Differential Evolution in High-Dimensional Search Spaces

Olusegun Olorunda
Department of Computer Science
University of Pretoria
South Africa
Email: olorunda@cs.up.ac.za

Andries P. Engelbrecht
Department of Computer Science
University of Pretoria
South Africa
Email: engel@cs.up.ac.za

Abstract—A possible way of dealing with a high dimensional problem space is to divide it up into smaller parts, and to have each part optimized by a separate population. A mechanism is then defined to construct a complete solution from the subpopulations, and to evaluate the entities contained in the subpopulations. This form of cooperation has been successfully applied to Particle Swarm Optimization (PSO), by [1] in the cooperative split PSO, and to genetic algorithms, in the cooperative coevolutionary genetic algorithm, developed by [2], on which the cooperative split PSO is based.

This paper investigates cooperation in differential evolution (DE) with the aim of determining the effects of multiple participants in dealing with high-dimensional problem spaces.

I. INTRODUCTION

Differential Evolution [3] is a population-based search strategy which is similar to standard evolutionary algorithms, i.e. it is a stochastic search for an optimal solution to a given problem. The main difference between differential evolution and standard evolutionary algorithms lies in the reproduction step. For the standard evolutionary algorithm, reproduction typically involves two parents, while with the differential evolution algorithm, reproduction involves three parents.

Section II provides background information on the DE algorithm and the particle swarm optimization (PSO) algorithm which is used as a control algorithm in this paper. Section III introduces the concept of a cooperative algorithm — how it works to split up the problem space and then recombine the solutions from the participant algorithms. Section IV, briefly, discusses the difficulty posed by high dimensional problem optimization. Section V outlines the experimental setup and results analysis and Section VI contains the various conclusions that were made from this analysis. Lastly (also in Section VI) details about possible further research with regards to this work are given.

II. BACKGROUND

This section outlines the basic workings of the differential evolution and particle swarm optimization algorithms.

A. The Differential Evolution Algorithm

The differential evolution (DE) algorithm makes use of an arithmetic operator which depends on the differences between randomly selected pairs of individuals. For each parent, \( \vec{C}_{g,n} \), in a particular generation, \( g \), an offspring, \( \vec{O}_{g,n} \), is created as follows:

1) Select three individuals, \( \vec{C}_{g,n_1}, \vec{C}_{g,n_2}, \vec{C}_{g,n_3} \), at random from the current population where \( n_1 \neq n_2 \neq n_3 \neq n \) and \( n_1,n_2,n_3 \sim U(1,\ldots,N) \). \( N \) denotes the total number of individuals in the population.
2) Select a random number \( i \sim U(1,\ldots,I) \), where \( I \) is the number of parameters contained in a single individual.
3) For each parameter \( j = 1,\ldots,I \) of \( \vec{O}_{g,n} \), if \( U(0,1) < p_r \) or \( j = i \), let

\[
\vec{O}_{g,n,j} = \vec{C}_{g,n_j} + \gamma(\vec{C}_{g,n_1} - \vec{C}_{g,n_2})
\]

otherwise, let

\[
\vec{O}_{g,n,j} = \vec{C}_{g,n_j}
\]

\( p_r \) is the probability of reproduction with \( p_r \in [0,1] \), \( \gamma \) is a scaling factor, with \( \gamma \in [0,\infty] \). \( \vec{O}_{g,n_j} \) and \( \vec{C}_{g,n_j} \) indicate, respectively, the \( j \)-th parameter of the offspring and the parent.

This means that each parameter, \( \vec{O}_{g,n_j} \), of the offspring, \( \vec{O}_{g,n} \), is a linear combination of 3 randomly selected individuals when \( U(0,1) < p_r \); otherwise the offspring inherits that parameter directly from its parent. Even when \( p_r = 0 \), at least one of the offspring parameters will differ from the parent when \( j = i \).

After reproduction, each parent in the population for the current generation is replaced with its offspring if its offspring represents a better solution. Otherwise the parent is carried over to the next generation.

B. The Particle Swarm Optimization Algorithm

The Particle Swarm Optimization (PSO) algorithm, introduced in [4] and [5], maintains a swarm of particles where each particle in the swarm is a potential solution to the problem being solved.

Let \( \vec{x}_i(t) \) denote the position of a particle \( i \) in the solution search space at a time step \( t \). The position of the particle could be changed by adding a velocity, \( \vec{v}_i(t) \), to the current position as follows:

\[
\vec{x}_i(t + 1) = \vec{x}_i(t) + \vec{v}_i(t + 1)
\]

The particle velocity is the driving force behind the optimization process. It represents the knowledge gained by the particle, both through the experience of the particle itself
(cognitive component) and through socially exchanged information acquired from interaction with other particles in its neighborhood (social component).

The first two PSO algorithms that were originally developed are the gbest PSO, which uses the star social network topology and the lbest PSO, based on a ring social network topology [6], [7]. The gbest PSO algorithm is used in this paper and the discussion on PSO will, therefore, be focused on that version of the PSO algorithm.

For a given particle \( i \) in a swarm, the velocity is calculated as follows:

\[
\vec{v}_{ij}(t+1) = \omega \vec{v}_{ij}(t) + c_1 r_1(t) [\vec{y}_{ij}(t) - \vec{x}_{ij}(t)] + c_2 r_2(t) [\vec{y}_{ij}(t) - \vec{x}_{ij}(t)]
\]

where \( \vec{v}_{ij}(t) \) is the particle \( i \) velocity at dimension \( j = 1, \ldots, n_x \), and at time step \( t \), \( \vec{x}_{ij}(t) \) is the particle \( i \) position at dimension \( j \), and at time step \( t \), \( c_1 \) and \( c_2 \) are positive constants of acceleration and are used to scale the contributions of the cognitive and social components, respectively, and \( r_1(t), r_2(t) \sim U(0,1) \) are random numbers sampled from the uniform distribution \( U(0,1) \) used to add a stochastic element to the PSO algorithm. An inertia weight value given by \( \omega \) is used to control the influence of the previous velocity on the new velocity.

\( \vec{y}_{ij}(t) \) is the personal best position, for a particle \( i \), and is the best position that the particle has visited until time step \( t \). \( \vec{x}_{ij}(t) \) represents the global best position, at time step \( t \). It denotes the best position (and consequently, solution) that has been found so far by any of the particles in the swarm.

A summary of the gbest PSO algorithm is given below:

1) Create and initialize an \( n_x \)-dimensional swarm, \( S \).
2) For each particle \( i = 1, \ldots, S \), set the personal and global best positions; update the particle’s velocity and position
3) Repeat step 2 until a stopping condition is reached.

III. THE COOPERATIVE SPLIT ALGORITHM

The cooperative co-evolutionary genetic algorithm (CCGA) was developed by Potter [2]. The cooperative split PSO algorithm, used in this paper, was developed by Van den Bergh and Engelbrecht [1], [8] and is based on Potter’s algorithm.

With Potter’s algorithm, an individual’s genes are distributed over a set of \( K \) disjointed subpopulations with each having the task of evolving a limited set of genes. Thus, no single subpopulation has the necessary information to solve the problem itself — they must all be combined to construct a solution. \( K \) is referred to as the split factor.

A solution is constructed by combining the best individuals from each subpopulation. The problem becomes how to determine the best individual from each subpopulation considering the fact that the individuals in question do not represent complete solutions. This is solved by maintaining the complete solution as a context vector. All components within the context are kept fixed and only the component that corresponds to the current subpopulation for which the best individual is sought is changed. For each individual in the subpopulation, the value of the corresponding context vector component is replaced with that of the individual. Other components in the context vector are kept fixed at their previously determined best values. The original fitness function is then used to evaluate the fitness of the context vector. The fitness function is evaluated after each component of the context vector is updated resulting in a finer grained search, which could possibly help to avoid premature convergence.

The cooperative split algorithm is outlined below for an \( n_x \)-dimensional problem with a split factor \( K \). The details of the algorithm as documented here differ slightly from the algorithm in [1], [8]. This is in order to provide a better explanation of the workings of the algorithm and to generalize it for any type of subpopulation (e.g., PSO, DE, etc).

1) Set iteration counter \( g = 0 \)
2) Initialize \( K \) subpopulations within an \( n_x \)-dimensional problem space. The populations are initialized so that they have access to an \( \frac{n_x}{K} \)-dimensional piece of the problem. Combining the best solutions found by the \( K \) subpopulations will result in an overall “best” solution which will be maintained in the context vector.
3) Execute the cooperative algorithm:
   a) If any subpopulations have not been selected for the current iteration, select a subpopulation, \( S_i \), with \( k = 1, \ldots, K \), (either randomly or following a sequential listing) and do the following:
      i) execute the corresponding algorithm (say DE)
      ii) for each individual \( i = 1, \ldots, S_k \)
          • if \( \mathcal{F}(b(k,S_k,x_i)) < \mathcal{F}(b(k,S_k,S_{k,S})) \) then
            \( S_k,S_{k} = S_k,x_i \) (replace best individual for the current subpopulation if the new best individual results in a better overall fitness of the context vector)
   b) If all subpopulations have been selected for the iteration, go to 4
4) Test for stopping condition. If stopping condition is not satisfied, set \( g = g + 1 \) and repeat 3.

Each subpopulation is only executed once per iteration. For this algorithm, \( b(k,z) \) returns an \( n_x \)-dimensional vector which is formed by concatenating the best solutions from all the subpopulations, except for the \( k \)-th component which is replaced with \( z \). \( z \) represents the solution vector of the individual (particle for a PSO) from subpopulation \( S_k \) that is currently being analyzed. The context vector is therefore defined as:

\[
b(k,z) = (S_1,y, \ldots, S_{k-1},y, z, S_{k+1},y, \ldots, S_K,y) \quad (5)
\]

This paper contains results obtained from both a cooperative DE and a cooperative PSO algorithm, with the cooperation handled by the split cooperative algorithm, as defined above.

IV. THE PROBLEM WITH HIGH DIMENSIONAL SEARCH SPACES

Most optimization algorithms are adaptable enough and efficient enough to sufficiently solve most optimization problems.
with an acceptable level of accuracy. As these problems grow in dimensionality, and become more complicated, it becomes harder for the algorithms to find solutions that are accurate enough. This is a problem as these algorithms are needed to be scalable in order for them to be more useful in research and industry.

A cooperative algorithm is supposed to divide up this complexity among the participating algorithms, thus, making the problem easier to solve. It is this “divide and conquer” theory which will be tested in this paper.

V. EXPERIMENTAL SETUP AND RESULTS ANALYSIS

This section details the experimental procedure under which the different algorithms were tested. Some information on the problems used for testing is given together with a brief explanation of the criterion used for evaluation (accuracy). Lastly, a summary of the different algorithm performances is given together with some comparison analysis.

All code was implemented using CILib (Computational Intelligence Library), an open source library and framework for computational intelligence algorithms, implemented in Java (see http://cilib.sourceforge.net for more information).

A. Algorithm Implementation Settings

General settings under which all the algorithms were tested are outlined as follows:

- Non-cooperative versions of the DE and PSO algorithms were implemented as controls, together with a sixteen participant cooperative PSO.
- The DE/rand/1 [9] strategy was used for the DE and a gbest topology was used for the PSO algorithms.
- The algorithms that were compared with each other were ensured to have the same total number of entities\(^1\), e.g. a non-cooperative PSO with 125 particles would be compared to a cooperative algorithm with 5 participants, each having 25 entities. This was done in order to ensure that the same number of fitness evaluations apply for all the algorithms being compared.
- On all the functions (except Michalewicz) the total number of entities was set at 320. For Michalewicz, the total number was 125.
- Measurements taken from each simulation are the accuracy values every 10 iterations (every iteration for Michalewicz) for 30 samples with random starting conditions. Each algorithm was executed for 10000 iterations on each problem, except Michalewicz (5000 iterations).
- The following naming convention was used for the cooperative algorithms: \(\text{CDE}_k\) represents a Cooperative DE with \(k\) participants and \(\text{CPSO}_k\) represents a Cooperative PSO with \(k\) participants.

Specific algorithm implementation details are given next:

- **DE** was implemented with the following settings:
  - Individuals are randomly selected from the population to be parents.
  - Probability of reproduction, \(p_r = 0.6\) and scaling factor, \(\gamma = 0.5\)
- **PSO** was implemented with the following settings:
  - All simulations have been run using the Synchronous Iteration Strategy [10].
  - The initial velocities were set to random values for each particle\(^2\).
  - Particle velocities and positions were updated using equations 4 and 3 respectively.
- **Cooperative DE** and **Cooperative PSO** were implemented with the following settings:
  - The problem was split into parts of equal dimension and assigned to each participant algorithm.
  - \(\text{CDE}_k\) was evaluated for \(k = 4, 8, 16\) for each fixed overall population size. For example, a 320-entity \(\text{CDE}_k\) would run with four 80-entity participants, eight 40-entity participants and sixteen 20-entity participants. \(\text{CPSO}_k\) is evaluated, only for \(k = 16\). For the Michalewicz function, the cooperative algorithms are evaluated for \(k = 5\).

B. Evaluation Criterion

The accuracy of the different algorithms with regards to solving the problems was measured. In order to do this, the mean of the 30 sample fitness values obtained at each resolution was calculated, to obtain an average performance of the algorithm in question. The natural logarithm of these average performances were then plotted against the number of iterations to illustrate which algorithm performed better. \(\ln(x) \sim x\) and therefore, using the natural logarithm as opposed to the actual mean values will enable the solution paths to be more “separated” while keeping the same shape as the original path.

The optimization problems that the algorithms have been tested on are minimization problems and, consequently, lower values are better.

C. Analysis of Results

This section documents the problems on which the different algorithms were executed and contains graphs illustrating the performance of the different algorithms on the different versions of the problem. A summary of the results obtained is given in Tables 1 and 2.

1) **The Spherical Function:** This is a very simple, unimodal function with no interaction between its variables. It is defined as follows:

\[
f(x) = \sum_{i=1}^{n} x_i^2
\]  

with \(-5.12 \leq x_i \leq 5.12\) and global minimum given by \(f(0, \ldots, 0) = 0\)

\(^1\)an entity refers to an individual in a population (DE) or a particle in a swarm (PSO)

\(^2\)an alternative would have been to set the initial velocities to zero
The results acquired when the different algorithms were run on a 32-dimensional Spherical function for 10000 iterations is given in figure 1. As can be seen, the algorithms are all able to minimize the function below 1 ($\ln(1) = 0$). The worst performing in this case is the CDE$_{16}$ algorithm, while CDE$_4$ shows the quickest convergence to the minimum (reaching a high level of accuracy in approximately 2500 iterations) closely followed by CPSO$_{16}$ and CDE$_8$ in that order.

Figure 2 illustrates the results for the 512-dimensional Spherical function. The non-cooperative algorithms show a reduced performance here as the increased complexity involved in solving the problem begins to manifest itself. DE is the worst performer and PSO does not do much better. Neither does CPSO$_{16}$. In comparison, the CDE$_k$ algorithms begin to show some of the advantages of cooperation. CDE$_8$ is, by far, the best performer, beating CDE$_4$ (which is second best) and CDE$_{16}$ (third best) by a huge margin. The CDE$_{16}$ algorithm converges quickest of the CDE$_k$ algorithms, but to a minimum which is not as good as those achieved by CDE$_4$ and CDE$_8$.

As can be observed from the graph path for CDE$_{16}$, it continues to converge, even after 10000 iterations. The optimizer does not stagnate, which is good, as it ensures that a better solution could yet be found.

The results at dimensionality 2048 are illustrated in figure 3.

Here, the non-cooperative versions of the DE and PSO stagnate almost immediately. CPSO$_{16}$ performs slightly better but stagnates as well. The CDE$_k$ algorithms, however, all reach better levels of accuracy in comparison, with the CDE$_{16}$ algorithm seeming to stagnate after about 4000 iterations. CDE$_4$ performs worse that CDE$_{16}$ but does not stagnate, thus keeping alive the possibility that a better solution might be found. CDE$_8$ out-performs all the other algorithms, reaching the best levels of accuracy and continuing to minimize the function after 10000 iterations.

2) The Rosenbrock Function: Rosenbrock’s valley is a classic optimization problem, also known as the Banana function. The global optimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial, however convergence to the global optimum is difficult and hence this problem has been repeatedly used to assess the performance of optimization algorithms [11]. It is defined as follows:

$$f(x) = \sum_{i=1}^{n-1}(100(x_{2i} - x_{2i-1})^2 + (1 - x_{2i-1}^2))$$

with $-2.048 \leq x_i \leq 2.048$ and global minimum given by $f(1, \ldots, 1) = 0$.

Figure 4 shows the results for the 32-dimensional Rosenbrock function.

The DE algorithm performs better than all the other algorithms with CDE$_4$ out-performing PSO to come second best. The CPSO$_{16}$ algorithm is slightly worse than PSO but does better than both the CDE$_8$ and CDE$_{16}$ algorithms which perform worse than all the others.

Figure 5 shows the results for dimension 512.
Again, the CDE algorithms seem to perform better on the higher dimensional problem, with CDE4, CDE8, and CDE16 performing almost equivalently. CDE8 performs only slightly better than CDE4 and CDE16 with the three algorithms outperforming CPSO16. The non-cooperative algorithms do not perform well on this version of the Rosenbrock function, with DE stagnating almost immediately, and PSO not performing much better than DE.

Figure 6 shows the results for dimensionality 2048.

DE and PSO perform almost equivalently, both stagnating almost immediately. CPSO16 does much better in comparison but falls short of the CDEk algorithms. CDE8 has slightly better performance than CDE16, but both optimizers are almost completely stagnated. The CDE4 algorithm performs worse than both the CDE8 and CDE16 algorithms but it has not stagnated after 10000 iterations and could, possibly, have found a better solution given more time.

3) The Rastrigin Function: Rastrigin’s function is based on the Spherical function, with the addition of some cosine modulation to produce many local minima. It is, consequently, highly multi-modal. The location of the minima are, however, regularly distributed [11]. It is defined as follows:

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

(8)

with \(-5.12 \leq x_i \leq 5.12\) and global minimum given by \(f(0, \ldots, 0) = 0\).

Results for the 32-dimensional version are illustrated in figure 7.

Rastrigin is a difficult function to minimize and all of the algorithms stagnate eventually. DE seems to perform quite badly on Rastrigin and performs worse than all the others. PSO performs second worst, with CPSO16 doing slightly better than PSO. The best performing algorithm is CDE4, which reaches a very high accuracy level. It is followed by CDE8. CDE16 also manages to minimize below 1 (\(\ln(1) = 0\)) but it stagnates at a comparatively worse minimum than the CDE4 and CDE8 algorithms.

When the dimensionality was increased to 512, the results as illustrated in figure 8 were acquired.

CDE16 performs better than the other algorithms, with CPSO16 and PSO performing almost equivalently for second and third place, respectively. The CPSO16 algorithm, however converges much quicker than PSO, suggesting that PSO might gain a better result beyond 10000 iterations. The CDE8 and CDE4 algorithms perform fourth and fifth, respectively, with DE performing worse than all the others.

The results for the 2048-dimensional Rastrigin function are illustrated in figure 9.

Again, the CDE16 algorithm performs better than all the others. Also, it does not stagnate over the course of 10000 iterations. CDE8 (second best) does slightly better than CPSO16 (third), and CDE4 (fourth best) does slightly worse. The DE algorithm stagnates immediately and PSO stagnates soon afterwards.

4) The Griewank Function: Griewank’s function is similar to Rastrigin’s function. It has many widespread local minima. The location of the minima are regularly distributed [11]. It is
given by:

\[ f(x) = \left( \sum_{i=1}^{n} \frac{x_i^2}{4000} \right) - \left( \prod_{i=1}^{n} \cos \left( \frac{x_i}{\sqrt{i}} \right) \right) + 1 \]  

with \(-600.0 \leq x_i \leq 600.0\) and global minimum given by \(f(0, \ldots, 0) = 0\).

Figure 10 illustrates the results for a 32-dimensional Griewank function.

DE minimizes the function so quickly that the value of \(\ln(\text{minimum})\) soon cannot be evaluated (\(\ln(0) = \text{undefined}\)). The second best performing algorithm is CDE\(_4\), with PSO and CDE\(_8\) performing third best and fourth best, respectively. CDE\(_8\) performs almost exactly the same as PSO, except for the fact that it converges slightly faster and to a slightly worse local minimum. CDE\(_16\) is fifth best (almost exactly equivalent to CDE\(_4\)) and CPSO\(_{16}\) performs worst.

Figure 11 illustrates results for the 512-dimensional version of the Griewank function.

DE has the worst performance of all the algorithms. CPSO\(_{16}\) performs second worst with PSO performing slightly better. The CDE\(_8\) algorithm is the best performer, actually doing better than on the 32-dimensional version of Griewank. CDE\(_4\) performs second best. CDE\(_{16}\) also performs well in comparison to PSO, CPSO\(_{16}\) and DE. This underlines how much more powerful the DE algorithm is when in cooperative mode.

Figure 12 shows the 2048-dimensional results.

The DE algorithm completely stagnates, once more. The PSO algorithm stagnates almost immediately afterwards. CDE\(_8\) continues to perform well, doing better than all the other algorithms on this high dimensional problem. CDE\(_{16}\) performs second best and CDE\(_4\) is third with CPSO\(_{16}\) performing fourth best.

5) The Ackley Function: This is a widely used multi-modal test function given by:

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

with \(-30.0 \leq x_i \leq 30.0\) and global minimum given by \(f(0, \ldots, 0) = 0\).

Results for the 32-dimensional version of the problem are illustrated in figure 13.

DE performs best, PSO is second best, CDE\(_4\) is third best, CDE\(_{16}\) is fourth best, CDE\(_{16}\) is fifth and CPSO\(_{16}\) is the worst performer. All the algorithms, however, reach high levels of accuracy.

Increasing the dimensionality to 512, the results illustrated in figure 14 were acquired.

Once again, the CDE\(_k\) do comparatively better than the other algorithms on the higher dimensional problem, with CDE\(_8\) performing best, CDE\(_4\) performing second best and CDE\(_{16}\) performing third best. The other 3 algorithms seem to have stagnated at almost the same point but CPSO16 does slightly better than PSO, with DE performing worst.

At dimensionality 2048, the results in figure 15 were acquired.
CDE_8 is the best performer here. CDE_16 does better than CDE_4, but the latter algorithm is still minimizing in a near-linearly decreasing fashion. CPSO_16 performs fourth best, doing slightly better than PSO, with DE the worst performer.

6) The Michalewicz Function: The Michalewicz function is a multi-modal test function (with \( n! \) local optima). The parameter \( m \) defines the “steepness” of the valleys or edges. Larger \( m \) leads to more difficult search. For very large \( m \) the function behaves like a needle in a haystack [11]. It is defined by:

\[
f(x) = -\sum_{i=1}^{n} \sin(x_i) \left( \sin \left( \frac{i \cdot x_i^2}{\pi} \right) \right)^{2 \cdot m} \quad (11)
\]

with \( m = 10; n = 10; 0 \leq x_i \leq \pi \) and global minimum given by \( f(?, \ldots, ?) = -9.66 \). Figure 16 shows the result of minimizing the Michalewicz function using the parameter settings as described above. The graph shows the minimum value attained plotted against the number of iterations. This is because \( \ln(x) = \text{undefined} \) for \( x \leq 0 \). Also, the number of iterations in this instance was set at 5000 (as opposed to 10000 for the other problems).

The DE algorithm performs best here, with CDE_8 coming very close to it in performance. The PSO algorithm is only slightly better than the CPSO_5 with both algorithms performing third and fourth best, respectively.

### VI. Conclusion

This paper investigated the benefits of cooperation as applied to the differential evolution algorithm. The result is a new algorithm, Cooperative Differential Evolution (Cooperative DE), which uses differential evolution within a cooperative split algorithm framework originally developed for PSO. The Cooperative DE algorithm has been shown, in this paper, to perform satisfactorily in comparison to a non-cooperative DE or PSO on a diverse range of benchmark problems for small dimensions. It, however, scales much better than its non-cooperative counterparts and outperforms them on higher dimensional, more complex versions of the same problems.

Looking at the relationship between the number of participants and the level of accuracy achieved, the Cooperative DE with eight participants did better than the four participant version in almost all the tests. It also performed better than any of the other algorithms in almost all the high dimensional tests. A possible reason for this is that, considering the total number of individuals that was used, CDE_8 achieved the best balance between cooperation and population size of the participant algorithms. More weight is given to this argument when it is considered that CDE_16 outperformed CDE_4, as well, in most
of the tests. Possibly, a larger size for the subpopulations in CDE$_{16}$ would result in better performance compared to an equally scaled CDE$_{8}$. More analysis is required to test this theory.

It would, thus, seem that cooperation is a feasible option for coping with high dimensionality in problem solving using differential evolution. The Cooperative DE algorithm, in general, scales better than its non-cooperative equivalent to better solve higher dimensional problems. It also seems to avoid early convergence on most high dimensional problems.

Further research in this area would involve investigating the effects of different stopping conditions for the cooperative split algorithm on different benchmark problems, to determine how close to the global optimum (optima) the Cooperative DE algorithm can get before it stagnates. Another area would involve determining the effects of noise on the performance of the Cooperative DE. Also, different strategies for splitting the optimization problem and recombining the solutions would need to be developed in order to determine the best setup for the cooperative split algorithm. Lastly, the possibility of developing a cooperative algorithm with heterogeneous subpopulations (specifically EA and PSO subpopulations) would be investigated.

ACKNOWLEDGMENT

The authors would like to thank everyone that contributed towards the completion of this paper.

REFERENCES