI) is a quite different strategy first introduced by Smith et al [5], aimed at reducing need for expensive fitness function evaluations. The approach can be regarded as a particularly simple type of surrogate model based EA. In such approaches, a surrogate model of fitness is learned, which is then used a portion of the time to assign an approximate fitness value, saving the expense of a full fitness evaluation, in the hope that the accuracy of the approximation does not mislead the search. In FI, this approximation is simply to base a solution's fitness on that of its parent or parents [5,7]. Smith et al [5] defined two basic FI approaches. In the first approach, the child computes and derives its fitness value from the average fitness of its two parents. In the second approach, the approximation is instead a weighted average, accounting for the child's relative distances from each of its parents.

In this paper, we use only the first approach, and hence when we approximate a candidate solution's fitness, we just calculate the mean fitness of its parents. As we elaborate later in section III, we define and test a simple algorithm, CCEA-FI, which combines the CC and FI strategies. The remainder of this article is set out as follows. In section II we provide some more background about FI and CC, and we then describe our CCEA-FI algorithm in section III. The evaluation of CCEA-FI is dealt with in section IV, which first indicates and justifies our choice of test functions, and then describes experimental details and finishes with displays of our results. We end in section V with a discursive conclusion.

II. BACKGROUND

A. Fitness Inheritance

Smith et al [5] proposed the concept of fitness inheritance (FI), introducing two simple approaches, termed 'averaged inheritance' and 'weighted average inheritance'. In an EA that uses FI, the basic strategy uses a simple FI parameter p_I in the following way: when an evaluation of a newly generated child solution is to be done, it is either guessed using the fitness inheritance (with probability p_I), or it is evaluated accurately (and expensively) using the fitness function (with probability $1-p_I$). Smith et al [5] evaluated FI on the simple ONEMAX toy test problem, and also on a more interesting and realistic aircraft routing problem, finding impressive evidence for the value of FI. Sastry et al [8] further explored fitness inheritance on ONEMAX, theoretically investigating the optimum population size and optimum values for the inheritance proportion, and interactions between them. Their theoretical results, verified by empirical tests on ONEMAX, showed that a fitness-inheritance mechanism could provide significant efficiency enhancement for separable problems, reducing the required number of function evaluations to about half, and yielding speed-ups of between 1.75-fold and 2.25-fold. Pelikan and Sastry [9] later found even more impressive speedups obtainable through the use of FI in other contexts, echoing many other studies [10-12]. They showed in [9] that around 30-fold speedup occurred when using FI within the Bayesian Optimization Problem, tested on a number of deceptive test problems as well as ONEMAX.

Finally, we clarify the simple details involved in using FI. In our case we incorporate FI within a simple CCEA framework, as detailed next in section III. In this as well as in other frameworks, FI tends to operate as follows. The fitness values of each individual in the initial population are always evaluated accurately using the full fitness function. After this, however, all new individuals generated from parents (whether one or many parents) can be subject to fitness inheritance. Its fitness is either calculated by averaging (maybe weighted) its parent fitnesses, or its fitness is accurately evaluated using the (expensive) fitness function. The parameter p_I dictates the relevant probability. When $p_I = 0$, no FI occurs at all, and when $p_I = 1$, all fitnesses (apart from those of the initial population) are inherited. The choice of $p_I = 1$ is clearly poor, since it will just lead to convergence to copies of the best solution from the initial population. However it turns out that surprisingly high values of p_I can be very effective. In our work, echoing that of others, we explore FI values from 0.1 to 0.8, and tend to express them as percentages (e.g. from 10% to 80%). We use only the averaged inheritance variant, calculating the inherited fitness of child c from that of its parents p_1 and p_2, as simply:

$$f'(c) = \frac{f(p_1) + f(p_2)}{2} \tag{1}$$

B. Co-operative Coevolution

Potter and DeJong [3] introduced the concept of co-operative co-evolution (CC) within the EA community. Their initial work found the idea promising, particularly for separable problems with independent components. However the success of the original approach on non-separable problems is somewhat limited [14]. The original CC framework operated as in figure 1.

```
1. Decompose the decision vector into m lower-dimensional subcomponents.
2. Set i = 1 to start a new cycle.
3. Optimize the ith subcomponent with a certain EA for a predefined number of fitness evaluations (FEs).
4. If i < m then i++, and go to Step 3.
5. Stop if halting criteria are satisfied; otherwise go to Step 2 for the next cycle
```

Figure 1. The original CC framework

In attempt to improve the CC framework for non-separable problems, Yang et al [14] proposed a variant in which the groups of parameters within a subproblem adapted over time, in attempt to maximize the degree to which interacting parameters were present in the same subproblem, This led to improved performance on non-separable problems. Van den Bergh et al [15] also explored a variant of the CC approach, this in time in the context of particle swarm, optimization (PSO), in which parameters were randomly assigned to subproblems, In such random assignment, also explored in [17], each subproblem comprises a random selection of parameters scattered across the decision parameter vector, and this random assignment may change in every cycle of the algorithm. Other key design strategy choices in CC approaches include how to choose the 'template' for the parameters that are *not* included in a given subproblem, so that the fitnesses can be evaluated. Typically two approaches are used: 'best', in which this template comes from the current best so far solution, and 'random', where the template comes from random selection of candidate solutions from the corresponding other subproblem populations [6]. Finally, in [16] the question of serial v. parallel is investigated, in the context of evolving the collection of subproblems – they can either be addressed in series, each benefiting from the solution obtained previously, or in (pseudo)-parallel.

III. Integrating these strategies: CCEA-FI

We combined CC and FI into a straightforward algorithm that we call CCEA-FI, which incorporates those design aspects of both of these strategies that seem effective (from the available literature so far), while at the same time being relatively free of additional parameters or complexity. CCEA-FI uses FI with averaged inheritance, and CC with random parameter grouping. Actually we also explored non-random grouping but, in common with other recent work found random grouping significantly more effective. We use the same test functions as were used in [17], in which Ray & Yao explored an approach called CCEA-AVP (adaptive variable partitioning), in which the subpoblem parameter groupings adapted over time in attempt to minimize interactions across subproblems. In this work, Ray and Yao compared CCEA-AVP against a baseline CCEA which used a fixed, contiguous grouping of parameters into subproblems. In our work, we adopt the broad structure of Ray & Yao's baseline CCEA as our 'template' algorithm for CCEA-FI – hence using the same overall flow of control, but we use random grouping instead of fixed, contiguous grouping, standard operators, and, of course, we incorporate fitness inheritance. Figure 2 shows the overall structure of CCEA-FI, where N_S is the number of evaluations to perform within each subproblem during a cycle, *FE* refers to the maximum number of function evaluations and p_I refers to the fitness inheritance proportion. Other parameters are: population size *N*, number of parameters (problem dimension) *D*, and *K*, the number of subproblems into which the problem is partitioned. In practice, it is always ensured that *K* is a divisor of the population size *N*.

```
1. Initialize Population P
2. Evaluate all N members of P, and let B be the best
   member of the population (breaking ties randomly)
3. until a total of FE function evaluations have been done:
        3.1 Partition the D dimensions randomly into K
            subpopulations, P1, …, PD where each
            subpopulation will evolve D/K parameters
        3.2 for j in 1, …, K
                3.2.1 initialize (sub)population Pj as N
                      copies of B
                3.2.2 evolve (sub)population Pj using
                      FI with inheritance $p_I$ until $N_S$ real
                      evaluations have elapsed
                3.2.3 Update best-so-far B
        3.3 Return to 3.1
```

Figure 2. The basic structure of CCEA-FI

To complete the description, we elaborate step 3.2.2. as follows: the EA used in this part is a steady-sate, replace-worst EA with binary tournament selection, uniform crossover, and single-gene mutation (Gaussian perturbation with standard deviation of one tenth of the parameter range), and a crossover rate of 0.7. In each generation, with probability 0.7 two parents are selected and a single child emerges from crossover, otherwise a single parent is selected and the child becomes a copy of it. In either case, the child is mutated, and the resulting candidate solution is then considered for evaluation. With probability p_I its fitness is inherited (by averaging its parents' fitnesses), otherwise a full evaluation is performed.

IV. Evaluating CCEA-FI

A. Test Functions

In order to test CCEA-FI, we used the same four test functions that were used in the [17], and we use (as in [17]) their 50-dimensional and 100-dimensional variants, and perform further experiments on their 500D and 1000D variants. For convenience these functions are defined below, where *n* is the number of parameters.

The Rastrigin function (separable), with parameters x_i in the range −5.12 to 5.12:

$$f(x) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10 \right]$$

The Schwefel function (separable), with parameters x_i in the range −500 to 500:

$$f(x) = 418.9829n - \sum_{i=1}^{n} \left(x_i \sin(\sqrt{|x_i|}) \right)$$

The Rosenbrock function (non-separable), with parameters x_i in the range −30 to 30:

$$f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right]$$

The Ackley function (non-separable), with parameters x_i in the range −32 to 32:

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i) \right) + 20 + e$$

B. Further Details of Algorithm and Baseline Experiments

We used a population size of 100, and ran experiments (all repeated 20 times independently) on each of the four functions at 50D, 100D, 500D and 1000D, continuing for 100,000 evaluations (50D and 100D), 2.5M evaluations (500D) and 5,000,000 evaluations (1000D). In all of the experiments reported here, we fixed *K* at 2, and experiment with a range of FI values from 0 (plain 'CCEA') to 90%.

C. Results

Our full results are detailed at at http://is.gd/cceafi, and we provide graphical summaries of results here, showing plots for the 100D, 500D, and 1000D cases, and focusing on the relative benefits of co-evolution (CC), fitness inheritance (FI), and their combination, over the basic EA. All results plotted show the mean (and sometimes also best and worst) of 20 independent runs.

We first focus on visualizing the benefits of both CCEA-FI over both CCEA alone and the basic EA, by looking at experiments comparing EA, CCEA, and CCEA-FI on 100D versions of the functions at 100,000 (total, real) fitness evaluations. These experiments use basic 'contiguous' grouping in the CC cases – i.e. the parameters 1—50 comprised one sub-population and parameters 51—100 comprised the other. Random-grouping produced better overall results for the CC cases, but showing the results for random grouping and the EA on the same plot make it that much harder to see the difference between the CCEA-FI variants.

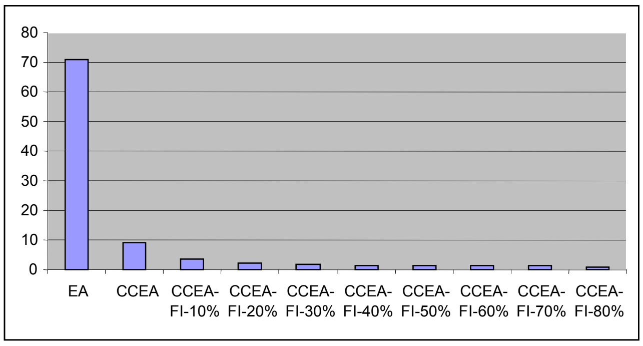

Figure 3: Mean results on Rastrigin, 100D, 10^5 evaluations, CC methods use contiguous grouping.

Figure 3 shows a typical pattern in which CCEA without FI seems a considerable improvement on the EA alone, while CCEA-FI provides further significant improvement, especially at higher rates of fitness inheritance.

The main exception to this pattern is the results on the Rosenbrock function, which we visualize in Figure 4, again for the 100D, 100,000 evaluations case Again we see the significant improvement of CCEA over EA, and further improvement as we start to introduce FI. However on the Rosenbrock problem, FI values above 50% led to very poor performance, which, if included in Figure 4, would render the other bars too small to see.

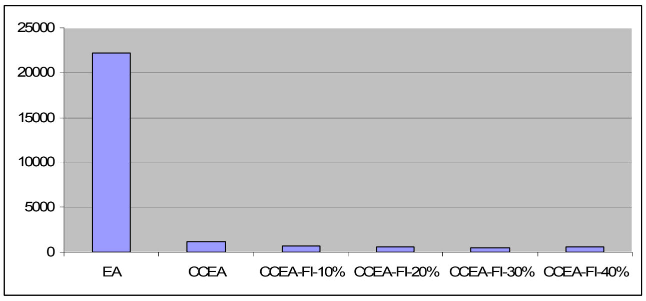

Figure 4: Mean results on Rosenbrock, 100D, 10^5 evaluations, omitting CCEA-FI with FI at 50% or above; CC methods use contiguous grouping.

In figure 5 we show corresponding results for the Ackley function. EAs in all cases, the CCEA and CCEA-FI experiments (except only the Rosenbrock function at high FI rates) were very significant improvements on the EA results; this time we omit the EA result, to enable clearer distinction between the other results.

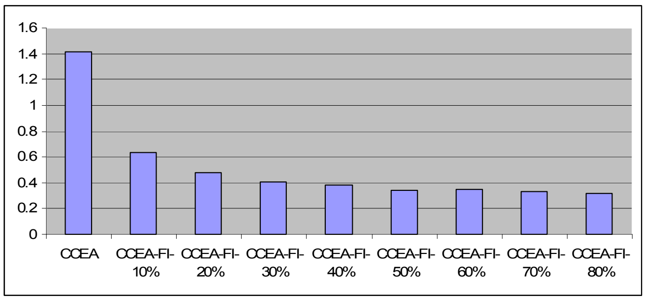

Figure 5: Mean results on Ackley, 100D, 10^5 evaluations, omitting EA results, to make comparisons among the CCEA-FI variants clearer. CC variants use contiguous grouping.

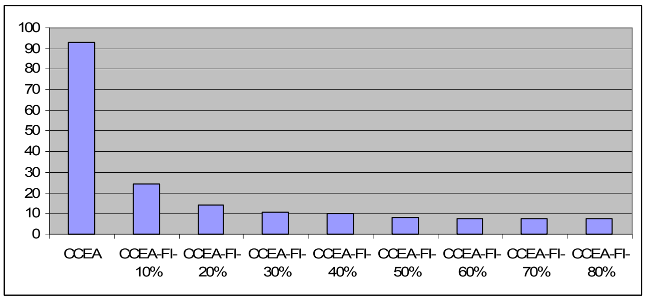

Figure 6: Mean results on Schwefel, 100D, 10^5 evaluations, omitting the EA results, to make comparisons among the CCEA-FI variants clearer.

We adopt the same approach to visualize the results on the Schwefel function in Figure 6.

We now show results using random grouping in the CC methods on the 100D functions, this time omitting the EA results, which were always considerably worse, but including the results for EA-FI (i.e. using fitness-inheritance, but not involving co-evolution). As before, we show the CCEA-FI results for a variety of FI parameters ranging between 10% and 90%, but for EA-FI we only show the single parameter that gave the best mean result in our EA-FI experiments. Hence, these plots indicate, for the 100D cases, the relative performances of CC, FI, and the CC-FI combination, with the 'basic EA' omitted but clearly understood to be considerably worse than any of the combinations shown.

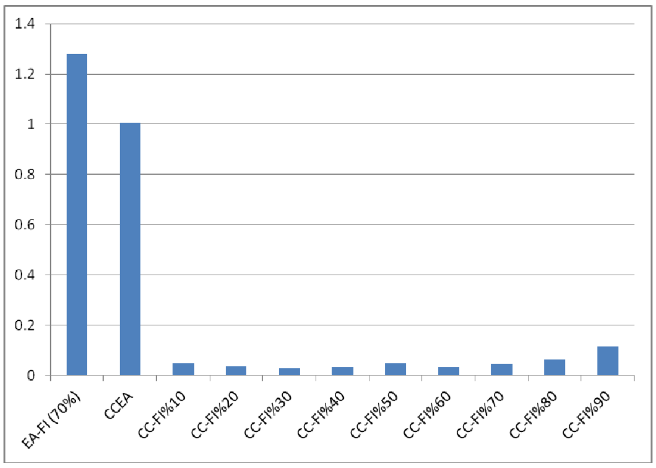

Figure 7: Mean results on Rastrigin, 100D, 10^5 evaluations, CC methods use random grouping.

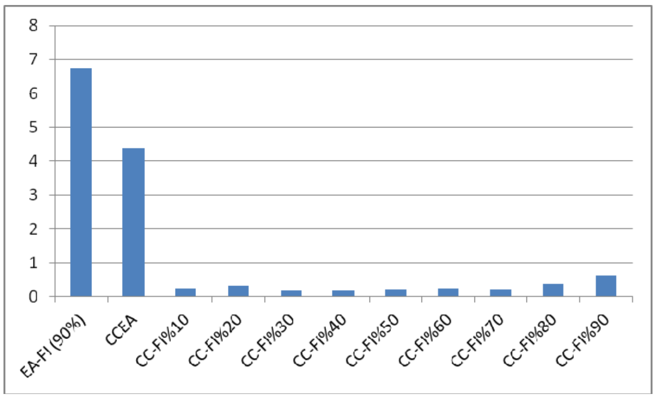

Figure 8: Mean results on Schwefel, 100D, 10^5 evaluations, CC methods use random grouping.

As we can see in Figures 7—10, the CCEA-FI combination can always yield results that are superior to either CC or FI alone. As ever, the Rosenbrock function provides some anomalous signals, but still showing CCEA-FI better than either CC or FI alone for 7 of the 9 CCEA-FI parameterisations.

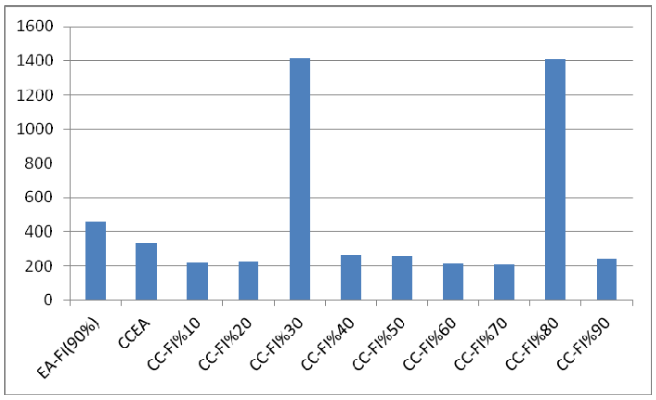

Figure 9: Mean results on Rosenbrock, 100D, 10^5 evaluations; CC methods use random grouping.

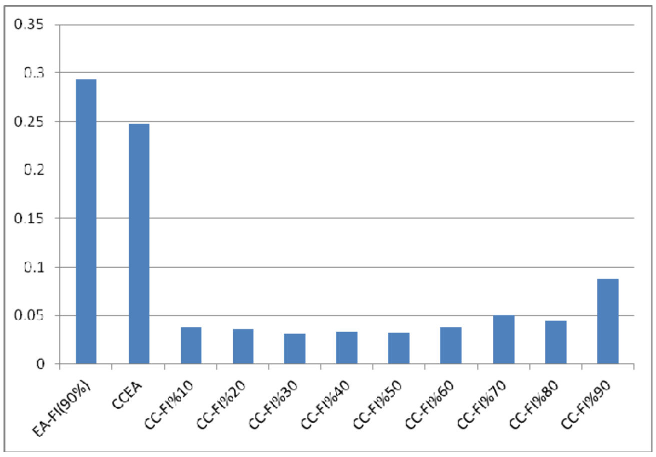

Figure 10: Mean results on Ackley, 100D, 10^5 evaluations, CC variants use random grouping.

We now turn our attention the 500D and 1000D test cases. In these cases, the better CCEA-FI results were obtained for lower percentages of fitness inheritance. So we take this opportunity to limit the presentation of CCEA-FI variants to 10%--50%, and are therefore able to fit in the 'best' and 'worst' of 20 runs in addition to the mean results. In all of these cases, the results for EA-FI (i.e. without CC) were always significantly worse than the CCEA-FI variants, and so are omitted from these plots.

As with the 100D cases, we can see that CCEA-FI variants provide better results than CCEA alone. However the Rosenbrock function continues to show distinctly unusual behavior, and we see that only the 10% case shows a better mean for CCEA-FI over CCEA, although all of the CCEA-FI cases for Rosenbrock show a better 'best of 20' result than CCEA alone. Note that the Rosenbrock plot shows the log values of the results.

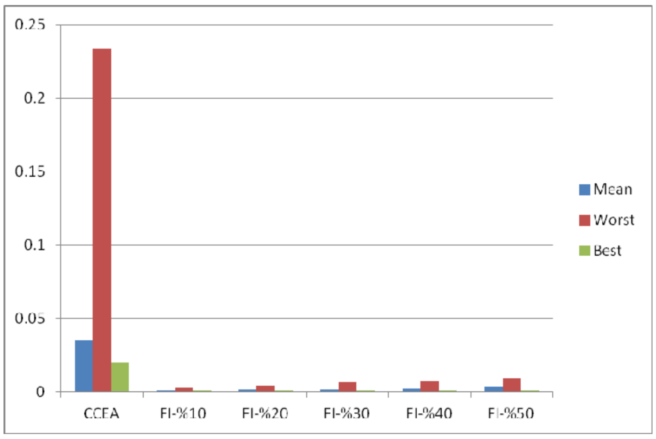

Figure 11: Best, mean and worst (of 20) results on Rastrigin, 500D, 500,000 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

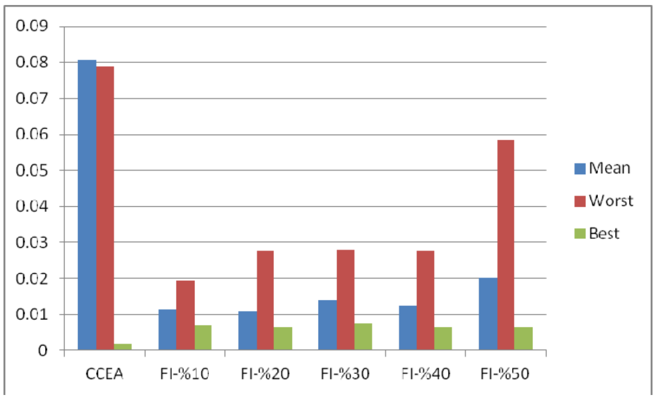

Figure 12: Best, mean and worst (of 20) results on Schwefel, 500D, 500,000 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

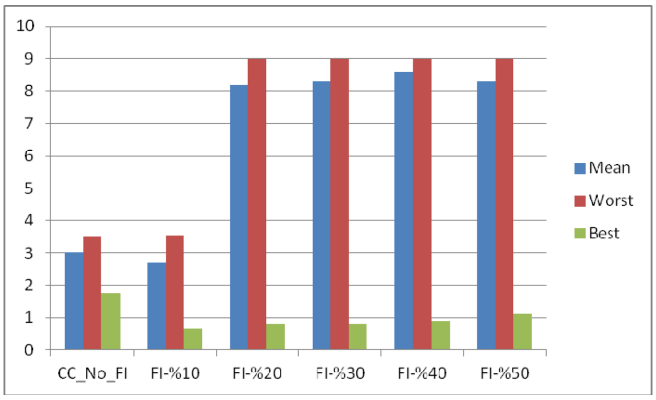

Figure 13: *log values* of best, mean and worst (of 20) results on Rosenbrock, 500D, 500,000 fitness evaluations, showing CCEA and CCEA-FI results; CC uses random grouping.

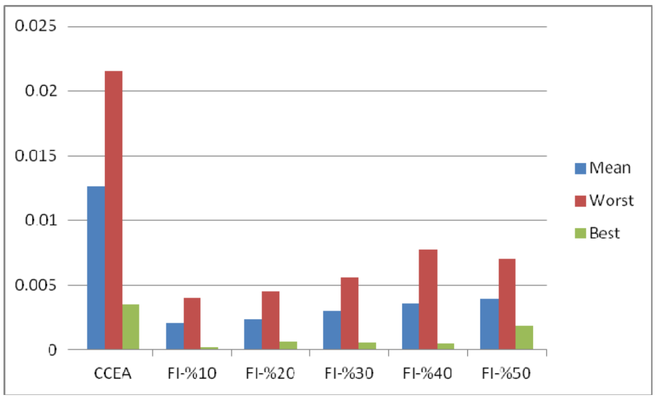

Figure 14: Best, mean and worst (of 20) results on Ackley, 500D, 500,000 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

Finally we show the results on 1000D versions of the test functions. Our display of results in these cases matches the display for 500D cases, i.e. we show only CCEA and CCEA-FI, with log values shown for the Rosenbrock function.

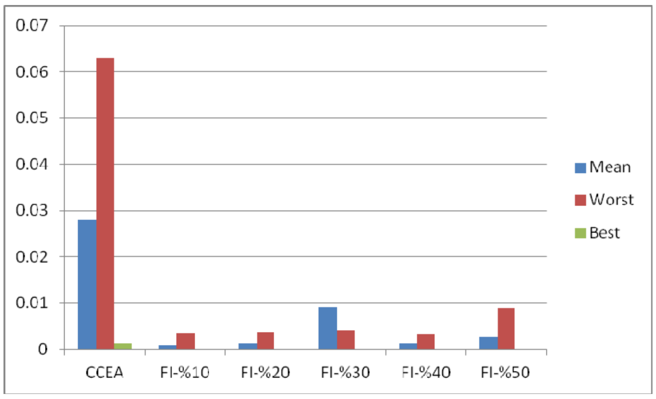

Figure 11: Best, mean and worst (of 20) results on Rastrigin, 1000D, 1,000,000 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

Again, particularly by inspection of Figures 11, 12 and 14, we can see that CCEA-FI provides considerable gains in solution quality over CCEA alone. In the Rosenbrock case, Figure 13, this is clearly the case when we look at the 'best of 20' results, indicating that CCEA-FI enables the EA to access superior results more readily than CCEA alone, but clearly with a high variance which dampens the quality of the mean performance of the CCEA-FI variants. The CCEA-FI variants with best performance on the 500D and 1000D tests are invariably those with FI percentages at 10%--30%, with 10% usually leading to the best mean result, but best 'best' result often appearing at 20% or 30%. This is in contrast to the results for 100D, where (with random grouping), better results were obtained for 30%--60% FI. We can also report (results not graphed here) that on 50D cases the best results were in the region of 60%--80% FI.

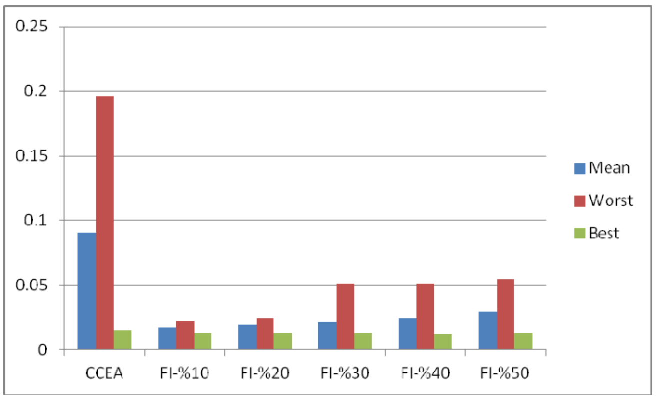

Figure 12: Best, mean and worst (of 20) results on Schwefel, 1000D, 1,000,000 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

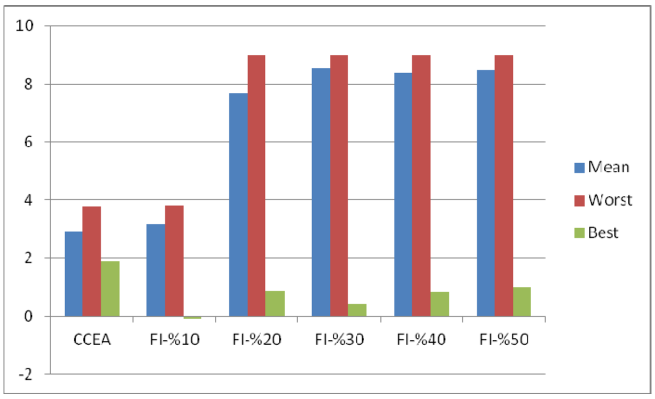

Figure 13: *log values* of best, mean and worst (of 20) results on Rosenbrock, 1000D, 1,000,000 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

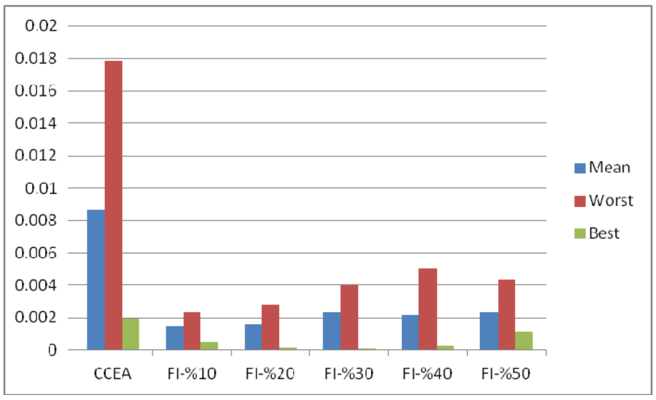

Figure 14: Best, mean and worst (of 20) results on Ackley, 1000D, 1,000,000 fitness evaluations, showing CCEA and CCEA-FI results; CC variants use random grouping.

For convenient comparison at the particularly 'large-scale' end, we also provide tables of results for the 1000D cases (tables for all experiments are provided at the site indicated below). Hence, Tables I—IV provide the best, mean and worst of 20 runs for Rastrigin, Schwefel, Rosenbrock and Ackley respectively, for the 1000D case using random grouping.

TABLE I. 1000D RESULTS ON RASTRIGIN FUNCTION – SUMMARISING 20 RUNS OF 5,000,000 FULL FITNESS EVALUATIONS

Algorithm	*Mean*	*Worst*	*Best*
CCEA	0.02804	0.062904	0.0013392
CCEA-FI-%10	0.0008792	0.0034268	0.00003939
CCEA-FI-%20	0.0012111	0.0036253	0.00005797
CCEA-FI-%30	0.009123	0.0040253	0.00004697
CCEA-FI-%40	0.001225	0.0033278	0.000061146
CCEA-FI-%50	0.0028189	0.0088574	0.000156187

TABLE II. 1000D RESULTS ON SCHWEFEL FUNCTION – SUMMARISING 20 RUNS OF 5,000,000 FULL FITNESS EVALUATIONS

Algorithm	*Mean*	*Worst*	*Best*
CCEA	0.0904649	0.1959498	0.01493262
CCEA-FI-%10	0.017106	0.0223839	0.0129554
CCEA-FI-%20	0.0189123	0.0245533	0.01307219
CCEA-FI-%30	0.0218056	0.0513977	0.0130536
CCEA-FI-%40	0.0246377	0.0513984	0.0128359
CCEA-FI-%50	0.0295285	0.0547801	0.013109008

TABLE III. 1000D RESULTS ON ROSENBROCK FUNCTION – SUMMARISING 20 RUNS OF 5,000,000 FULL FITNESS EVALUATIONS

Algorithm	*Mean*	*Worst*	*Best*
CCEA	848.39683	6345.2692	77.4156733
CCEA-FI-%10	1476.6755	6493.2428	0.78365498
CCEA-FI-%20	50000084	1.00E+09	7.438030118
CCEA-FI-%30	350001290	999999999	2.661932555
CCEA-FI-%40	250010061	999999999	7.290962858
CCEA-FI-%50	300000453	999999999	9.633148573

TABLE IV. 1000D RESULTS ON ACKLEY FUNCTION – SUMMARISING 20 RUNS OF 5,000,000 FULL FITNESS EVALUATIONS

Algorithm	*Mean*	*Worst*	*Best*
CCEA	0.0086429	0.0178052	0.00196058
CCEA-FI-%10	0.0014621	0.0023462	0.000520456
CCEA-FI-%20	0.0015707	0.0028076	0.000169317
CCEA-FI-%30	0.0023392	0.0039804	9.60644E-05
CCEA-FI-%40	0.0021775	0.0050634	0.000264814
CCEA-FI-%50	0.0023551	0.0043888	0.001101761

V. Discussion and Conclusions

The idea we explore in this article is to combine the two distinct strategies involved in co-operative co-evolution and fitness inheritance, with a view to achieving additionality. Our approach has been to design a simple algorithm that combines the two strategies (CCEA-FI) and evaluate them on the same set of test functions that were explored in a fairly recent contribution to the co-evolution literature [17]. In addition we examined performance on 500D and 1000D versions of these test functions. The raw findings indeed suggest that CCEA-FI generally achieves significantly better performance than either a CC-based EA without FI, or an EA with FI but without CC. In almost every case, the best values over 20 runs for 'best', 'worst' and 'mean' were obtained with CCEA-FI. Only the Rosenbrock function presented anomalous results, showing higher sensitivity to the fitness inheritance percentage parameter, but still generally showing better results than CCEA alone for one or more CCEA-FI parameterisation at 50D—500D, and better 'best of 20' results at 1000D.

Considering the speedup obtainable by using CCEA-FI compared to CCEA, a general inspection of the results tables (not shown here, but reported at http://is.gd/cceafi) suggests that, on both the 50D and 100D versions, the best CCEA-FI results at 50,000 function evaluations were close to the results of CCEA at 100,000 evaluations on the corresponding problem. For example, the best result recorded by CCEA on the 100D Rastrigin function at 100,000 evaluations is bested by CCEA-FI at 20% inheritance at 50,000 evaluations.

Finally, in this paper we have investigated the additionality of fitness inheritance in the context of a co-operative co-evolution algorithm, and demonstrated that this approach certainly promises further investigation. Though outperforming results in [17], however, we made no attempt in this paper to strive for best-so-far results on the problems we have looked at, or indeed compare with the current state of the art in large-scale optimization. We note that state of the art results in this context seem at the moment to arise from the use of sophisticated and self-adaptive versions of differential evolution in combination with a CC approach (e.g. [18]). Such strategies are entirely amenable to the incorporation of FI, and in current work we are exploring this idea.

Full details of the experimental results summarised here are provided at http://is.gd/cceafi .

Acknowledgment

We are grateful to the Libyan embassy in London, UK, for sponsorship of the first author's PhD studies.

References

[1] .R. Sarker, M. Mohammadian, X. Yao, Evolutionary Optimization, Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[2] R. E. Bellman. Dynamic Programming. Ser. Dover Books on Mathematics. Princeton University Press, 1957

[3] M. A. Potter and K. A. D. Jong, "A cooperative coevolutionary approach to function optimization," in PPSN III: Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature, 1994, pp. 249–257.

[4] Y. Wang, B. Li, (2009) A Self-adaptive Mixed Distribution Based Univariate Estimation of Distribution Algorithm for Large Scale Global Optimization,‖ in book: Nature-Inspired Algorithms for ptimization, in series: Studies in Computational Intelligence, Raymond Chiong(Ed), vol. 193, Chiong, Raymond (Ed.), pp. 171-198, Hardcover ISBN: 978-3-642-00266-3, Springer-Verlag.

[5] R. E. Smith, B. A. Dike, and S. A. Stegmann, "Fitness inheritance in genetic algorithms," in SAC '95: Proceedings of the 1995 ACM symposium on applied computing, pp. 345–350, ACM Press, 1995.

[6] R. P. Wiegand, W. C. Liles, and K. A. De Jong, "An empirical analysis of collaboration methods in cooperative coevolutionary algorithms," in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). San Francisco, California, USA: Morgan Kaufmann, 7-11 2001, pp. 1235–1242.

[7] S. C. Agrawal, Metamodeling: a study of approximations in queueing models. MIT Press, 1985

[8] K. Sastry, M. Pelikan, and D. E. Goldberg, "Efficiency enhancement of genetic algorithms via building-block-wise fitness estimation," in Congress on Evolutionary Computation, 2004. CEC2004., pp. 720–727,2004.

[9] M. Pelikan and K. Sastry, "Fitness inheritance in the bayesian optimization algorithm," in *Genetic and Evolutionary Computation – GECCO 2004* (K. Deb, ed.), vol. 3103 of *Lecture Notes in Computer Science*, pp. 48–59, Springer Berlin / Heidelberg, 2004.

[10] M. Reyes-Sierra and C. A. C. Coello, "Dynamic fitness inheritance proportion for multi-objective particle swarm optimization," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pp. 89–90, ACM, 2006.

[11] E. Mezura-Montes, L. Mu˜noz-D´avila, and C. A. C. Coello, "A preliminary study of fitness inheritance in evolutionary constrained optimization," in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, vol. 129 of *Studies in Computational Intelligence*, pp. 1–14, Springer, 2007.

[12] K. Sastry, D. E. Goldberg, and M. Pelikan. Don't evaluate, inherit. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt,M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke,editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 551–558, San Francisco, California,USA, 7-11 2001. Morgan Kaufmann.

[13] da Fonseca, L. G.; Lemonge, A. C. C. & Barbosa, H. J. C. (2012), A study on fitness inheritance for enhanced efficiency in real-coded genetic algorithms., *in* 'IEEE Congress on Evolutionary Computation' , IEEE, , pp. 1-8.

[14] Z. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. Information Sciences, 178:2986–2999,August 2008.

[15] F. van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. IEEE Transactions on Evolutionary Computation 8(3), pages 225–239, 2004.

[16] Popovici, Elena, and Kenneth De Jong. "Sequential versus parallel cooperative coevolutionary algorithms for optimization," in IEEE Congress on Evolutionary Computation CEC 2006, 0-0 2006, pp. 1610–1617.

[17] Ray, T. and Yao, X. , "A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning," in Proceedings of the IEEE Congress on Evolutionary Computation, (Trondheim,Norway), pp. 983– 989, 2009.

[18] Yang, Zhenyu, Ke Tang, and Xin Yao. "Large scale evolutionary optimization using cooperative coevolution." *Information Sciences* 178.15 (2008): 2985-2999.