Merging and Decomposition Variants of Cooperative Particle Swarm Optimization

New Algorithms for Large Scale Optimization Problems

Jay Douglas
Brock University
St. Catharines, Ontario
jd11ps@brocku.ca

Andries Engelbrecht
Institute for Big Data and Data Science, University of Pretoria
Pretoria, South Africa
engel@cs.up.ac.za

Beatrice Ombuki-Berman
Brock University
St. Catharines, Ontario
bombuki@brocku.ca

ABSTRACT

Many optimization algorithms suffer under the curse of dimensionality - a problem that describes a decrease in performance as the number of problem variables increases. Particle swarm optimization (PSO) is no exception to this performance degradation. Cooperative methods have since been introduced for PSO in order to increase its effectiveness for high dimensional problems by following a divide and conquer approach to large dimensional problems. Performance still suffers for such cooperative PSO (CPSO) variants, mostly due to the dependencies among variables. This paper demonstrates the performance of two new variants of CPSO, namely decomposition and merging cooperative swarm optimization, referred to as DCPSO and MCPSO respectively. The goal of these variants is to improve performance for large scale problems with variable dependencies. Empirical results show that DCPSO and MCPSO were able to perform better than standard CPSO-S_k, CPSO-H_k and a random grouping variant, CCPSO, for specific problem classes. While empirical results show that the best strategy between merging and decomposition is very much problem dependent, the MCPSO generally converged earlier than the DCPSO.

KEYWORDS

Particle swarm optimization, cooperative particle swarm optimization, large-scale optimization, random grouping, merging, decomposition

ACM Reference Format:
Jay Douglas, Andries Engelbrecht, and Beatrice Ombuki-Berman. 2018. Merging and Decomposition Variants of Cooperative Particle Swarm Optimization: New Algorithms for Large Scale Optimization Problems. In *Proceedings of 2018 2nd International Conference on Intelligent Systems (ISMSI 2018),* Andries Engelbrecht and Beatrice Ombuki-Berman (Eds.). ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The merit of any meta-heuristic comes from its ability to provide reasonably good solutions for a taxonomy of optimization problems. While most meta-heuristics will have little issue optimally solving lower dimensional problems, one class of problems that becomes increasingly difficult to optimize are large scale optimization problems (LSOPs). Examples of LSOPs can be seen in the training of support vector machines (SVMs) [14] or deep neural networks (DNNs) [10] where there is a large number of decision variables to optimize. Algorithms optimizing these problems are prone to suffering from the "curse of dimensionality", seen in the deterioration of performance as the number of decision variables increases. An additional issue that arises with LSOPs is the existence of variable dependencies, meaning that certain variables are reliant on the values of other variables, and thus should be optimized together. While variable dependencies can exist for smaller dimensional problems, the grouping of related variables is not an issue, as smaller dimensional problems do not suffer from the curse of dimensionality, allowing the algorithms to evaluate all decision variables together. Since the position update of an entire decision vector for LSOPs is often not optimal in regards to performance due to the curse of dimensionality, the problem space is often separated into smaller sub-spaces, thus certain related variables may be separated. As such, it is necessary for algorithms optimizing LSOPs to address both the curse of dimensionality as well as variable dependency.

Different approaches to LSOPs have been introduced, including both problem decomposition and non-decomposition based methods [6]. The most notable of the two is the divide-and-conquer approach which uses cooperative coevolution (CC) algorithms to decompose the search space into smaller sub-spaces. The first example of decomposition for high dimensional problems was the cooperative coevolutionary genetic algorithm (CCGA) proposed by Potter and De Jong [7]. The CC framework has since been adapted to various meta-heuristics such as genetic algorithms (GAs), differential evolution (DE), and particle swarm optimization (PSO). The cooperative particle swarm optimization (CPSO), proposed by Van den Bergh and Engelbrecht [11], applies the divide-and-conquer approach to PSO. The generalized CPSO-S_k algorithm in [11] divides an n-dimensional problem amongst k sub-swarms, while the hybrid CPSO-H_k algorithm alternates between running as a PSO and CPSO-S_k with information exchange between the two. While proving to be an efficient way to combat the curse of dimensionality for separable problems, the static groupings of CPSO-S_k and CPSO-H_k saw worse performance for problems with higher degrees of variable dependence. As a result, variants of CPSO have been developed to improve performance under the presence of variable

ISMSI 2018, March 2018, Phuket, Thailand

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

dependencies, such as the random grouping CCPSO algorithm proposed by Yang *et. al* [5], which randomly permutates dimensions every iteration amongst the sub-swarms.

The purpose of this paper is to propose two new variants, namely the decomposition CPSO (DCPSO) and merging CPSO (MCPSO). These algorithms address the curse of dimensionality problem by using the same divide-and-conquer approach as seen in CPSO-S$_k$, as well as variable dependencies by iteratively restructuring the sizes of sub-swarms, allowing for various groups of variables to be optimized together. DCPSO and MCPSO are initialized as a PSO and CPSO-S algorithm, respectively, and either decompose or merge swarms throughout optimization. Approaches to the design issues associated with these algorithms are highlighted, such as when to restructure swarms, which swarms to restructure, and how to restructure them. An empirical analysis is performed to compare the proposed algorithms with the original CPSO-S$_k$ and CPSO-H$_k$ algorithms, as well the random grouping approach.

Section 2 provides an overview of vanilla PSO, as well as various cooperative variants. DCPSO and MCPSO are introduced and explained in section 3. Section 4 outlines the experimental details used to compare the performance of the various CPSO algorithms. A discussion of the results given in section 5.

2 BACKGROUND

This section covers the background knowledge directly related to the proposed variants developed in this paper. Topics include the standard PSO in section 2.1, and the cooperative PSO and its variants in section 2.2.

2.1 Particle Swarm Optimization

Particle swarm optimization (PSO) is a population-based stochastic algorithm designed to solve continuous-valued optimization problems. PSO is inspired by bird flocking behavior [4].

PSO consists of a swarm or collection of particles, initially randomly dispersed throughout the problem space. Each particle represents a candidate solution to the optimization problem. For every iteration, each particle updates its position according to its own current velocity, the best position it has ever found, and the neighbourhood best position ever found by the particle's neighborhood.

Algorithm 1 outlines the basic approach to vanilla PSO, where the objective function value of every particle is updated each iteration along with its velocity and position. More specifically, the position of a particle is updated using

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{1}$$

where the new particle velocity, $\mathbf{v}_i(t+1)$, is defined by

$$\mathbf{v}_i(t+1) = \omega\mathbf{v}_i(t) + c_1\mathbf{r}_{1i}(\mathbf{y}_i(t) - \mathbf{x}_i(t)) + c_2\mathbf{r}_{2i}(\hat{\mathbf{y}}_i(t) - \mathbf{x}_i(t)) \tag{2}$$

where $\mathbf{x}_i$ is the particle's current position, $\mathbf{y}_i$ is the particle's best position, and $\hat{\mathbf{y}}_i$ is the neighborhood best particle position of particle i. $\mathbf{r}_1$ and $\mathbf{r}_2$ are random vectors with each element of the vectors sampled from a uniform distribution in the range [0,1]. The three coefficients, ω, c_1, and c_2 respectively represent the inertia, cognitive, and social weights.

Algorithm 1 Vanilla PSO

```
Initialize an n-dimensional PSO: P
repeat
    for each particle i = 1, ..., s do
        if f(P.x_i) < f(P.y_i) then
            P.y_i = P.x_i
        if f(P.y_i) < f(P.ŷ) then
            P.ŷ = P.y_i
    for each particle i = 1, ..., s do
        Update particle velocity, v_i (Equation 2)
        Update particle position, x_i (Equation 1)
until stopping criterion is met
```

2.2 Cooperative Particle Swarm Optimization

One big problem with vanilla particle swarm optimization is that it suffers in performance when the dimensionality of the problem increases [11]. This issue is formally known as the "curse of dimensionality", a problem common to many stochastic optimization algorithms.

Since vanilla PSO tends to evaluate the objective function value of the position vector as a whole, it is possible to take one step forward and two steps backward [11]. This means that the objective function value can actually move away from the optimal solution, while only one dimension gets closer – an obvious problem if the objective function value is still evaluated as better despite multiple dimensions being unoptimized.

Van den Bergh and Engelbrecht proposed the co-operative PSO (CPSO) [11] to address both the problem above and the curse of dimensionality problem. The first CPSO algorithm implemented by Van den Bergh and Engelbrecht partitioned the n-dimensional problem into n one-dimensional problems, with each dimension being optimized by a single sub-swarm. This CPSO variant was referred to as the split-CPSO (CPSO-S) [2], summarized in Algorithm 2. A major issue with the CPSO-S is that the objective function is defined for the original n-dimensional problem, and not for the one-dimensional sub-problems. As a result, CPSO-S must address the issue of deciding which particle to select from each sub-swarm to be represented in the overall solution, as well as how to evaluate the degree of effectiveness that each sub-swarm has on the overall solution – problems known as particle selection and credit assignment, respectively. A greedy approach to particle selection is to simply select the best particle from each sub-swarm. A simple solution to credit assignment is just to give equal credit to all sub-swarms.

Particle selection and credit assignment are solved by what is known as a context vector ($\mathbf{b}$ in Algorithm 2). Initially, in order to maintain the stochastic nature of particle swarm optimization, the context vector is initialized to randomly selected particles from each sub-swarm for each dimension. This function is formalized by $\mathbf{b}(j, z)$, which concatenates the global best positions from each sub-swarm into an n-dimensional vector, except the jth sub-component, which is replaced by z, where z is the position of any particle from P_j. The resulting vector can thus be quantified by the fitness function as f($\mathbf{b}(j, z)$).

Algorithm 2 CPSO-S

Initialize n one-dimensional PSOs: $P_j, j = 1, \ldots, n$
repeat
  for each sub-swarm $j = 1, \ldots, n$ **do**
    for each particle $i = 1, \ldots, s$ **do**
      if f(**b**$(j, P_j.x_i))$ < f(**b**$(j, P_j.y_i))$ **then**
        $P_j.y_i = P_j.x_i$
      if f(**b**$(j, P_j.y_i))$ < f(**b**$(j, P_j.\hat{y}))$ **then**
        $P_j.\hat{y} = P_j.y_i$
    for each particle $i = 1, \ldots, s$ **do**
      Update particle velocity, $P_j.v_i$ (Equation 2)
      Update particle position, $P_j.x_i$ (Equation 1)
until stopping criterion is met

While CPSO-S assumes one-dimensional sub-swarms, the granularity of the splits can be increased, as is done with CPSO-S$_k$, while still preventing the issue with one step forward, two steps backward. Van den Bergh and Engelbrecht [11] proposed CPSO-S$_k$ to divide an n-dimensional problem space into k parts. The reason for utilizing sub-swarms of dimensions larger than one is to address variable dependencies by ideally grouping related variables together in a single sub-swarm.

A major issue with the CPSO algorithms is that they are prone to stagnation, as a result of only updating a single sub-space at a time [11]. PSO on the other hand does not become stuck in pseudominima for this reason, since PSO evaluates the entire n-dimensional vector upon updating its positions. In an attempt to combine the convergence of CPSO-S$_k$ and the pseudominima resistant properties of PSO, a hybrid approach was proposed, referred to as CPSO-H$_k$ [11]. This algorithm works to interleave both CPSO-S$_k$ and PSO together by running CPSO-S$_k$ for one iteration, followed by one iteration of the PSO algorithm. The hybridization comes from the information exchange that is performed after every iteration of one half of the algorithm: after one iteration of CPSO-S$_k$, the context vector **b** is used to return an n-dimensional vector containing the global best of all sub-swarms, which is then used to overwrite a random particle of the PSO half. Following this, the PSO half is ran for one iteration, resulting in its global best position being split up into sub-vectors of the correct dimensionality and used to replace random particles of each of the j sub-swarms of the CPSO-S$_k$ half.

Note that at no point during information exchange should an algorithm overwrite the global best of the other half of the algorithm. Due to the frequent information exchange, it is possible that the diversity of solutions will stagnate due to the constant sharing of particle positions [11]. In this case, only a portion of the particles from each half of the algorithm should participate in information exchange, thus preserving a portion of each swarm's diversity.

More recently, Yao and Li have combined the CPSO-S$_k$ algorithm with the idea of randomly grouping variables in sub-swarms and using an adaptive weighting scheme for their cooperative coevolving PSO (CCPSO) [5]. These concepts were originally introduced by Yang *et al.* for differential evolution [12] [13]. The advantage of this random regrouping is that it does not require prior knowledge about the variable dependencies. The adaptive weighting is simply used to fine tune the solutions found by the algorithm.

The CCPSO algorithm works by creating k sub-swarms, initialized by randomly assigning dimensions to each sub-component, without overlap between sub-swarms. This random permutation of dimensions is noted to be done as frequently as possible, in order to increase the probability of related variables being grouped together, and thus is done at the start of every iteration. In this way, the first part of CCPSO can be thought of as a CPSO-S$_k$ algorithm which randomly permutates dimensions amongst the sub-swarms at the start of every iteration.

After every iteration of the randomly permutated CPSO-S$_k$ part, a k-dimensional vanilla PSO algorithm is initialized and evolved for a few iterations. Each decision variable in this PSO swarm is associated with one of the k sub-swarms of the CPSO-S$_k$ part. This PSO swarm, known as the weight vector, is evaluated by multiplying each corresponding decision variable by parts of the context vector associated with its assigned sub-swarm. If a better position is found during these few iterations, the context vector is updated with this new position. Note that the bounds of this weight vector will change based on the values of the context vector in order to ensure that when their decision variables are multiplied by the context vector during fitness evaluation, that it is evaluated within a feasible search space. The k-dimensional PSO used to optimize the weight vector does not persist at the end of its few iterations of optimization, and thus should be reinitialized for the next time it is ran.

3 DECOMPOSITION AND MERGING CO-OPERATIVE PARTICLE SWARM OPTIMIZATION

This section proposes two new CPSO variants, developed to address the variable dependency problem experienced by CPSO-S and CPSO-S$_k$. Sections 3.1 and 3.2 respectively introduce and motivate the decomposition CPSO and the merging CPSO, including the approach to updating the personal and global best positions of each sub-swarm as swarms are decomposed and merged. The frequency at which sub-swarms merge or decompose is discussed in section 3.3. Additional design issues are discussed in section 3.4.

3.1 Decomposition Variant

The decomposition CPSO (DCPSO) starts with a single swarm which optimizes the complete n-dimensional problem. Over time, the problem is iteratively decomposed into smaller dimensional problems, with each sub-problem assigned to a sub-swarm. Decomposition occurs until the n-dimensional problem is partitioned into n one-dimensional problems. The DCPSO therefore starts with a vanilla PSO, moving towards a CPSO-S configuration. Refer to figure 1 for an illustration of the DCPSO.

Design issues that are important for the DCPSO are when to further decompose, which sub-swarm(s) to decompose, and how to decompose. For initial iterations, DCPSO is able to excel in exploration through the use of an n-dimensional PSO. By evolving larger subspaces, it is also able to avoid stagnation, since a greater number of sub-components will be evaluated according to the fitness function, making it initially resilient against pseudominima. For later iterations, exploitation of the lower dimensional sub-spaces will result in a higher degree of convergence towards solutions through the use of smaller dimensional sub-swarms to fine-tune the solutions.

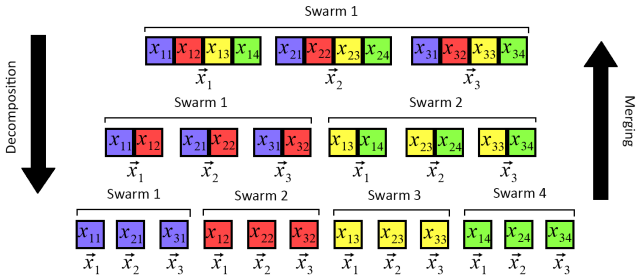

Figure 1: An illustration of the process of decomposition and merging using three particles with positions x_1, x_2, x_3 optimizing dimensions j = 1, 2, 3, 4

Algorithm 3 Decomposition Algorithm

Initialize an n-dimensional PSO: P
repeat
  for each swarm $j = 1, \ldots, k$ **do**
    for each particle $i = 1, \ldots, s$ **do**
      if $f(\mathbf{b}(j, P_j.\mathbf{x}_i)) < f(\mathbf{b}(j, P_j.\mathbf{y}_i))$ **then**
        $P_j.\mathbf{y}_i = P_j.\mathbf{x}_i$
      if $f(\mathbf{b}(j, P_j.\mathbf{y}_i)) < f(\mathbf{b}(j, P_j.\hat{\mathbf{y}}))$ **then**
        $P_j.\hat{\mathbf{y}} = P_j.\mathbf{y}_i$
    for each particle $i = 1, \ldots, s$ **do**
      Update particle velocity, $P_j.\mathbf{v}_i$ (Equation 2)
      Update particle position, $P_j.\mathbf{x}_i$ (Equation 1)
  if decomposition condition is true **then**
    decompose each P_j into n_r sub-swarms
until stopping criterion is met

At each decomposition step, each sub-swarm is divided into n_r sub-swarms, as seen in algorithm 3. This also means that the number of sub-swarms increases by a multiplicative factor of n_r. There are multiple ways to perform this split – one approach is to evenly split each particle position by the n_r factor. However, this approach is naive and does not take into consideration variable dependency. A random approach may also be taken as to randomly select which dimensions will be grouped together in the new sub-swarm. The personal and global best positions will also have to be decomposed for the new sub-swarm. In order to evaluate the overall solution, the context vector, $\mathbf{b}$, must be maintained as done with CPSO-S_k. Ultimately, the context vector will consist of the same values it had before the decomposition, but will have new sub-swarms optimizing each of its dimensions for future iterations.

3.2 Merging Variant

The merging CPSO (MCPSO) works from a different direction compared to DCPSO. The MCPSO starts with n one-dimensional sub-swarms (one sub-swarm per dimension), and over time merges sub-swarms until the complete n-dimensional problem is optimized. Effectively, the MCPSO moves from a CPSO-S configuration to the vanilla PSO. The MCPSO is also illustrated in figure 1.

Algorithm 4 Merging Algorithm

Initialize n one-dimensional PSOs: $P_j, j = 1, \ldots, n$
repeat
  for each swarm $j = 1, \ldots, k$ **do**
    for each particle $i = 1, \ldots, s$ **do**
      if $f(\mathbf{b}(j, P_j.\mathbf{x}_i)) < f(\mathbf{b}(j, P_j.\mathbf{y}_i))$ **then**
        $P_j.\mathbf{y}_i = P_j.\mathbf{x}_i$
      if $f(\mathbf{b}(j, P_j.\mathbf{y}_i)) < f(\mathbf{b}(j, P_j.\hat{\mathbf{y}}))$ **then**
        $P_j.\hat{\mathbf{y}} = P_j.\mathbf{y}_i$
    for each particle $i = 1, \ldots, s$ **do**
      Update particle velocity, $P_j.\mathbf{v}_i$ (Equation 2)
      Update particle position, $P_j.\mathbf{x}_i$ (Equation 1)
  if merging condition is true **then**
    merge every n_r sub-swarms
until stopping criterion is met

As with the DCPSO, the MCPSO has to consider the following design issues: when to merge sub-swarms, which sub-swarms to merge, and how to merge sub-swarms. Unlike DCPSO, the initial iterations for MCPSO will experience very early convergence due to the exploitation of individual dimensions through the early use of CPSO-S. As a trade-off, pseudominima will prove to be a problem for initial iterations. Additionally, variable dependency is not initially taken into consideration as much as would be done in DCPSO. For later iterations, the MCPSO algorithm will be able to combine the solutions of each sub-swarm in order to evaluate them as whole for the n-dimensional problem. During this time, variable dependency will be addressed through larger k values in hopes of fine tuning the merged solutions.

As for the DCPSO, the n_r factor will specify how the algorithm will merge swarms. More specifically, for MCPSO, every n_r swarms will be merged into one larger dimensional sub-swarm, as seen in algorithm 4. Each time MCPSO merges swarms, there is a divisional factor of n_r sub-swarms left, each with n_r times more dimensions than they had last time. One approach for merging is to simply merge whole swarms together, which would combine previous variable groupings with possible new variable relationships. A more random approach might be made by randomly choosing dimensions to merge. Regardless, the personal and global best positions will also have to be merged in an identical way to the method used for particle positions. The context vector of the overall solution must be maintained similar to DCPSO, where the actual context vector is left unchanged.

3.3 Frequency of Merging / Decomposition

The rate at which decomposition or merging occurs is an important control parameter, with a significant impact on the performance of the algorithms. The frequency should be selected to ensure that sufficient time is allocated to both exploring and exploiting by considering all dependencies among variables, and by focusing on dimensions individually. In this paper, frequency of decomposition / merging is calculated using:

$$n_f = \frac{n_T}{1 + (\frac{log(n_x)}{log(n_r)})} \quad (3)$$

where n_T is the total number of iterations, n_f is the number of iterations to wait before merging or decomposing, n_x is the total number of dimensions, and n_r is the number of swarms to be joined together, or split. This formula will allow the swarms to restructure evenly throughout the iterations, which is best when no information about the problem space is known a priori, since it gives each sub-swarm grouping the same amount of time to perform on the problem space.

3.4 Additional Considerations

One common parameter between DCPSO and MCPSO is the n_r value, noted in equation (3), which determines the number of resulting sub-swarms when splitting with DCPSO, and the number of sub-swarms to merge with MCPSO. Regardless of how the restructuring is performed, this n_r value may be adjusted in order to affect how much change is seen at the restructure iteration in regards to exploration and exploitation. Depending on performance, the algorithm may either want to make quicker or more gradual steps towards certain types of search associated with different k values. For higher n_r values, the algorithm is either splitting into or merging together more dimensions than it would compared to lower values, where the lowest possible n_r value is two. This parameter will have more flexibility for higher dimensional problems, as there is a greater number of dimensions that the algorithm can either split or merge together.

4 EXPERIMENTAL SETUP

This section describes the empirical process followed to analyse the performance of the proposed CPSO algorithms. The performance measures monitored for each of the algorithms are given in section 4.1. Section 4.2 provides an overview of the statistical tests used to verify results. The test problems are highlighted in section 4.3. The parameters of each algorithm are detailed in section 4.4.

4.1 Performance Measures

In order to quantify the performance of each algorithm, the quality of the global best particle at the end of the optimization process was used. This measure reflects the accuracy of the algorithms.

4.2 Statistical Methods

Each algorithm was executed for 30 independent runs on each benchmark problem. A Mann-Whitney U-test was performed on every pair of algorithms for each problem at a 95% confidence level in order to establish whether or not the differences were statistically significant or not. The test's outcomes were used to compute the total number of wins and losses for each algorithm over different problem classes. The algorithms were then ranked based on the difference between their wins and losses.

4.3 Benchmark Problems

The problems used in this paper are a combination of unimodal, multimodal, separable, non-separable, and rotated problems, listed in table 1. Separable problems have no dependencies among their decision variables, allowing optimization of each decision variable in isolation. On the other hand, non-separable problems have at least two decision variables that are dependent on one another. The difficulty in solving non-separable problems increases in the number of dependencies and whether these dependencies are linear or non-linear. Variable dependencies can be introduced by rotating the decision space, thereby turning each of the separable problems into non-separable problems [3] [9]. Problem dimensionality varied between 30, 500, and 1000 dimensions in order to test performance for both high and low dimensional problems.

Table 1: Test Functions

Test Function, f(x)	Domain	Class
Absolute Value, f_1	[-100 .. 100]	Unimodal Separable
Ackley, f_2	[-30 .. 30]	Multimodal Non-separable Rotated
Egg Holder, f_3	[-512 .. 512]	Multimodal Non-separable
Elliptic, f_4	[-100 .. 100]	Unimodal Separable Rotated
Generalized Griewank, f_5	[-600 .. 600]	Multimodal Non-separable Rotated
Hyperellipsoid, f_6	[-5.12 .. 5.12]	Unimodal Separable
Michalewicz, f_7	[0 .. π]	Multimodal Separable
Norwegian, f_8	[-1.1 .. 1.1]	Multimodal Non-separable
Quadric, f_9	[-100 .. 100]	Unimodal Non-separable
Quartic, f_{10}	[-1.28 .. 1.28]	Unimodal Separable
Generalized Rastrigin, f_{11}	[-5.12 .. 5.12]	Multimodal Separable Rotated
Generalized Rosenbrock, f_{12}	[-30 .. 30]	Multimodal Non-separable Rotated
Salomon, f_{13}	[-100 .. 100]	Multimodal Non-separable
Schaffer 6, f_{14}	[-100 .. 100]	Multimodal Non-separable
Schwefel 1.2, f_{15}	[-100 .. 100]	Unimodal Non-separable Rotated
Schwefel 2.21, f_{16}	[-100 .. 100]	Unimodal Separable
Spherical, f_{17}	[-5.12 .. 5.12]	Unimodal Separable
Step, f_{18}	[-100 .. 100]	Multimodal Separable
Vincent, f_{19}	[0.25 .. 10]	Multimodal Separable
Weierstrass, f_{20}	[-0.5 .. 0.5]	Multimodal Separable

4.4 Control Parameters

Each algorithm was executed on each benchmark problem for 10000 fitness function evaluations. The swarm parameters of the algorithms are listed in table 2.

4.4.1 CPSO-S_k Parameters. The parameters in table 2 used by CPSO-S_k were chosen according to Van den Bergh and Engelbrecht [1], which shows that these parameters lead to convergence and performed empirically well. $\mathbf{r_1}$ and $\mathbf{r_2}$ were kept random in order to maintain diversity in the population [8].

4.4.2 CPSO-H_k Parameters. Both the CPSO-S_k part and the PSO part of the hybrid CPSO-H_k algorithm utilize the same parameters as given in table 2, aside from the k-value for PSO, which only utilizes a single swarm of 10 particles. Information exchange is limited to only half the particles of each algorithm, as recommended by Van den Bergh and Engelbrecht [11]. This exchange is performed at the end of each iteration for both halves of the algorithm.

4.4.3 CCPSO Parameters. The random variant shares the same parameters as mentioned in table 2, although the decision variables that each sub-swarm optimizes are randomly regrouped amongst the sub-swarms. At the end of every iteration of the CPSO-S_k part of CCPSO, a new k-dimensional vanilla PSO algorithm is initialized and ran for three iterations which shares the ω, c_1, c_2, and number of particles given in table 2.

4.4.4 DCPSO Parameters. Due to the fact that each swarm eventually splits into additional sub-swarms, the k-value of DCPSO naturally changes, as does the total number of particles in the problem space, because each sub-swarm will have 10 particles. The values used for ω, c_1, and c_2 can be found in table 2. The number of splits that occur in each sub-swarm, n_r, is set to two. The DCPSO is initialized as an n-dimensional PSO algorithm, and restructures evenly over the specified maximum number of function evaluations, until DCPSO eventually becomes a CPSO-S.

4.4.5 MCPSO Parameters. The MCPSO parameters are listed in table 2 and follow closely to those seen in DCPSO, aside from the initialization of the algorithm, which begins as CPSO-S. n_r in the context of merging refers to how many sub-swarms are merged during the restructure iteration, and is also set to two. This algorithm also restructures evenly over the maximum number of function evaluations, ending its last fitness evaluations as a PSO algorithm containing one n-dimensional swarm.

Table 2: Swarm Parameters

Parameter	Value
# of sub-swarms (k)	6
# of particles per sub-swarm	10
ω	0.729844
c_1	1.496180
c_2	1.496180

5 EXPERIMENTAL RESULTS DISCUSSION

This section analyses the performance of the five algorithms on the problems discussed in section 4.3. The sections that follow respectively considers the 30, 500, and 1000 dimensional problems.

5.1 30 Dimensions

Table 3 shows mixed results for 30-dimensional problems. The CPSO-S_k ranked the best for separable unimodal problems. The CCPSO algorithm was found to be the best performing algorithm for rotated unimodal problems. DCPSO ranked first for non-separable unimodal, separable multimodal, and rotated multimodal problems. MCPSO tied for first alongside DCPSO for separable multimodal problems, along with being the best performing algorithm for non-separable multimodal problems. On average, DCPSO had the highest rank across all problem classes. For these lower dimensional problems, CPSO-S_k ranked as the best algorithm for separable unimodal and multimodal problem classes. Problems with variable dependencies were best optimized by DCPSO.

Table 3: Paired Mann-Whitney U Test for problems with 30 dimensions (S: Separable, NS: Non-separable, R: Rotated).

Algorithm	Result	Problem Classes					
		Unimodal			Multimodal		
		S	NS	R	S	NS	R
CPSO-S_k	Wins	17	1	2	5	8	2
	Losses	4	0	4	10	10	4
	Difference	13	1	-2	-5	-2	-2
	Rank	1	2	3	2	3	4
	S Rank	2					
	NS Rank	3					
CPSO-H_k	Wins	14	0	0	1	2	3
	Losses	8	1	3	15	20	4
	Difference	6	-1	-3	-14	-18	-1
	Rank	2	4	4	3	5	3
	S Rank	2.5					
	NS Rank	4					
CCPSO	Wins	5	0	7	5	6	9
	Losses	18	4	0	10	18	7
	Difference	-13	-4	7	-5	-12	2
	Rank	5	5	1	2	4	2
	S Rank	3.5					
	NS Rank	3					
DCPSO	Wins	8	4	3	14	18	7
	Losses	13	0	2	2	5	2
	Difference	-5	4	1	12	13	5
	Rank	4	1	2	1	2	1
	S Rank	2.5					
	NS Rank	1.5					
MCPSO	Wins	11	2	1	14	20	2
	Losses	11	2	4	2	3	6
	Difference	0	0	-3	12	17	-4
	Rank	3	3	4	1	1	5
	S Rank	2					
	NS Rank	3.25					

5.2 500 Dimensions

Results for the larger 500-dimensional problems listed in table 4 begin to show better performance for algorithms which attempt to address variable dependency. The random approach by CCPSO was the best ranking algorithm for non-separable and rotated unimodal functions, while DCPSO performed the best for rotated multimodal problems. MCPSO maintained the highest win-loss ratio for separable unimodal, separable multimodal, and non-separable multimodal problems. For all 500-dimensional problem classes, DCPSO and MCPSO tied as the best performing algorithms in regards to their average rank across all tested problem types. MCPSO ranked the best for both of the separable problem classes. For problems with variable dependency, it was noted that CCPSO and DCPSO tied in their average rank across these non-separable problems.

Table 4: Paired Mann-Whitney U Test for unrotated problems with 500 dimensions (S: Separable, NS: Non-separable, R: Rotated).

Algorithm	Result	Problem Classes					
		Unimodal			Multimodal		
		S	NS	R	S	NS	R
CPSO-S_k	Wins	5	2	2	7	7	3
	Losses	19	6	6	13	20	10
	Difference	-14	-4	-4	-6	-13	-7
	Rank	4	4	3	3	4	4
	S Rank	3.5					
	NS Rank	3.75					
CPSO-H_k	Wins	1	0	0	2	1	0
	Losses	21	8	8	18	26	15
	Difference	-20	-8	-8	-16	-25	-15
	Rank	5	5	4	5	5	5
	S	5					
	NS Rank	4.75					
CCPSO	Wins	14	8	8	6	14	12
	Losses	10	0	0	14	14	4
	Difference	4	8	8	-8	0	8
	Rank	3	1	1	4	3	2
	S Rank	3.5					
	NS Rank	1.75					
DCPSO	Wins	17	6	5	16	22	13
	Losses	5	2	3	2	5	2
	Difference	12	4	2	14	17	11
	Rank	2	2	2	2	2	1
	S Rank	2					
	NS Rank	1.75					
MCPSO	Wins	20	4	5	17	24	8
	Losses	2	4	3	1	3	5
	Difference	18	0	2	16	21	3
	Rank	1	3	2	1	1	3
	S Rank	1					
	NS Rank	2.25					

5.3 1000 Dimensions

After further increasing the number of dimensions to 1000, the performance of each algorithm noted in table 5 seem to follow a similar trend as seen with 500 dimensions. CCPSO remained the best performing algorithm for non-separable and rotated unimodal problems. DCPSO ranked best for all separable, non-separable and rotated multimodal problems. MCPSO performed the best for separable unimodal problems, however tied with DCPSO for best performing algorithm of the separable multimodal problems. When averaging the ranks over all problem classes, it was ultimately DCPSO that ranked the highest. DCPSO managed to pull ahead of CCPSO as the best performing algorithm across all problems with variable dependencies.

Table 5: Paired Mann-Whitney U Test for unrotated problems with 1000 dimensions (S: Separable, NS: Non-separable, R: Rotated).

Algorithm	Result	Problem Classes					
		Unimodal			Multimodal		
		S	NS	R	S	NS	R
CPSO-S_k	Wins	5	1	2	6	7	3
	Losses	19	7	6	14	23	12
	Difference	-14	-6	-4	-8	-16	-9
	Rank	4	5	3	2	4	4
	S Rank	3					
	NS Rank	4					
CPSO-H_k	Wins	2	2	0	3	2	0
	Losses	21	5	8	17	26	15
	Difference	-19	-3	-8	-14	-24	-15
	Rank	5	4	4	3	5	5
	S Rank	4					
	NS Rank	4.5					
CCPSO	Wins	14	8	8	6	14	12
	Losses	10	0	0	14	14	4
	Difference	4	8	8	-8	0	8
	Rank	3	1	1	2	3	2
	S Rank	2.5					
	NS Rank	1.75					
DCPSO	Wins	17	5	5	17	23	13
	Losses	5	2	3	2	3	2
	Difference	12	3	2	15	20	11
	Rank	2	2	2	1	1	1
	S Rank	1.5					
	NS Rank	1.5					
MCPSO	Wins	20	3	5	17	22	10
	Losses	3	5	3	2	4	5
	Difference	17	-2	2	15	18	5
	Rank	1	3	2	1	2	3
	S Rank	1					
	NS Rank	2.5					

5.4 Summary of Results

Overall, each of the problem classes tested had a specific best performing algorithm when considering all 30, 500, and 1000 dimensional problems. For the separable unimodal problem class, MCPSO had the best average rank. Non-separable unimodal problems were best optimized by the DCPSO algorithm. CCPSO managed to rank first for rotated unimodal problems across all dimensions. Both the separable and non-separable problem classes were best optimized by MCPSO. The MCPSO algorithm additionally proved to be the best performing algorithm for rotated multimodal problems, which similarly to CCPSO, was not outranked across any of the dimensions tested.

6 CONCLUSION

This paper proposed two new cooperative particle swarm optimization (CPSO) algorithms known as decomposition CPSO (DCPSO) and merging CPSO (MCPSO) respectively, which attempt to address variable dependencies for high dimensional problems. The idea of changing the number of decision variables in a sub-swarm offers a new approach to grouping related variables together that may be compared to other CPSO variants that attempt to address the same issue of variable dependencies. Additionally, experiments were performed to empirically analyse the performance of each algorithm against other CPSO variants for a wide variety of problem classes. The comparative algorithms included the generalized CPSO-S_k algorithm which divides n-dimensional problems into k sub-swarms, the hybrid CPSO-H_k algorithm which runs as both a PSO and CPSO-S_k, and the random approach by CCPSO which works to randomly permutate decision variables amongst the sub-swarms. Results showed that both the DCPSO and MCPSO were able to not only outperform both CPSO-S_k and CPSO-H_k for large-scale optimization problems (LSOPs), but also were able to outrank the random CCPSO variant for certain problem classes. Trade-offs between DCPSO and MCPSO were noted, specifically in regards to higher dimensional problems, where DCPSO ranked the best for non-separable problems, while MCPSO remained the top performer for separable problems. It is hypothesized that the initial optimization as a CPSO-S algorithm seen in MCPSO was able to perform well due to the lack of variable dependencies for the separable problems. Since DCPSO initially optimizes as an n-dimensional PSO algorithm, it was able to group related variables found in non-separable problems at the start of its iterations, resulting in superior performance overall. This suggests that the number of decision variables in each sub-swarm at the start of iterations is an important factor when considering the problem class being optimized.

Currently, the proposed DCPSO and MCPSO variants statically regroup every sub-swarm over a set number of iterations. Future work should investigate different decomposition and merging strategies in order to best address variable dependencies while maintaining performance for higher dimensional problems. Experiments detailing the performance of the decomposition and merging variants should include additional performance measures, allowing for a better understanding of how exactly DCPSO and MCPSO traverse the search space. Additional cooperative algorithms should also be introduced to further establish the best performing algorithm for LSOPs. Due to the success in scalability of the proposed algorithms, the dimensionality of problems may further be increased to even higher dimensions and degrees of variable dependency.

7 ACKNOWLEDGEMENTS

This work is based in part on the research supported by the National Research Foundation (NRF) of South Africa (Grant Number 46712). The opinions, findings and conclusions or recommendations expressed in this article is that of the author(s) alone, and not that of the NRF. The NRF accepts no liability whatsoever in this regard.

REFERENCES

[1] F Bergh and Andries P Engelbrecht. 2001. Effects of swarm size on cooperative particle swarm optimisers. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 892–899.

[2] Van Den Bergh and Andries Petrus Engelbrecht. 2000. Cooperative Learning in Neural Networks using Particle Swarm Optimizers. In *South African Computer Journal*, Vol. 26. Citeseer, 84–90.

[3] Antony W Iorio and Xiaodong Li. 2006. Rotated Test Problems for Assessing the Performance of Multi-objective Optimization Algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. ACM, Seattle, WA, 683–690.

[4] James Kennedy. 2011. Particle Swarm Optimization. In *Encyclopedia of Machine Learning*. Springer, 760–766.

[5] Xiaodong Li and Xin Yao. 2009. Tackling High Dimensional Nonseparable Optimization Problems by Cooperatively Coevolving Particle Swarms. In *IEEE Congress on Evolutionary Computation, 2009. CEC 2009*. IEEE, Trondheim, Norway, 1546–1553.

[6] Sedigheh Mahdavi, Mohammad Ebrahim Shiri, and Shahryar Rahnamayan. 2015. Metaheuristics in large-scale global continues optimization: A survey. *Information Sciences* 295 (2015), 407–428.

[7] Mitchell A Potter and Kenneth A De Jong. 1994. A Cooperative Coevolutionary Approach to Function Optimization. In *Parallel Problem Solving from Nature – PPSN III*. Springer, 249–257.

[8] K Premalatha and AM Natarajan. 2009. Hybrid PSO and GA for Global Maximization. *Int. J. Open Problems Compt. Math* 2, 4 (2009), 597–608.

[9] Ralf Salomon. 1996. Re-evaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions. A Survey of Some Theoretical and Practical Aspects of Genetic Algorithms. *BioSystems* 39, 3 (1996), 263–278.

[10] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[11] Frans Van den Bergh and Andries P Engelbrecht. 2004. A Cooperative Approach to Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation* 8, 3 (2004), 225–239.

[12] Zhenyu Yang, Ke Tang, and Xin Yao. 2007. Differential Evolution for High-dimensional Function Optimization. In *2007 IEEE, Congress on Evolutionary Computation*. IEEE, Singapore, 3523–3530.

[13] Zhenyu Yang, Ke Tang, and Xin Yao. 2008. Large Scale Evolutionary Optimization using Cooperative Coevolution. *Information Sciences* 178, 15 (2008), 2985–2999.

[14] Zhi-Qiang Zeng, Hong-Bin Yu, Hua-Rong Xu, Yan-Qi Xie, and Ji Gao. 2008. Fast training Support Vector Machines using parallel sequential minimal optimization. In *Intelligent System and Knowledge Engineering, 2008. ISKE 2008. 3rd International Conference on*, Vol. 1. IEEE, 997–1001.