2015 IEEE Symposium Series on Computational Intelligence

Engineering Fitness Inheritance and Co-operative Evolution into State-of-the-Art Optimizers

Aboubakar Hameed, Anna Kononova, David Corne
Dept. of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, UK
aah30@hw.ac.uk, anna.kononova@gmail.com, dwcorne@gmail.com

***Abstract*—The engineering of stochastic black box optimization methods, particularly evolutionary algorithms (EAs), represents the most common and successful approach to solve lareg-scale parameter optimization problems. Currently several strategies are being explored to improve performance when the number of parameters is large (e.g. 1000 parameters). Prominent among these techniques are variants of differential evolution, while an algorithm engineering strategy being explored is 'co-operative co-evolution' (CC), which involves successively optimizing subsets of the design parameters, with an organized approach occasionally reconciling these 'subspace' optimizations. Recent work has shown that combining CC with fitness inheritance (FI) – a technique heretofore rarely explored in the context of large-scale optimization – can reliably lead to better performance. However that work was done in the context of a simple underlying EA (allowing us to be more confident that the benefits were due primarily to the combination of CC and FI). Here we explore the extent to which CC and FI provides added value when engineered together in the context of more sophisticated, so-called state of the art underlying algorithms, pre-adorned with a variety of additional enhancements. To that end, in this paper we explore SaNSDE, and DECC-DML – two recent high-performance techniques in the field of large-scale optimization. We also explore two basic adaptive parameter setting strategies for the FI component. We find that engineering FI (and CC, where it otherwise wasn't) into these algorithms can provides either competitive or improved results (*Abstract*)**

I. INTRODUCTION

Evolutionary algorithms have been used abundantly, and usually with reported success, to solve very many real-world optimization problems [1]. However, the community as a whole still has huge amounts to learn in terms of how best to design and engineer an evolutionary algorithm (EA) in a way that maximises its chances of doing well on the problem at hand, or on a problem of a given class. Among the set of challenges that still face EA researchers, one of the more urgent ones is that of *large-scale* problems. The usual meaning of 'large-scale', in this context, is an optimization problem with a reasonably (or unreasonably) large number of decision parameters [2]. Another common meaning of 'large-scale', and a closely-related challenge, refers to problems where the processor time requirement for evaluating a single solution is very high. In both meanings of 'large-scale' (and also in cases which combine the two), the challenge for the algorithm design community is to find search strategies that provide sufficiently good solutions in as small as possible a number of fitness evaluations. In those cases where the computational time-complexity of the fitness function is high, the need to make progress in reduced numbers of evaluations is obvious. On the other hand, when the number of parameters is high, the issue tends to be different: in these circumstances, standard algorithms tend to converge prematurely, long before effecting a suitably extensive exploration of the parameter space; the challenge therefore becomes that of making the most of the available fitness evaluations, to achieve a better level of exploration in the available time.

In this article, we continue from [3] in exploring two quite distinct strategies that are both known to be effective on their own. The first of these, which is currently under rampant investigation within the EA community, is the technique of co-operative co-evolution (CC). This is best understood as a 'Divide and Conquer' strategy, which works by breaking down a many-parameter problem into a number of smaller sub-problems. Originally developed by Potter and De Jong [4], the essential idea behind their 'CCEA' algorithm was to partition the parameter space into separate sub-problems, and separately evolve solutions to each of these sub-problems, using, for each such sub-problem, fixed values for the parameters not included within it. A common-sense arrangement was made to periodically update these 'fixed' values, making use of best-so-far solutions from other sub-problems. Many variations on this idea have since been explored. In general, any 'CC' approach tends to be configured to evolve solutions to sub-problems one after another, each time using the best parameter values for the previous sub-problem when moving on to the next. In a sensibly configured 'CCEA', there are mechanisms for more

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.238

sophisticated interaction between sub-problems. For example, sub-problems might be solved in parallel, and there might then be brief spells of 'whole-problem' evolution, or hill-climbing, rooted in the best parameters found from the sub-problems so far. Meanwhile, it tends to be the case that the convergence speed of a basic EA is faster than that of a version of that EA with CC engineered into it. But, well-designed versions of CC have frequently proven more effective than their substrate EAs in tackling large scale problems [5].

Meanwhile, fitness inheritance (FI) is an entirely different approach to speeding up black box optimization, introduced by Smith, Dike and Stegmann [6]. FI tries to cut down on the need for 'real' function evaluations, by making a proportion of evaluations estimates, based on parents' fitnesses. The approach can be seen as a variation on a 'surrogate model' based EA. In a surrogate model approach, a model of fitness (the 'surrogate' is learned while the algorithm is running, and used some proportion of the time in place of the 'full' fitness function, to provide a fast estimate. In fitness inheritance, this surrogate model approximation to fitness is formed by 'inheriting' fitness values from one or more of an individual's parents [6, 7].

The work in this article follows on from that of [3]. Noting that both CC and FI could be intuitively regarded as 'orthogonal' and had not previously been explored in combination, the question explored in [3] was whether CC and FI could provide independent, additive improvements to a 'substrate' EA. The findings from [3] were positive, indicating that a simply-engineered combination of CC and FI could yield an algorithm that was superior to the underlying EA with only CC, or with only FI (and indeed, also superior to the EA with neither). It remains to be seen if there is benefit in similar engineering of more sophisticated EAs whose performance already approaches or occupies the state of the art, such as Yang, Tang and Yao's 'SaNSDE' [8] or Omidvar, Li and Yao's 'DECC-DML' [9].

We note that the latter would seem to be a 'big ask'. Although independently effective, CC and FI may, either separately or in combination, interfere with one or more of the carefully crafted mechanisms that boost performance in thoroughbred algorithms such as those mentioned. Of course, CC and/or FI may be included in such an algorithm, in which case we would be interested in incorporating the missing of the two – however the same potential problem still arises.

Meanwhile, a limitation of [3] was its focus on a limited number of test functions in terms of 'large-scale' (500D or 1000D tests). It therefore remains to be seen whether combining CC and FI could provide robust benefits in a large-scale optimization context. Finally, the detailed findings of [3] were that, although CC+FI was generally beneficial, there was certainly sensitivity to the 'fitness inheritance proportion' parameter, begging the future incorporation of an adaptive approach to the latter.

To explore these issues, in this paper we engineer CC+FI versions of both SaNSDE and DECC-DML, and we make our comparisons mainly on a commonly employed suite of 20 500D and 1000D test functions with emphasis on 1000D [10].

The new engineered versions of CC and FI incorporate an important amendment to the CC+FI combination that improves upon [3], and we also developed two simple adaptive versions of the CC+FI combination, to remove the need to specify the main parameter of FI, the 'fitness inheritance proportion'.

The remainder of the paper is set out as follows. In section II we run through the key background material; this involves brief introductions to the two key protagonists in this paper, fitness inheritance and co-operative co-evolution, each then followed by a brief associated literature review; the background material concludes by recalling the initial work on combining these two approaches. In section III we look at engineering CC and FI into SaNSDE, leading to an algorithm, which we denote CCDE-nFI; it is compared with DECC-G [11] and JACC-G [12] (for reasons noted in the section – essentially, the former supersedes SaNSDE, and the latter further supersedes it). In section IV we then describe our two simple approaches to adapting the fitness inheritance parameter in CCDE-nFI, and then we test them in the context of engineering CC and FI into DECC-DML. A concluding discussion is then offered in section V.

Our code is freely available at http://is.gd/cceafi, at which site can be found java code for CCDE-nFI (the engineered version of SaNSDE), an amended version Li's original DECC-DML [dmml] in matlab (this was necessary, as mentioned in section IV), and DECC-DMLaFI (our engineered version of DECC-DML, incorporating adaptive fitness inheritance).

II. BACKGROUND

A. *Fitness Inheritance* : *Introduction*

'Fitness inheritance' (FI) refers to the simple idea of making a fast estimate of an individual's fitness, rather than performing an accurate evaluation. FI can be potentially be deployed in any 'black box' algorithm context. Where c is an individual in the population, and '*eval*(c)' represents the evaluation of its fitness in the context of pseudocode, then we can effect the deployment of FI simply by replacing '*eval*(c)' with:

```
with prob. $P_{FI}$, run inherit($c$)
otherwise run eval($c$)
```

However, sometimes care needs to be taken about the context of the evaluation. In the majority of cases, this means that we should ensure two things: (i) the initial population is always evaluated with *eval*, and (ii) updating of the 'best so far' solution, and similar book-keeping, are done on the basis of fitness values returned by *eval*. With the latter protections in place, we can then replace a proportion P_{FI} of evaluations with fast 'inherited' estimates.

There are three commonly used approaches to producing an inherited fitness [13]. Where *ch* is the child of parents *p1* and *p2*, and where f_x represents the stored value of the previous evaluation of *x* (whether real or inherited), these are:

Averaged inheritance, in which the estimate of the child's fitness is simply the mean of its parents fitnesses:

$$inherit(ch) = \frac{f_{p1} + f_{p2}}{2}$$

Weighted inheritance: this is in the spirit of averaged inheritance, but introduces weights according to the child's different similarities to its two parents:

$$inherit(ch) = \frac{w_{p1} \cdot f_{p1} + w_{p2} \cdot f_{p2}}{w_{p1} + w_{p2}}$$

Parental inheritance: this is an extreme version of weighted inheritance, in which the entire current population *P* is used to estimate the fitness of *ch*:

$$inherit(ch) = \frac{\sum_{p \in P} w_p \cdot f_p}{\sum_{p \in P} w_p}$$

How weights are calculated in the latter two approaches depends on the algorithm and encoding context, but the overall idea is to guess the child's fitness by appealing to a very simple linear approximation to the fitness landscape that is either based solely on the parents, or based on the entire population. Naturally, also, the obvious variations can be made to calculate inherited fitness for children of just one parent, or of more than two parents.

B. *Fitness Inheritance: Brief Review*

As previously indicated, Smith, Dike and Stegmann [6] originated the concept of fitness inheritance (FI), at the same time introducing the first of the two simple inheritance schemes above. In the original work, Smith et al evaluated the concept on two problems. The first of these was the well-known (and these days very little used) 'ONEMAX test problem, in which the objective is simply to maximize the total number of 1s in a binary chromosome. Their second choice of test case was somewhat more interesting, being a realistic aircraft routing problem. On both problems they found impressive performance from FI. In [14], Sastry, Pelikan and Goldberg made some headway into a theoretical understanding of FI, untangling the relationship between population size and the fitness inheritance proportion on the ONEMAX landscape. Their conclusions, as regards separable landscapes, were that FI could lead to significant efficiency enhancements, halving the required number of function evaluations, and yielding speed-ups of between 1.75-fold and 2.25-fold. Pelikan and Sastry [15] went on to find even more impressive speedups (e.g. 30-fold) when using FI in other application, and this has also been found in many other studies [16—20].

Although little work so far has combined FI with CC (we only know of [3]), FI is increasingly being explored outside the region of 'standard' EAs, and being applied to nonstandard applications. For example in [21], FI is used as an essential tool for reducing a number of fitness evaluations when solving a problem of constructing robust continuous multi-objective test functions with various noise-induced features capable of uncovering truly capacity of the tested algorithms. Meanwhile, a modification of FI where fitness of a solution is based on its positional relationship with other particles has also demonstrated its benefits when used in PSO [21], typically suffering from overwhelmingly high number of fitness evaluations needed to find acceptable solution.

C. *Co-operative Co-evolution (CC): Brief Review*

The basic idea of co-operative coevolution (CC) is to solve a large-scale (many design variables) optimization problem by (i) decomposing it into several smaller sub-problems; (ii) solving each of the sub-problems; (iii) constructing a solution to the larger problem by combining the solutions to the smaller ones. In the original framework [4] (and in the majority of subsequent research), the 'decomposition' into sub-problems is simply effected by a partitioning of the decision variable space. For example, if we need to optimize a function of 1,000 variables, v1,v2,…,v1000, then we can decompose this into two smaller problems: the first sub-problem only considers variables 1—500; and the second only considers variables 501—1000.

In the above example, when optimizing the first sub-problem, the quality of solutions can only be estimated by assuming a reference set of fixed values for variables 501—1000. Similarly, when optimizing the second sub-problem, there needs to be a reference fixed set of values for variables 1—500. Typically, these reference sets may start at random, but be updated as the algorithm runs via the occasional evaluation of a judiciously composed full solution. The initialization and updating of these 'reference sets' is a key dimension of variation among the literature of CC variants, along with other key aspects such as the details of decomposition and its adaptation.

D. *Co-operative Co-evolution (CC): Brief Review*

Many researchers have investigated variations on the basic CC framework. Relatively recently, Yang, Tang and Yao [11] proposed their variant, DECC-G, in which the groups of parameters within a sub-problem adapted over time, in attempt to maximize the extent to which interacting parameters were present in the same sub-problem, This led to improved performance on non-separable problems, when compared to the standard framework. Another example of a variant of CC is that of Van den Bergh and Engelbrecht [23], who explored CC under the framework of particle swarm optimization (PSO). In their case, parameters were randomly assigned to sub-problems (this is also the case in [24]). In such random assignment, each sub-problem comprises a random selection of parameters scattered across the decision parameter vector, and this random assignment may change in every cycle of the algorithm. Among other key algorithm strategy choices in CC is the question of how, for any given sub-problem, to choose the reference set of fixed parameters that are *not* included in that sub-problem. The most common approach, naturally enough, is to use the best-so-far 'complete' solution, updating these reference sets whenever a new best-so-far is found.

An example of CC in use in the context of Differential Evolution is provided by Trunfio [25], who investigated the use of several variants of search space decomposition in parallel during short learning phases, allowing to adapt the size of subcomponents during the CC search. Meanwhile, Sayed et al [26] proposed a technique for the identification of variable interaction aimed at limiting the number of interdependent variables among decomposed problems – a common theme in CC research also echoed in [9]. Sayed et al's particular technique, related to the concept of separability and based on random resampling and propagation of variable partitions, is reported to improve the ability of the decomposition-based optimization models in scaling up to dimension 1000.

Another effective decomposition scheme is proposed in Mei, Li and Yao [27], which is designed especially for the large-scale capacitated arc routing problem. In this scheme, route information about best-so-far solution is actively employed in constructing the following decomposition to guarantee the non-decreasing quality of the decomposition. Differentiation between a vast number of possible decompositions is made based on special distances between the solutions which allows efficient identification of promising regions of the search.

E. Combining CC and FI

The idea explored in [3] was to combine CC and FI, with a view to understanding whether they provided independent additive benefits. A simple algorithm that combined the two strategies (CCEA-FI) was designed evaluated, and evaluate them on the same set of test functions that were explored in [24]. These were primarily 50D and 100D functions, and the results pointed squarely to the suggestion 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, CCEA-FI achieved superior statistics, although (typically) occasional anomalous results came from the Rosenbrock function, which showed high sensitivity to the fitness inheritance proportion parameter.

Nevertheless, promise was clearly shown in [3] for engineering both CI and FI into a 'substrate' algorithm, and the stage was set two main potential lines of research. First, to investigate any benefits from incorporating the CC and FI combination into existing state of the art algorithms, and second, to investigate adaptive versions.

III. Engineering CC and FI into SaNSDE: CCDE-nFI

Self-adaptive Differential Evolution (SaDE) was introduced by Qin et al. [28], with the aim of reducing the notorious problem-dependence of DE with respect to its main variation operators. SaNSDE [8] later incorporated further improvements – primarily by adopting additional adaptation and exploration mechanisms for the 'scale factor', *F*, a key parameter in differential evolution (DE) operators. SaNSDE is a sophisticated black box optimization algorithm with a srog performance profile, which, arguably can still be considered among the 'state of the art', and we took it as a candidate for the 'engineering into it' of both CC and FI. It turns out, as it happens, that it is not easy to find a state of the art algorithm that doesn't already have CC installed, so SaNSDE was a good choice for us in this respect.

We implemented SaNSDE from the description in the literature, and engineered CC+FI into it, using the same CC+FI framework as in [3], but with one key improvement. The basic CCEA framework deployed a 'random-grouping' version of CC, and basic FI with Averaged-Inheritance. The improvement was simply to ensure that – in the steps of CC where a best from each subpopulation is chosen to populate new reference sets – these were constrained to be chosen on the basis of real evaluations (although not actually requiring additional real evaluations). Preliminary tests showed that was led to significant improvement in results at high levels of the fitness inheritance proportion (as is intuitively reasonable). The lack of this strategy is presumably an explanation for reduced performance at high levels of inheritance as seen in [3], and also in [13], in which, although not involving CC, use us made occasionally of the 'best so far' irrespective of how its fitness was calculated

We denote our engineered version of SaNSDE as 'CCDEnFI'. Table I summarizes the results, showing the performance of CCDEnFI with the fitness inheritance proportion fixed at 90%.

It turns out that CCDEnFI outperformed our implementation of unadorned SaNSDE with high statistical confidence at most levels of the FI proportion, and to save space we omit those comparative results. Instead we focus on comparing directly with improved 'descendants' of SaNSDE that have been independently developed by the associated network of authors.

The first of the comparative algorithms listed is DECC-G [9], which 'supersedes' SaNSDE in this context, being a version of SaNSDE that has independently been configured with CC. The second is JACC-G [12], which further improves on DECC-G via developments in the area of its embedded DE algorithm. The choice of test functions is the suite used in Yang Tang and Yao [11], which enables the maximal comparisons we can make with previously published results for DECC-G and JACC-G.

TABLE I. COMPARING CCDE-NFI (WITH FI AT 90%) WITH DESCENDANTS OF SANSDE

F	D	DECC-G Mean	JACC-G Mean	CCDE-nFI Mean
F1	500	6.33e-27	-	**7.6e-313**
	1000	2.17e-25	2.7e-80	**8.73e-123**
F2	500	5.95e-15	-	**2.15e-297**
	1000	5.37e-14	**2.3e-20**	INF
F3	500	6.17e-25	-	**0.00e+00**
	1000	3.71e-23	2.4e-10	**0.00e+00**
F4	500	**4.58e-05**	-	3.9e+01
	1000	1.01e-01	**8.0e-05**	8.11e+01
F5	500	4.92e+02	-	**2.97e+02**
	1000	9.87e+02	9.83e+02	**9.23e+02**
F6	500	0.00e+00	-	0.00e+00
	1000	0.00e00	0.00e+00	0.00e+00
F7	500	**1.50e-03**	-	1.56e-03
	1000	8.40e-03	**1.2e-03**	4.62e-03
F8	500	-209491	-	-209491
	1000	-418983	-418983	-418983
F9	500	0.00e+00	-	0.00e+00
	1000	3.55e-16	0.00e+00	0.00e+00
F10	500	9.13e-14	-	**5.07e-14**
	1000	2.22e-13	1.4e-14	**1.05e-13**

Matching the key experimental variables with those reported in association with the comparative results, each experiment of CCDEnFI was run for 2,500,000 function evaluations (real, not inherited) for 500D cases, and for 5,000,000 function evaluations in the 1000D cases. The population size was 100, and the sub-problems (in the context of CC) were always of dimension 100 (hence, 5 sub-problems in the 500D cases, and 10 sub-problems in the 1000D cases).

As we have mentioned, to save space we show only the results for CCDEnFI using 90% fitness inheritance. This achieves better or equivalent mean performance than DECC-G on 8 of the 10 500D cases (winning 5, losing 2), and 8 of the 10 1000D cases (winning 6, losing 2). Meanwhile, it performs better or equivalently to JACC-G on 7 of the 10 1000D test cases (winning 4, losing 3). Comparative 500D results were not available for JACC-G.

Simple statistical analyses of these results (using Student's *T* test and a prior confidence level of 0.05) suggests competitive (i.e. 'equal' performance) for CCDE-nFI when compared with either JACC-G or DECC-G. There is not enough evidence to claim superiority of CCDEnFI at a high level of confidence in this context, especially after the Bonferroni correction. However we can conclude that simply engineering CC and FI into a sophisticated algorithm (such as SaNSDE) in a straightforward way, is able to provide potentially superior results, and at least competitive results, to those available via a suite of alternative sophistications, such as those incorporated beyond SaNSDE in each of DECC-G and JACC-G.

Finally, the performance (not shown here) of CCDEnFI with lower levels of fitness inheritance was generally similar to that shown for 90%, although lower levels (10% -- 40%) occasionally showed reduced (and highly problem-dependent) performance. Obviously, to achieve candidacy as a usable algorithm, CCDEnFI (or, the engineering of CC and FI into an algorithm in general) needs to either come along with good guidelines for setting the fitness inheritance parameter, or there needs to be a viable adaptive approach. The latter notion is investigated next.

IV. ENGINEERING ADAPTIVE CC AND FI INTO DECC-DML

So far, we have investigated the combination of CC and FI in [3] and herein, without the intention of declaring a new algorithm, but with the intention of exploring their potential as adornments to arbitrary black box algorithms. Nevertheless, to be deployable and 'proposable' an algorithm that incorporates both CC and FI must come along with either a good set of fixed parameters, or with an adaptation scheme for its key parameters. The single key parameter that looms large in this respect is the fitness inheritance proportion (often abbreviated as 'FI', at the risk of overloading that term). In this section we take a first step at investigating adaptive schemes for it in the CC+FI context.

Two simple 'first-cut' adaptive approaches were designed as follows:

Method A: In this approach, the performance of 10%, 20%, … and so on, up to 90% FI are sampled in the first batch of evaluations, with each being the sole setting for a duration of FItest evaluations (hence 9xFItest evaluations are devoted to this sampling process). The value of FI that performed best in this early sampling is then used for the remainder of the run.

Method B: This begins in the same way as method A. However the sampling process is repeated every EPOCH evaluations. After a set of EPOCH evaluations is completed, the sampling process is repeated (consuming 9xFItest evaluations), and the best FI from that most recent sampling is then used for a further (EPOCH - 9xFItest) evaluations; this process is repeated until termination.

Our adaptive schemes were tested in the context of a further investigation into engineering CC and FI to a sophisticated state of the art algorithm. In this case, the algorithm of choice was DECC-DML [9]; DECC-DML is a further co-operative co-evolution approach based on DE, which improves on the previous 'random grouping' method for constructing CC sub-problems, aimed at being more effective at identifying interacting variables, which are then best treated within the same sub-problem.

DECC-DML and its available results provide a further opportunity for us to test the engineering of combined CC and FI. However, a slight drawback – which has turned out not easy to avoid in the context of current large-scale optimization research – is the fact that DECC-DML already incorporates CC. Nevertheless, the spirit of investigation remains in that we are exploring the capability of combinations of CC and FI, especially in the context of already-sophisticated methods. We therefore soldiered on, and proceeded by retaining DECC-DML's existing (and quite sophisticated) variant of the CC mechanism, and engineered our adaptive versions of FI into the existing DECC-DML code (which the authors had made available).

The latter was done, and the resulting 'DECC-DML-aFI' algorithm was compared with DECC-DML over the CEC 2008 'large scale global optimization' test suite of 20 1000D test functions [10]. Following preliminary investigation, the method A and B parameters FItest and EPOCH (only in method B) were set to 100 and 300,000 respectively. Table II summarizes the results. The results in each case are the means of 20 independent runs, each of which continued for a maximum of 5,000,000 real evaluations.

Note: DECC-DML source code in Matlab, as presumably deployed in [9] was obtained from Xiaodong Li's website [29]. We engineered the inclusion of FI directly into this version. However, we noticed that the mechanisms in the original DECC-DML source code, associated with calculating and updating the 'delta' value, incorporated some calls to function evaluations that were not accounted for in the total which counted towards algorithm termination. The results for DECC-DML shown in Table II are therefore from our own runs with the corrected version of the source code, ensuring comparison on an equal basis in terms of real function evaluations.

In Table II, the best mean result for any given function is highlighted in bold, while underline is used to indicate with of the two adaptation methods achieved the best mean (independent of whether either achieved overall best mean for that function).

Analysis of Table II shows that the laurels are shared quite equally between the three approaches. Original DECC-DML has 7 'wins', compared with 7 wins for DECC-DML-aFI Method B, and 6 wins for the Method A version. Similarly, both Method A and Method B show a complementary performance profile over the 20 functions.

This suggests that, at best, we can say that the engineering of FI into DECC-DML (via either of the two adaptive methods) has adjusted its overall performance profile, while not providing an 'overall' improvement (and certainly not providing any overall detriment) to its performance, as estimated over this particular function suite.

Isolating each DECC-DML-aFI method and comparing that with DECC-DML, we note that Method A achieves a better mean value on 13 of the 20 functions, while Method B achieves a better mean on 9 of the 20 functions. If we consider these success rates against expectations according to cumulative binomial probability (assuming null hypothesis of a success rate of 0.5 in each case), we note that 13 or more 'wins' can be expected to occur with probability 0.132, while 9 wins (or equivalently, 11 wins for DECC-DML) can be expected with probability 0.411. This further confirms that the engineering of FI into the algorithm has not caused any overall loss in performance, but has clearly changed the performance profile in a way we have yet to fully understand.

TABLE II. ENGINEERING SIMPLE DAPATIVE FI INTO DECC-DML (DECC-DMLAFI

F	(corrected) DECC-DML results	DECC-DML-aFI Method A	DECC-DML-aFI Method B
F1	8.68625e-08	2.94803e-08	**1.876923e-08**
F2	**1.091e+3**	1.1209e+03	1.12580e+03
F3	1.94621e-07	1.23452e-07	**9.007184e-08**
F4	6.68762e+12	**5.56665e+12**	5.78655e+12
F5	**2.82387e+08**	2.97985e+08	2.84147e+08
F6	**9.2674e+04**	1.09202e+06	6.37554e+05
F7	2.31131e+08	**1.65121e+08**	4.10521e+08
F8	**9.34754e+07**	9.36069e+07	1.16837e+08
F9	1.3458e+08	1.27753e+08	**1.25674e+08**
F10	1.2646e+04	**1.26387e+04**	1.32528e+04
F11	4.60642e+00	6.86595e-06	**3.226461e-06**
F12	**3.97037e+06**	4.09242e+06	4.86637e+06
F13	1.320e+03	1.15158e+03	**1.27881e+03**
F14	4.31999e+08	**4.18162e+08**	4.19249e+08
F15	1.5586e+04	**1.55049e+04**	1.62337e+04
F16	3.41033e+01	1.71398e+01	**5.378018e-05**
F17	**7.04423e+06**	7.14235e+06	8.36006e+06
F18	7.95526e+03	3.97663e+03	**3.90853e+03**
F19	1.783e+07	**1.73677e+07**	2.16407e+07
F20	**9.944e+02**	1.0201e+03	1.00183e+03

In terms of recommendations for practitioners, the changed performance profile of the FI-engineered version may well be more favourable for particular landscapes (and intuition would suggest these to be those on the smoother side), but understanding that requires further research. Ultimately, however, our results seem consistent with the notions that:

- Engineering CC and FI into a good algorithm may well lead to improved performance, and will not lead to reduced performance;
- Engineering CC and FI into a sophisticated, state of the art algorithm (or engineering FI into a state of the art algorithm that already includes CC) may also lead to improvement, and will likely not lead to detriment. However, the more 'state of the art' the original algorithm is, the more the exercise may provide diminishing returns.

This theme will be a little more explored in the next section.

V. Concluding Summary and Discussion

Earlier work had suggested promise for using *bot*h co-operative coevolution (CC) and fitness inheritance (FI) in the design of a black box optimization algorithm (especially with large-scale problems in mind). Essentially, that paper had shown that the strategy of combining CC and FI works well, providing independent and additional improvements, in the case where the underlying EA being engineered was otherwise a straightforward algorithm.

In this paper, however, we further explored the strategy of combining CC and FI by jumping in at the 'deep end' by seeing if combining these strategies could lead to any advancement for algorithms that were already 'state of the art', in the sense that they already comprise a variety of sophisticated mechanisms that boost their performance.

The raw results of this effort were mixed. Of course, comparing new approaches with 'state of the art' algorithms can always be seen as a risky strategy, in cases where the context seems to require relentless advances in performance. However, in the case of this paper, our experiments amount to further probing of the behavior of CC and FI combined.

The question (perhaps rationalized a little after the fact) is *not* 'Can CC+FI improve a state of the art algorithm (which doesn't already use both…)?'. We would *a priori* not hold out high expectations that this is possible, because 'state of the art' algorithms tend to incorporate a complex cookbook of mechanisms that would likely be perturbed by significant additional mechanisms such as either CC, FI or both. Instead, the question is – given that we already know CC+FI can improve a 'straightforward' algorithm – 'Can CC and FI be engineered into arbitrary black box optimization algorithms, without doing harm?'. (alternatively: if the algorithm contains either CC or FI already, can we engineer the other one into it without doing harm?). If the answer to that was positive, it would suggest that adding CC and FI to simpler algorithms would lead to improvements, and adding them to state of the art algorithms would leave their overall performance profile unchanged. This is precisely what you want if, beforehand, you don't actually know where your algorithm sits in the virtual 'algorithm league table'.

In that sense, we would tentatively conclude that the new approach to combining CC and FI explored here (involving the amended and adapted FI) represents a recommended algorithm-enhancement strategy.

Of course, this area of algorithm design is crying out for a deeper understanding of (among many other things) how, why and what circumstances CC and FI can provided additive improvement. As we speculated in [3], the distinct nature of the two mechanisms seemed to herald good potential for their combination. Intuition suggests that FI works best when the landscape is locally (at least) smooth; however, if the local shape is *not* smooth, one might nevertheless expect FI to work well if the ruggedness is within reasonable bounds; in that circumstance, for example, an occasional inherited fitness that ends up being a poor estimate may have the same effect, in terms of algorithm dynamics, as the injection of diversity via low-pressure selection. Meanwhile CC works well when the problem can be partitioned into sub-problems which, to at least a small extent, cluster the interacting variables. In combined CC and FI, FI operates on the sub-problems which, depending in the decomposition strategy, may therefore be somewhat more rugged landscapes, however the earlier observation still suggests benefit for FI.

We need to move beyond such speculation, and forward to theoretical lines of investigation that might shed light on CC and FI combinations, as well as associated black box algorithm design mechanisms. In ongoing work, we are slowly moving in that direction, by means of gathering data at runtime. Such data includes tracking the ruggedness of sub-problems, and tracking the context of use and the performance of individual FI operations during a CC+FI run. We think that analyses of these data will illuminate potential directions for theoretical analysis and further algorithm design strategies.

Acknowledgment

We are grateful to the Libyan government for support of the first author. We are grateful to a variety of sources for support of the second and third authors.

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] Hameed, A.; Corne, D.; Morgan, D.; Waldock, A., "Large-scale optimization: Are co-operative co-evolution and fitness inheritance additive?" Computational Intelligence (UKCI), 2013 13th UK Workshop on, vol., no., pp.104, 111, 9-11 Sept. 2013.

[4] 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.

[5] Y. Wang, B. Li, (2009) A Self-adaptive Mixed Distribution BasedUnivariate Estimation of Distribution Algorithm for Large Scale Global Optimization, in 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.

[6] 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.

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

[8] Yang, Zhenyu, Ke Tang, and Xin Yao. "Self-adaptive differential evolution with neighborhood search." Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on. IEEE, 2008.

[9] Omidvar, Mohammad Nabi, Xiaodong Li, and Xin Yao. "Cooperative co-evolution with delta grouping for large scale non-separable function optimization."Evolutionary Computation (CEC), 2010 IEEE Congress on. IEEE, 2010

[10] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, , and Z. Yang, "Benchmark functions for the cec'2008 special session and competition on large scale global optimization," Nature Inspired Computation and Applications Laboratory, USTC, China, Tech. Rep., 2007, http://nical.ustc.edu.cn/cec08ss.php

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

[12] Yang, Zhenyu, et al. "An adaptive coevolutionary differential evolution algorithm for large-scale optimization." *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009.

[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] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Zhao, Li, Wan-Ke Cao, and Yu-Tao He. "Fitness inheritance-based evolutionary algorithm and its application in hybrid electric vehicle design." *International Journal of Wireless and Mobile Computing* 7.2 (2014): 180-186.

[20] Barbour, Robert, David Corne, and John McCall. "Accelerated optimisation of chemotherapy dose schedules using fitness inheritance." Evolutionary Computation (CEC), 2010 IEEE Congress on. IEEE, 2010.

[21] Chi-Keong Goh, K. C. Tan, C.Y. Cheong, Yew Soon Ong, An investigation on noise-induced features in robust evolutionary multi-objective optimization, Expert Systems with Applications, 37(8):5960-5980, 2010

[22] Chaoli Sun, Jian-Chao Zeng, Jengshyang Pan, Songdong Xue, Yaochu Jin, A new fitness estimation strategy for particle swarm optimization, Information Sciences, 221:355-370, 2013.

[23] 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.

[24] 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.

[25] Giuseppe A. Trunfio, A Cooperative Coevolutionary Differential Evolution Algorithm with Adaptive Subcomponents, Procedia Computer Science, Volume 51, Pages 834-844, 2015.

[26] Eman Sayed, Daryl Essam, Ruhul Sarker, Saber Elsayed, Decomposition-based evolutionary algorithm for large scale constrained problems, Information Sciences 316, 457–486, 2015

[27] Yi Mei, Xiaodong Li, Xin Yao, Cooperative Coevolution With Route Distance Grouping for Large-Scale Capacitated Arc Routing Problems. IEEE Trans. Evolutionary Computation 18(3): 435-449 (2014)

[28] Qin, A. Kai, and Ponnuthurai N. Suganthan. "Self-adaptive differential evolution algorithm for numerical optimization." Evolutionary Computation, 2005. The 2005 IEEE Congress on. Vol. 2. IEEE, 2005.

[29] http://goanna.cs.rmit.edu.au/~xiaodong/publications/publications.html