Comparative Study On Cooperative Particle Swarm Optimization Decomposition Methods for Large-scale Optimization

Mitchell D.Clark

Submitted in partial fulfillment of the requirements for the degree of

Master of Science

Faculty of Mathematics and Science, Brock University,
St. Catharines, Ontario

Acknowledgements

I would like to thank Dr. B. Ombuki-Berman for her continued guidance throughout this thesis. I would also like to thank Dr. A.P. Engelbrecht for his continued support on this topic. Thank you as well to the department of Computer Science at Brock University for supplying me with the means and knowledge necessary to complete this paper.

Abstract

The vast majority of real-world optimization problems can be put into the class of large-scale global optimization (LSOP). Over the past few years, an abundance of cooperative coevolutionary (CC) algorithms has been proposed to combat the challenges of LSOP's. When CC algorithms attempt to address large scale problems, the effects of interconnected variables, known as variable dependencies, causes extreme performance degradation. Literature has extensively reviewed approaches to decomposing problems with variable dependencies connected during optimization, many times with a wide range of base optimizers used. In this thesis, we use the cooperative particle swarm optimization (CPSO) algorithm as the base optimizer and perform an extensive scalability study with a range of decomposition methods to determine ideal divide-and-conquer approaches when using a CPSO. Experimental results demonstrate that a variety of dynamic regrouping of variables, seen in the merging CPSO (MCPSO) and decomposition CPSO (DCPSO), as well varying total fitness evaluations per dimension, resulted in high-quality solutions when compared to six state-of-the-art decomposition approaches.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Large-scale global optimization (LSOP) problems, have been extensively researched using meta-heuristics such as genetic algorithms (GA) [1], particle swarm optimization (PSO) [2], Artificial Bee Colony Optimization (ABC) [3], Ant Colony Optimization (ACO) [4], and differential evolution (DE) [5], to name a few. LSOP's can be found in a wide variety of problems such as artificial neural network (ANN) training, universe exploring [6], and building retrofitting [7]. LSOPs remain a popular field of study, due to the characteristic of, *the curse of dimensionality* [8] and their increasing prevalence in real-world issues. The curse of dimensionality, is the increase in difficulty of finding an optimal solution, due to the added number of decision variables. As large scale problems increase in size, so too does the complexity. One such complexity, known as variable dependencies, is the interconnected decision variables, known as *non-separable*, *separable* or *partially separable*.

Non-separable problems are those in which all variables are inter-connected and require optimization concurrently. Separable problems are problems with fully unlinked variables and can be optimized individually. Finally, partially separable problems combine separable and non-separable variables that compose the optimization problem. Variable dependencies have less impact to the performance of meta-heuristic algorithms, due to the variables being optimized together in a group which address the variable dependencies automatically. However, this technique suffers greatly from the curse of dimensionality.

Cooperative coevolution (CC) [9], a divide-and-conquer approach to address LSOPs, proves to be an invaluable technique on problem sets with mainly separable variables [10]. This CC approach, first synthesised in a GA, decomposes the decision variables into sub-sections, thus minimizing the curse of dimensionality. This CC approach inspired the development of other CC based meta-heuristics, including cooperative particle swarm optimization (CPSO) [11], Ant Colony Optimization, DE, firefly, etc [12]. Despite the growing interest in cooperative approaches, extensive consideration for variable dependencies has not been fully evaluated in CPSO. This technique, however, is valuable, as it minimizes curse of dimensionality while addressing variable dependencies.

Although various decomposition approaches exist in the literature [13, 14, 15, 16, 11], their suitability has not been fully evaluated for various meta-heuristic algorithms. A decomposition approach should be optimizer

independent for wider applicability. However, most of the recent competitive decomposition approaches have been synthesized within the context of self-adaptive differential evolution with neighborhood search algorithm (SaNSDE) [17, 18, 19], due to its noticeable performance advancements over non-differential evolution-based cooperative coevolutinary meta-heuristics. Thus, to the author's knowledge, there is lack of comprehensive comparison of the suitability of a given decomposition approach across various meta-heuristic optimizers. To narrow this gap, this thesis provides a comparative study of various decomposition approaches within a PSO optimizer. This thesis aims to address this by extending on our previous research of Douglas et al. where two new decomposition approaches were proposed [20].

In this thesis, using a Cooperative PSO as the base optimizer, we compare various decomposition approaches originally proposed for different meta-heuristic algorithms found in the literature. The decomposition approaches include differential grouping [15], dynamic [2, 20], self-learning [14], variable clustering [13] and static [21]. The decomposition approaches analyzed are all state-of-the-art techniques, which have been demonstrated as the best performers within their respective meta-heuristic optimizer in the literature. Additionally to the best of the author's knowledge, this is the first study providing a scalability across a wide range of dimensions.

As large scale optimization problems scale to larger dimensions, so does the number of variable dependencies. A variety of decomposition and non-decomposition approaches have been implemented to combat the increas-

ing number of variable dependencies. Unlike decomposition methods, which break problems down into small sub-sets, non-decomposition approaches often focus on operators, hybridization, and parallel operations to address such problems [22]. Some of the common non-decomposition approaches are variants of, PSO, GA, ACO, and firefly algorithm variants (FFA), to name a few.

As mentioned above, decomposition methods utilize the divide-and-conquer approach, breaking the original problem into many small sub-problems to be optimized irrespective of the others. Static grouping based methods, originally seen in the context of cooperative GA [9] and CPSO [11], maintain sub-groups throughout the optimization. This methodology in CPSO was known as CPSO-S and CPSO-Hk [11]. Each of these algorithms statically divides the n-dimensional problem into k sub-problems. The CPSO-S algorithm divides the problem into 1-dimensional problems, each solved by varying swarms. The CPSO-Hk algorithm alternates between the meta-heuristic algorithm PSO and the CPSO-S algorithm, to account for variable dependencies. Early work from Liu *et al.* began the cooperative co-evolutionary trend, when results showed impressive results using this framework [23]. Cooperative DE (CCDE) was then proposed and showed increased performance over other non-decomposition based approaches [12]. Building off this success, a cooperative ABC algorithm was proposed [24], followed by a cooperative Firefly algorithm [25]. All algorithms showed exceptional performance when compared to other decomposition and non-decomposition based approaches.

These early approaches all lacked a dynamic component, which would give the algorithm the ability to link inter-connected variables together throughout the optimization. The CCPSO algorithm was the first cooperative PSO to implement such an approach, inspired by the CPSO-Sk [11]. The CCPSO alters the decision variables randomly throughout the run [26]. Following the random regrouping approach, the merging CPSO (MCPSO) and decomposition CPSO (DCPSO) approaches were proposed, both algorithms decomposed the variables over some fixed number of iterations into sub-swarms half the size each split, or twice the size for DCPSO and MCPSO, respectively [2]. The DCPSO begins as a single n-dimensional PSO and is split into sub-swarms half the size ending in n single dimension sub-swarms; the MCPSO is the opposite. Following the success of the random grouping approach, the random adaptive grouping algorithm was proposed [21], where the problem is decomposed into m s-dimensional sub-components, where $m \cdot s = n$ and m and s are fixed throughout the run. Rather than changing the decomposition sub-group sizes, the RAG algorithm selects the worst-performing $m/2$ sub-groups, and randomly shuffles them in hopes of finding linked variables.

Based on the early decomposition approaches, the recursive differential grouping (RDG) algorithm was proposed [15]. This algorithm focused exclusively on finding linked variables before optimization, resulting in fewer fitness evaluations, but benefiting from fixed size optimal sub-groups. Other decomposition approaches such as the affinity propagation evolutionary consistency (APEC) algorithm were proposed in hopes of combating the dis-

advantages of the RDG algorithm. Unlike the RDG algorithm, APEC functioned by finding variable dependencies throughout optimization without the use of fitness evaluations by calculating the probability variables are interconnected using optimization data [13]. Recent literature has focused on self-learning components, where more optimization evaluations are spent on variable groups that contribute greater to the overall fitness. One such algorithm is the circular sliding controller algorithm, where fitness evaluations increase when the window is over optimal sub-groups [14].

1.1 Contributions

This thesis addresses several issues that contribute to the literature on large scale optimization problems. The first is to determine which decomposition approaches are most suitable for the CPSO paradigm. This thesis completes a comprehensive empirical study through assessing various decomposition methods, not currently reviewed with a CPSO optimizer. Secondly, this thesis performs an extensive scalability study for a range of problem dimensions. Finally, this thesis further evaluated the performance of two previously proposed decomposition approaches [2], with state-of-the-art modern approaches implemented in a CPSO context. Comparison with decomposition approaches for other meta-heuristics is outside the scope of this thesis.

The rest of this thesis is as follows. Chapter 2 provides the necessary background and related work. The empirical process is discussed in Chapter

3. The results and experimental analysis are discussed in Chapter 4 and the concluding remarks and future works are provided in Chapter 5.

Chapter 2

Background

This chapter outlines the necessary background information about the existing problem decomposition variants. The vanilla PSO is reviewed in Section 2.1, followed by the CPSO methodology and inspired decomposition methods.

2.1 Particle Swarm Optimization

PSO use particles' neighbourhood best and personal best positions to calculate velocity [27]. The neighbourhood best, often refers to the entire swarm, also known as global best. The next position is then calculated using the following:

- Momentum: A fraction of the previous step size in the previous direction

- Cognition: Calculated stochastically based on the difference between the current position and the personal best position
- Social: Calculated stochastically based on the difference between the current position and the neighbourhood best positions.

Working with a swarm, a PSO aims to find optimal points in the landscape, usually referring to minima or maxima. PSO algorithms can be used in a wide range of problems, including, LSGO, vehicle routing problems, ANN training, or other continuous or discrete problems. [27].

Vanilla PSO

The PSO algorithm refers to the algorithm created in 1998 by Shi and Eberhart [28]. This PSO begins by randomly initializing the particle positions within feasible search space associated with the problem. The particle positions are updated with the use of four pieces of information: the current position, the previous velocity, the particle's best position, and the global best position.

The positions are then updated using the velocity determined in

$$\vec{v_i}(t+1) = w\vec{v_i}(t) + c_1\overrightarrow{r_1}(t) \cdot (\vec{p_{best}} - \vec{x_i}(t)) + c_2\overrightarrow{r_2}(t) \cdot (\vec{g_{best}}(t) - \vec{x_i}(t)) \quad (2.1)$$

where $\vec{x_i}$ refers to the particles current position, $\vec{v_i}$ the particles velocity, $\vec{p_{best}}$ the particles personal best position and $\vec{g_{best}}$ is the swarms global best position. The w refers to the inertia value. This refers to the momentum

which moves the particles in the direction of the previous search space. The constant c_1 and c_2 are the cognitive and social acceleration coefficients respectively. The cognitive value, is used to determine how much the new position moves towards the particles best position ($g_{\vec{best}}$). The social acceleration coefficient is used to determine how much the particle moves towards the best position found by the entire swarm as from the first iteration. $\vec{r_1}$ and $\vec{r_2}$ are vectors of random numbers, where each random value is a sampled from a uniform distribution over the range [0,1]. This is then added to the current position as shown in

$$\vec{x_i}(t+1) = \vec{x_i}(t) + \vec{v_i}(t+1) \tag{2.2}$$

The Vanilla PSO algorithm is shown in algorithm 1. Where P refers to the particle, y_i refers to the local best, x_i refers to the particle position and $\hat{y}$ refers to the global best. These values are all matrix values of size equal to the total dimensionality of the problem being solved.

Algorithm 1 Vanilla PSO

```
Init: n-dimensional PSO, P
while Stopping condition not true do
    for each particle i = 1,..., P.s do
        if f(P · x_i) < f(P · y_i) then
            P · y_i = P · x_i        // Update Local Best
        if f(P · x_i) < f(P · ŷ) then
            f(P · ŷ) = P · y_i       // Update Global Best
    end
    for Each particle i = 1, ..., P.s do
        Update velocity using Equation 2.1
        Update position using Equation 2.2
    end
end
```

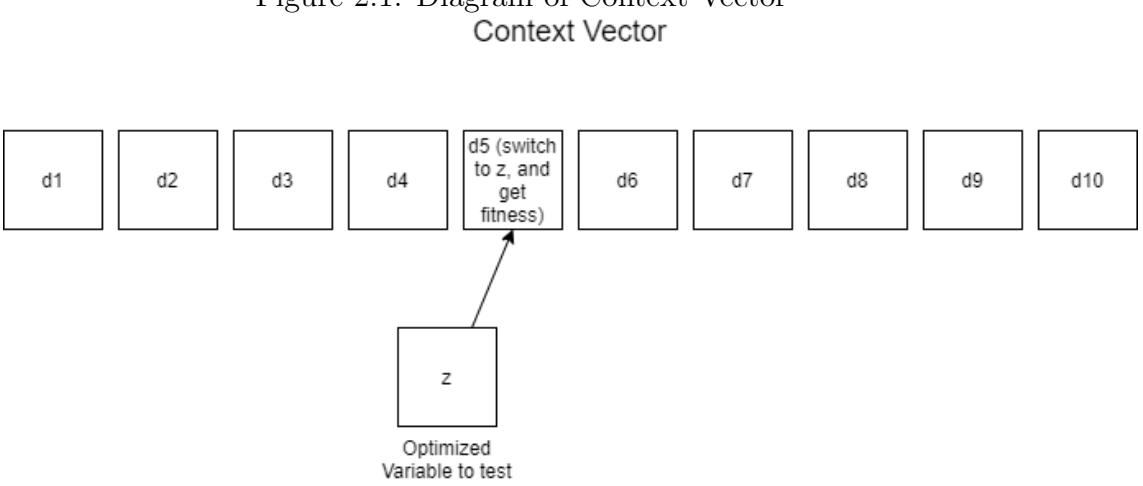
2.2 Cooperative PSO

The cooperative PSO (CPSO) algorithm decomposes the problem into smaller sub-sections, to minimize the curse of dimensionality [26]. The CPSO-S algorithm divides the problem into n single-dimension problems, optimizing each dimension with its own swarm from the original vanilla PSO algorithm. The CPSO-Sk algorithm, divides the problem into k/n subswarms.

This technique brings into consideration the method to ensure the problem is optimized as its original size, that is, an n-dimensional problem, and it must be optimized as such. This resulted in the creation of the context vector, where the individual variables are substituted throughout optimization. The context vector function is known as $b(j, z)$, where the function takes in variables j and z, variable j is then replaced with variable z and the

resulting fitness is returned. This technique allows for each variable to be tested, while continuing to optimize as a full size problem. Figure 2.1, shows a context vector, where $d1, d2...d10$ is the original size of the problem and the variable being optimized, z, is substituted for the corresponding variable in the context vector. Once the variable is substituted, the context vector now holds a complete solution and the fitness can be generated, we can then determine if this variable change improved or hindered the fitness value.

Figure 2.1: Diagram of Context Vector



With the context vector enforced, the CPSO algorithm can now optimize freely as a single dimensional vanilla PSO for each decision variable in the set. This is shown in Algorithm 2. Where n refers to the number of sub-swarms to be solved, j refers to the sub-swarm in scope the fitness value is calculated through function $b(j, z)$ discussed above. Figure 2.1 shows a visualization of the sub-swarms each optimizing one dimension of the context vector.

Algorithm 2 CPSO-S

Init: n one-dimensional PSOs, Pj, j = 1,...,n
for *each sub-swarm j = 1,...,n* **do**
  for *each particle i = 1,..., P.s* **do**
    if $b(Pj.xi, y) < f(Pj.yi)$ **then**
      $Pj.yi = Pj.xi$ // Update Local Best of j subswarm
    if $b(Pj.xi, \hat{y}) < \mathrm{f}(Pj \cdot \hat{\mathbf{y}})$ **then**
      $\mathrm{f}(Pj \cdot \hat{\mathbf{y}}) = Pj.yi$ // Update Global Best to context vector
  end
  for *Each particle* $i = 1, ..., P.s$ **do**
    Update velocity using Equation 2.1
    Update position using Equation 2.2
  end
end

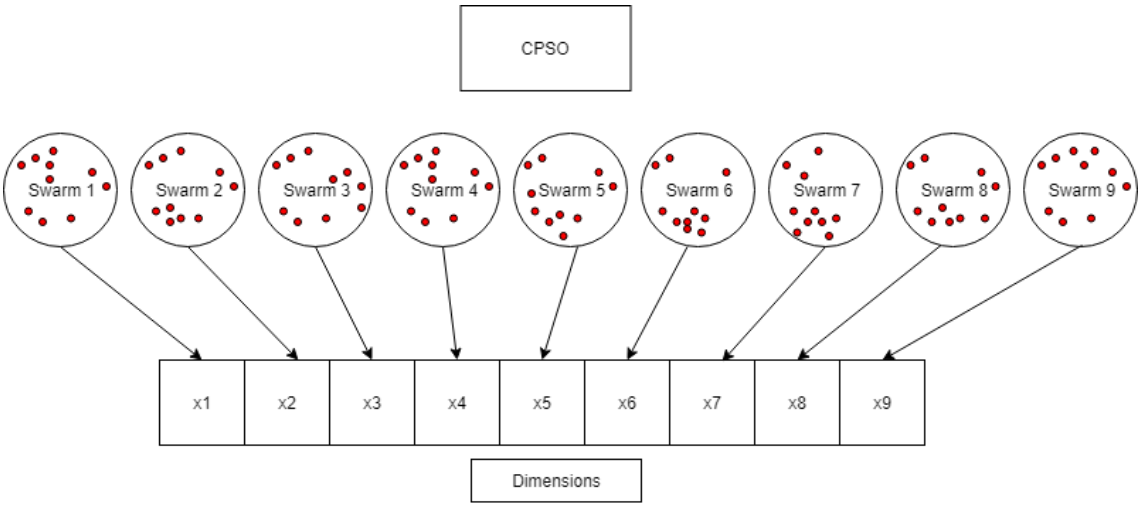


Figure 2.2: Diagram of CPSO-S algorithm and its mechanics

2.2.1 Decomposition and Merging Cooperative Particle Swarm Optimization

The decomposition cooperative particle swarm optimization (DCPSO) and the merging cooperative particle swarm optimization (MCPSO) are discussed in section 2.2.1.

DCPSO

The DCPSO algorithm utilizes the exploration and exploitation benefits of a vanilla PSO and CPSO algorithm by beginning optimization as one n-dimensional PSO swarm that splits at a fixed rate over the total number of iterations until ending optimization as a CPSO algorithm optimizing an individual decision variable per swarm [2]. The fixed frequency of decomposition is given by

$$n_f = \frac{n_T}{1 + \left(\frac{\log(n)}{\log(n_r)}\right)} \tag{2.3}$$

where n_T is the total number of iterations, n_f is the number of iterations between decomposition steps, n is the total number of dimensions, and n_r is the number of resulting sub-swarms from each split. In this thesis n_r is 2. During each phase of the decomposition, the decision variables are shuffled to ensure a variety of variable interactions being considered. Algorithm 3 shows the pseudo-code for the DCPSO algorithm.

Algorithm 3 DCPSO

Init: n-dimensional PSO, P
while *Stopping condition not true* **do**
  if *Decomposition condition is true* **then**
    decompose each P_j into n_r sub-swarms // Decomposes problem into n_r size sets
  for *each sub-swarm j = 1,...,k* **do**
    for *each particle i = 1,..., P.s* **do**
      if $b(Pj.xi, y) < f(Pj.yi)$ **then**
        $Pj.yi = Pj.xi$ // Update Local Best of j subswarm
      if $b(Pj.xi, \hat{y}) < \mathrm{f}(Pj \cdot \hat{\mathbf{y}})$ **then**
        $\mathrm{f}(Pj \cdot \hat{\mathbf{y}}) = Pj.yi$ // Update Global Best to context vector
    end
    for *Each particle* $i = 1, ..., P.s$ **do**
      Update velocity using Equation 2.1
      Update position using Equation 2.2
    end
  end
end

MCPSO

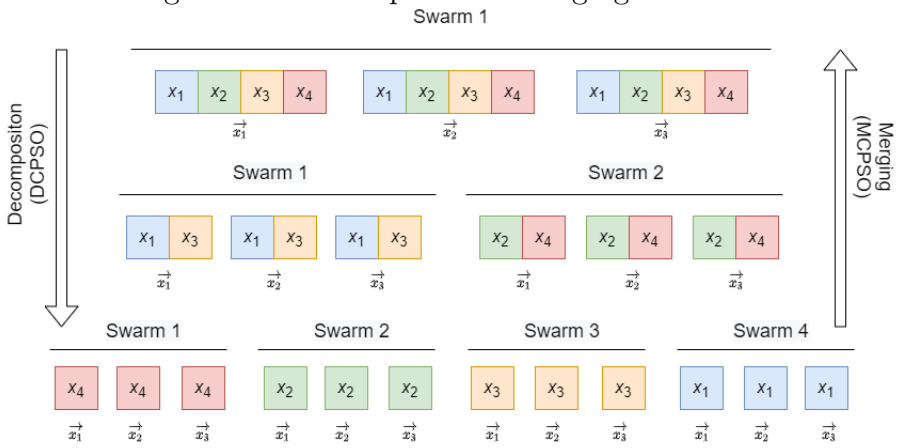
The MCPSO algorithm utilizes exploration and exploitation by benefiting from a CPSO being merged into a vanilla PSO. This algorithm works in reverse order to the DCPSO by spending its beginning iterations as a CPSO-S and finalizing its run as a vanilla PSO [2]. The inner decomposition steps (CPSO-Sk), are a size of the original sub-swarm multiplied by two, and randomly shuffled each merging step. The MCPSO algorithm utilizes Equation 2.3, in this case, n_f is the number of iteration between merging steps, n_r is the number of sub-swarms merged together from each joining, also a static

value of two. The merging will take place when n_f iterations have happened between each merging. Algorithm 4 shows the pseudo-code for the MCPSO algorithm. Figure 2.3 clearly shows the structure of the DCPSO and MCPSO.

Algorithm 4 MCPSO

Init: n one-dimensional PSOs, $\mathrm{P}_j, j = 1, ..., n$
while *Stopping condition not true* **do**
  if *Merging condition true* **then**
    merge each n_r sub-swarms // merges problem into n_r size sets
  for *each sub-swarm j = 1,...,k* **do**
    for *each particle i = 1,..., P.s* **do**
      if $b(Pj.xi, y) < f(Pj.yi)$ **then**
        $Pj.yi = Pj.xi$ // Update Local Best of j subswarm
      if $f(Pj.xi, \hat{y}) < \mathrm{f}(Pj \cdot \hat{\mathbf{y}})$ **then**
        $\mathrm{f}(Pj \cdot \hat{\mathbf{y}}) = Pj.yi$ // Update Global Best to context vector
    end
    for *Each particle* $i = 1, ..., P.s$ **do**
      Update velocity using Equation 2.1
      Update position using Equation 2.2
    end
  end
end

Figure 2.3: Decompose and merging mechanics



2.2.2 Recursive Differential Grouping

The recursive differential grouping (RDG) decomposition strategy is the only static decomposition method implemented in this paper. The RDG method utilizes directional derivatives to determine variable dependencies prior to the execution of the evolutionary algorithm (EA). Directional derivatives use the formula below to determine the amount of change the other variables notice. This is known as perturbing as shown in Equation 2.4:

$$\mathcal{D}_{\mathbf{u}} f(\mathbf{x}) = \sum_{i=1}^{n} \frac{\partial f(\mathbf{x})}{\partial x_i} u_i \tag{2.4}$$

If the fitness change by perturbing decision variable x_i varies for different values of x_j, x_i and x_j interact [15]. Perturbing is the act of slightly altering a single decision variable to examine its effects on other variables and how they react to the change.

RDG uses Equation 2.4 to perturb the decision variables X_1 of $x_{l,l}$ and $x_{u,l}$, where X_1 is a subset of decision variables and $x_{l/u,l}$ is a single decision variable where l and u refers to the lower bound and upper bound respectively. The same then happens for subset $X2$, for variable $x_{l,m}$ and $x_{u,m}$ which are the variables of a different subset to compare for interactions, where m is the middle between the lower bound and upper bound. The fitness differ-

ence between $x_{l,l}$ and $x_{u,l}$, known as (δ_1) and the fitness difference between $x_{l,m}$ and $x_{u,m}$ is (δ_2). Both values refer to the fitness difference between the lower and upper bound after perturbing the decision variables. If the fitness difference between (δ_1) and (δ_2) is greater than a threshold, ϵ, then some interaction exists between the subsets X_1 and X_2. The magnitude, ϵ is shown in Equation 2.5.

$$\epsilon = \alpha \cdot \min \{|f(\mathbf{x}_1)|, \cdots, |f(\mathbf{x}_k)|\}$$

(2.5)

Where, $x_1, ..., x_k$ are k randomly generated candidate solutions, and α is the control coefficient [15].

The RDG method determines interactions by using a recursive method to traverse all the variables, resulting in some number of fitness evaluations being utilized prior to the execution of the EA. This algorithm is completed as follows:

1. Determine the interaction between the first decision variable x_1 and all remaining decision variables using equation 2.4

2. If the interaction is not found, x_1 is placed in its own grouping and the algorithm moves to x_2 and repeats the interaction tests.

3. If any interactions are noticed, the remaining decision variables are divided into two equally-sized groups G1 and G2.

4. Interactions are then x_1 and G1, and x_1 and G2 will be identified recursively until all variables that interact are placed into subset X1.

5. X1 is then compared to all other decision variables (not including decision variables in X1) to identify any individual variables that interact with x_1.

6. These variable interactions will now be placed in a group that is non-separable.

7. The RDG method next moves to the next decision variable not already grouped and repeats the process.

8. The algorithm then outputs groups of separable and non-separable decision variables

.

This procedure is shown in algorithm 5 and the interaction method is shown in algorithm 6.

Algorithm 5 RDG

Init: seps and nonseps as empty groups
Set all decision variables to the lower bounds: $x_{l,l} = \text{lb}$
Calculate fitness: $y_{l,l} = f(X_{l,l})$
Assign the first variable x_1 to the variable subset X_1
Assign the rest of variables to the variable subset X_2
while *X2 is not empty* **do**
  $[X_1^*] = \text{INTERACT}(\text{X}_1, X_2, x_{l,l}, y_{l,l}, \epsilon)$
  if $[X_1^*]$ *is the same with* X_1 **then**
    if X_1 *contains one decision variable* **then**
      Add X_1 to seps
    else
      Add X_1 to nonseps
    Empty X_1 and $[X_1^*]$
    Assign the first variable of X_2 to X_1 Delete the first variable in X_2
    else
      X1 = $[X_1^*]$
      Delete the variables of X_1 from X_2
end

Algorithm 6 *INTERACT function*

$\mathbf{x}_{u,l} = \mathbf{x}_{l,l}; \mathbf{x}_{u,l}(X_1) = \mathbf{ub}(X_1)$
Calculate the fitness change: $\delta_1 = y_{l,l} - f(\mathbf{x}_{u,l})$
$\mathbf{x}_{l,m} = \mathbf{x}_{l,l}; \mathbf{x}_{l,m}(X_2) = (\text{lb}(X_2) + \mathbf{ub}(X_2))/2$
$\mathbf{x}_{u,m} = \mathbf{x}_{u,l}; \mathbf{x}_{u,m}(X_2) = (\text{lb}(X_2) + \mathbf{ub}(X_2))/2$
Calculate the fitness change: $\delta_2 = f(\mathbf{x}_{l,m}) - f(\mathbf{x}_{u,m})$

if $|\delta_1 - \delta_2| > \epsilon$ **then**
  if X_2 *contains one variable* **then**
    $X_1 = X_1 \cup X_2$
  else
    Divide X_2 *into equally-sized groups* G_1, G_2
    $[X_1^1] = \text{INTERACT}(X_1, G_1, \mathbf{x}_{l,l}, y_{l,l}, \epsilon)$
    $[X_1^2] = \text{INTERACT}(X_1, G_2, \mathbf{x}_{l,l}, y_{l,l}, \epsilon)$
    $[X_1] = X_1^1 \cup X_1^2$
return X_1

2.2.3 Circular Sliding Controller

Circular sliding controller (CSC) algorithm utilizes a variety of decomposition techniques to address LSGO problems [14]. The CSC algorithm uses a window which slides circularly around the context vector, changing the sub-swarm size, also known as the window size, each full completion of the context vector. The context vector in this case is used as the circular structure as shown in figure 2.4. The sliding step is how large the step size is for each optimization step. It is decided dynamically by determining whether the window centers on an active, inactive or regular region. The activity of each sub-region is calculated as follows

$$r_s^i = \frac{n_s^i}{n_s^i + n_f^i} \tag{2.6}$$

where n_s^i equals the number of successes in optimizing that sub-regions and n_f^i is the number of fails, that is the optimization in that iteration was unsuccessful at finding a new optimum. These values are then used to calculate the activity of these regions, where the top 30 percent most active areas are marked as active, the bottom 30 percent as inactive and the rest are regular. These active regions then indicate that more time should be spent in these areas as they tend to find more optimums than their inactive counterparts.

The sliding step size is then calculated by max{1,m/5} for active regions, max{1,2m/5} for regular regions and max{1,3m/5} for inactive regions, where m is the window size. The window size is statically selected

prior to the optimization as $W = \{w_1, w_2..., w_t\}$, these values are selected as parameters for the run. The values, however, are selected based on roulette strategy, allowing the most optimal window sizes to be selected a higher amount of iterations. Roulette strategy selects the more ideal window size a greater number of times, while less ideal will be selected less [14]. The window sizes, use a window fitness:

$$r_w^i = \frac{|v - v'|}{|v|} \tag{2.7}$$

where r_w^i is the window fitness, v is the best fitness of the previous cycle, v' is the best fitness value of the current cycle. The Roulette strategy:

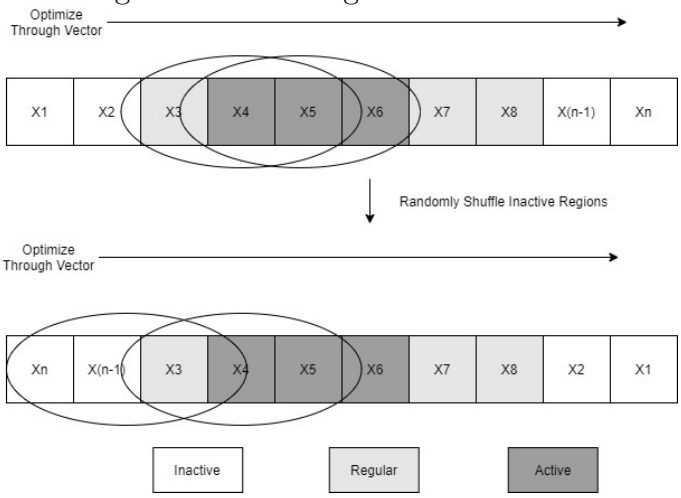
$$p_w^i = \frac{10^{r_w^i}}{\sum_{j=1}^{t} 10^{r_w^i}} \tag{2.8}$$

where the fitness of the current window, divided by the sum of all fitness values determines the probability of window selection, thus favoring more ideal window sizes.

The CSC framework can be summarized as follows:

1. Begin a new cycle, where i is the first location on the circular structure
2. Randomly select a window size from the window pool.
3. Optimize the variables in the current window and record the number of success n_s and failing n_f.
4. Identify the activity of the sub-regions.

Figure 2.4: CSC algorithm mechanics



5. Update context vector.

6. If window is not back at the beginning, $i = i + s$, and go back to step 3.

7. Record the best optimization results of the current cycle, Terminate algorithms stop condition if satisfied.

8. Re-calculate selection probability for the selected size of the window.

9. Randomly permute the order of decision variables in inactive regions and go to step 1 for a new cycle.

2.2.4 Affinity Propagation Evolution Consistency-Based Decomposition

Affinity propagation evolution consistency (APEC) based decomposition is an adaptive decomposition approach that group's variables without the use of additional fitness evaluations by examining the interaction probability of each variable paring. The APEC algorithm uses the fitness evaluations within the cooperative co-evolutionary algorithm to determine relationships dynamically through the process of the optimization.

APEC algorithm begins with the evolution consistency between variables, that is, the amount the variables interact throughout evolution as shown in Equation 2.9.

$$I_i^{k,m} = \begin{cases} 1 & x_i^k > x_i^m \\ 0 & x_i^k = x_i^m \\ -1 & x_i^k < x_i^m \end{cases} \quad k,m \in [1,T] \quad I_j^{k,m} = \begin{cases} 1 & x_j^k > x_j^m \\ 0 & x_i^k = x_j^m \\ -1 & x_j^k < x_j^m \end{cases} \quad k,m \in [1,T] \tag{2.9}$$

where $I_i^{k,m}$ is the evolution direction of the ith variable between the kth and mth generation, and $I_j^{k,m}$ represents the jth variable [13]. $I_i^{k,m} = 1$ indicates that the value of the variable becomes smaller from the kth generation to the mth generation, $I_i^{k,m}$ = -1 indicates the value of the variable becomes larger from the kth to mth generation, and $I_i^{k,m} = 0$ signifies the value of this variable remains unchanged from the kth to mth generations. Given this information, we can calculate the evolution consistency between these

two variables by adding them into matrix $EC_{i,j}$, where this represents the relationship between two variables. This process can be shown in Equation 2.10:

$$EC_{i,j} = \frac{1}{T(T-1)} \sum_{k\in[1,T],m\in[1,T]k\neq m} I_i^{k,m} I_j^{k,m} \tag{2.10}$$

where T represents the number of iterations between evolutionary steps.

This information is then processed through an Affinity Propagation Clustering Algorithm (AP) [29] which processes the groups and clusters them based on the relationships they contain. These groups are then optimized by the cooperative evolutionary algorithm. AP is discussed in section 2.2.4. This is shown in algorithm 7. Where X refers to the historical evolution data of size $T \cdot D$, with T and D being the steps and total dimensions respectively.

Algorithm 7 APEC

Init: The collected historical evolutionary data X with $T \cdot D$
for *i = 1 : D - 1* **do**
  Compute the indicator I_i according to Equation 2.9
  for *j = i + 1 : D* **do**
    Compute the indicator I_j according to Equation **??**
    Compute the evolution consistency $EC_{i,j}$ according to Equation 2.10
    $EC_{j,i} = EC_{i,j}$
  end
end
Obtain the median vector p of $|\boldsymbol{EC}|$ in row
Cluster variables using the AP algorithm with $|\boldsymbol{EC}|$ as the similarity matrix and p as the preference vector
Obtain the variable groups G according to the clustering results

Affinity Propagation (AP) Clustering Algorithm

As previously mentioned, $EC_{i,j}$ represents the relationship between each variable i and j. This matrix is then able to be clustered into groups based on relationships with other variables using the AP clustering algorithm [29]. This clustering algorithm can use any clustering algorithm of choice. AP was chosen here as it has shown to have good results as well as adaptively determines the number of clusters [29].

The AP clustering algorithm begins by taking the similarities S between data points as input. Each entry $s(i,j)$ is a subset of S and it indicates the similarity amount between the jth and ith input. There are two kinds of information that are exchanged between data points: responsibility and availability, where $r(i,k)$ is the responsibility and $a(i,k)$ is the availability matrix. The responsibility matrix is the similarity of point i and point k. The availability matrix is the points suitability of becoming a clustering center, known as exemplar.

Initially, each entry $a(i,k)$ in A is set to zero. Then, each entry $r(i,k)$ is updated as shown in Equation 2.11

$$r(i,k) = s(i,k) - \max_{k's.t.k' \neq k} \{a\,(i,k') + s\,(i,k')\} \tag{2.11}$$

where $a(i,k)$ is then set to Equation 2.12.

$$a(i,k) = \min \left\{ 0, r(k,k) + \sum_{i's.t.i' \notin \{i,k\}} \max \{0, r(i',k)\} \right\} \tag{2.12}$$

The diagonal values $a(k,k)$ are updated differently, and are shown in Equation 2.13:

$$a(k,k) = \sum_{i's,t.i' \neq k} \max \{0, r(i',k)\} \tag{2.13}$$

After the calculations above are completed, those with $r(i,i) + a(i,i) > 0$ are chosen as the exemplars, which are the cluster groups.

2.2.5 Random Adaptive Grouping

The Random Adaptive Grouping (RAG) decomposition method is a random and adaptive grouping strategy with fixed sub-swarm size throughout the optimization process [16]. The RAG algorithm works as follows. The n-dimensional problem vector is divided prior to execution into m s-dimensional sub-components, where m is the amount of sub-swarms and s is the size of each sub-swarm. These original groups are randomly assigned as current information on variable dependencies is unknown. The CPSO algorithm is then performed over a number of fitness evaluations, known as T. During the optimization of T fitness evaluations, the difference in fitness evaluation in each sub-component Δf_i. This value will be used to determine the bottom

$m/2$ sub-components with the worst performance (smallest Δf_i) over the length of T fitness evaluations. The smallest Δf_i sub-components will then be randomly reassigned variables from the other similarly poor-performing sub-components. The Δf_i value is then reset back to 0, and the optimization process is redone for T iterations with the newly arranged decision variables. This process is completed until the total fitness evaluations are used up, continuously arranging the poorest performing sub-components. This process is shown in pseudo-code in algorithm 8.

Algorithm 8 RAG

Init: FEV$_Global, T, FEV_Local = 0$
An n-dimensional problem vector is randomly divided into m s-dimensional sub-components
Randomly mix indices of variables
$i = 1$
Evolve i sub-component with CPSO algorithm, record CBS and PBS
$\Delta f_i = |PBS - CBS|$
if $i < m$ **then**
  Continue to optimize variable groupings with CPSO
if $FEV_{Local} < T$ **then**
  Re-initialize $i = 1$ and optimize this sub-component with CPSO
Choose $m/2$ sub-components with the worst performance ($m/2$ smallest Δf_i) and randomly mix sub components
  Init: T, FEV$_Local$,$\Delta f_i = 0$
if $FEV < MaxEvals$ **then**
  Optimize new groupings with CPSO and repeat algorithm
else
  Return best solution

2.3 Previous Work

Previous work in the field of decomposition is increasing as problem search spaces become increasingly large and evolutionary algorithms struggle to find optimal solutions without decomposing the problem. Most previous work is completed on algorithms such as genetic algorithms, Differential Evolution (DE), and Self-Adaptive DE [30] due to there noticeable advantages and ease of implementation.

Cooperative co-evolution originated with use in the GA meta-heuristic [9], where the researches discovered the benefits of the divide-and-conquer approach when solving LSGO problems. This research was quickly expanded into various other meta-heuristics, including ACO [4], PSO [26], FF [25], DE [12] and ABC [24], all of which showed promising results across various benchmark problems in contrast to their non-decomposition based counterparts.

The CPSO approach was one of the earlier techniques, proposed shortly after the cooperative GA. This paper expanded on a few variants of cooperative co-evolution. The CPSO-SK and CPSO-HK technique, the CPSO-SK technique looked into changing the sub-grouping sizes using the variable k, with this the CPSO would hopefully mitigate the effects variable dependencies cause in fully separating the variables in a traditional CPSO approach. The CPSO-HK attempted to build on this further, by decomposition the problem into k subsets, however with the addition of alternating between

this CPSO-Sk and a PSO algorithm, this built a foundation for the need for connecting inter-connected variables and was shown in the literature to have benefits across many problems.

Shortly after the cooperative co-evolutionary foundation was built, literature quickly proposed a variety of techniques aimed to mitigate the effects of variable dependencies [14, 13, 15, 16, 21, 2]. Majority of these techniques are proposed on the DE meta-heuristic as the technique to evaluate the performance of the decomposition approach.

The range of approaches to solving the issue of variable dependencies has been broken into a variety of techniques since the inception of the original cooperative co-evolutionary technique. Many of the techniques are adaptive in nature, meaning they change the variable grouping sizes and variables dynamically as the base optimizer is running [2, 13, 14], while others perform static grouping where the grouping sizes stay unchanged through optimization [15, 21, 16]. Of these techniques, benefits and drawbacks to both can be noted. Primarily, with the dynamic approach, variables and grouping sizes are constantly changing, thus being more likely to find optimal sub-groups. This however, is unnecessary if variable groupings are optimally chosen prior to the run. Choosing variable groupings prior to optimization, as used in the RDG approach, causes a number of fitness evaluations to be taken prior to optimization, hence resulting in less fitness evaluations for the meta-heuristic to find ideal solutions.

Research on decomposition approaches implemented on the CPSO base

optimizer is much more scarce, with few researchers choosing to display results with this meta-heuristic algorithm. Research on cooperative PSO decomposition is shown in detail in Douglas et. al. [20]. Where the researchers compared various cooperative PSO decomposition techniques against the proposed decomposition approaches from the paper, known as DCPSO and MCPSO. Their paper showed promising results for CPSO decomposition and revealed that decomposing the problem size dynamically increased the performance against traditional CPSO methodology and techniques as well as gave valuable insight into decomposition based on the PSO meta-heuristic.

Chapter 3

Experimental Setup

Each test was run 30 times, and an average of all global best fitness values was taken. Section 3.1 examines the PSO parameters used across all tests, section 3.2 discusses the parameters used over the different decomposition approaches, section 3.3 determines the benchmark functions compared against and Section 3.4 refers to the statistical analysis completed in this thesis.

3.1 PSO Parameters

The PSO parameters were chosen based on values that tend to converge well [31], these values were later compared by Harrison *et al*, and shown to have reasonably good results [32]. These values are static throughout all tests for equal comparison. These values are shown below in Table 3.1.

The velocity begins at zero, the starting random values are initialized

Parameter	Value
Particles Per Swarm	20
w	0.7298
c1	1.49618
c2	1.49618
Fitness Evaluations	3000*d

Table 3.1: PSO parameters

within the bounds of each benchmark problem. Velocity clamping is set to $\delta = 0.1$.

3.2 Decomposition Parameters

Subsection 3.2.1 determines the DCPSO parameters, 3.2.2 refers to the MCPSO parameters, 3.2.3 looks at the RDG parameters, 3.2.4 discusses the CSC parameters, 3.2.5 determines the APEC parameters and 3.2.6 determines the RAG parameters. All algorithms use the same set of PSO parameters from table 3.1.

3.2.1 Decomposition Cooperative PSO

The number of resulting sub-swarms from each split, n_r is two. Swarms are decomposed every n_f iterations as shown in Equation 2.3.

3.2.2 Merging Cooperative PSO

The number of resulting sub-swarms from each split, n_r is two. Swarms are decomposed every n_f iterations as shown in Equation 2.3.

3.2.3 Recursive Differential Grouping

The RDG algorithm uses the control coefficient $\alpha = 10^{-12}$ and $k = \log_2(n)$ as the static parameters throughout all benchmark functions.

3.2.4 Circular Sliding Controller

The parameter set used within this algorithm is chosen as $W = \{1, 5, 10, 20, 50, 100\}$ for problem dimension of 1000. These values are scaled down to the following for $W = \{\lceil 0.001 * n \rceil, \lceil 0.005 * n \rceil, \lceil 0.01 * n \rceil, \lceil 0.001 * n \rceil, (0.02 * n), \lceil 0.05 * n \rceil\}$. The values were scaled up in equal proportion for the test of 2000 dimensions.

3.2.5 Evolution Consistency Based Decomposition

The APEC algorithm utilized a variable $T = 50$ as shown in the literature [13]. This variable has been shown to not vary results significantly from tested values between $T = 10$ to $T = 60$.

3.2.6 Random Adaptive Grouping

The RAG algorithm is completed with a T value set to 0.1 of the total fitness evaluations, resulting in 10 variable reassignments. The sub-swarm size was

statically set to 10 throughout all tests completed. These parameters were chosen as they are shown to achieve optimal results over most situations as shown in Sopov et al. [21].

3.3 Benchmark Functions

The benchmark functions tested are from the CEC'2010 Special Session and Competition on Large-Scale Global Optimization [33]. These functions utilize a range of separable, partial separable, and fully non-separable functions. This benchmark also has a variety of multi-modal and uni-modal functions allowing for a range of functions to test. The CEC'2010 benchmark suite is composed of the following benchmark functions:

1. Three separable functions
2. Five single-group m non-separable functions
3. Five $\frac{D}{2m}$-group m non-separable functions
4. Five $\frac{D}{m}$-group m non-separable functions
5. Two non-separable functions

Separable functions refer to problems where no variables are inter-connected. Single-group m non-separable functions are those which have a single group of size, m, with inter-connected variables and the rest are separable. $\frac{D}{2m}$-group m non-separable functions are those which have the total dimensionality of

the problem, divided by $2m$ groups. For example, in a problem with 1000 dimensions (D), and m of 50, these group sizes would be 1000/100, that is, 10 groups of size m, non-separable variables, the rest (50 variables) are separable. $\frac{D}{m}$-group m non-separable functions are total dimensions divided by the value m group, so in the example above this would be 20 groups, each of size 5. Finally, Non-separable functions are those in which all variables are inter-connected.

3.4 Statistical Analysis

The algorithms were run for 30 independent runs for each of the benchmark problems outlined in Section 3.3. A Friedman test was then used to determine if distributions between any pair of the algorithms were considered significantly different with a confidence interval of 95 percent. After completion of this test, if a significant difference was found the algorithms were tested with a Mann Whitney test utilizing the Holm-Bonferroni statistical test [34] to mitigate the family-wise error rate (FWER). The highest number of winning algorithms for each tested criterion was recorded, that is, overall best fitness and consistency of the fitness values was ranked based on these criteria.

3.4.1 Friedman Test

The Friedman test is used to determine if a significant difference exists between any of the two algorithms [35]. The Friedman test functions by ranking row-wise per problem, for example, each problem instance $i \in [1, \ldots, n]$, each algorithm $j \in [1, .., k]$ is ranked from 1 to k. Ties are then broken by using average ranks to derive the p-value. The Friedman test has no way of showing where significance exists between two of the algorithms and is utilized to determine if any two pairs show significant difference which will then be tested with a suitable post-hoc test to obtain the p-value for each pairwise comparison, the Mann-Whitney U test was utilized in this thesis as the post-hoc comparison.

3.4.2 Holm-Bonferroni Family Wise Error Rates (FWER)

The Holm-Bonferroni FWER ranking is a technique used to deal with testing multiple algorithms against each other [34]. Without this technique, the Type 1 error change becomes increasingly high as more algorithms are tested. This error can be shown as

$$1 - (1 - \alpha)^{(k-1)} \tag{3.1}$$

where k is the number of algorithms. With $k = 6$ and $\alpha = 0.05$, which is the case in this thesis, the probability of a false discovery would increase to approximately 0.23 which is unacceptably high. The Holm-Bonferroni test reduces this change of a type 1 error. This algorithm uses the following

formula:

$$\alpha/(n - \text{rank} + 1) \tag{3.2}$$

The p-values are then ranked in order from lowest to highest and updated based on Equation 3.2 to decrease the chances of receiving a type 1 error.

Chapter 4

Results

This chapter discusses the results for different dimensions, which provides the scalability study.

4.1 30 Dimensions

4.1.1 Accuracy

The algorithms performance is ranked in Table 4.1 and Table 4.2 for accuracy and consistency respectively. These results are based on the 30-dimensional problem set which is the smallest problem size tested. For separable problems, the CSC algorithm is the winner with the MCPSO algorithm being close behind. In the partially separable problem sets the RDG algorithms performance increases significantly over the higher dimensional problems with this algorithm being the winner in $D/2m$ and D/m non-separable problems. For

Table 4.1: 30 dimension statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep** W/T/L Rank	**S-Non** W/T/L Rank	**D/2m-Non** W/T/L Rank	**D/m - Non** W/T/L Rank	**Non** W/T/L Rank
DCPSO	8, 1, 6 3	16, 5, 4 2	12, 5, 8 2	10, 2, 13 4	7, 0, 3 2
MCPSO	12, 0, 3 2	19, 5, 1 1	11, 9, 5 3	12, 4, 9 3	6, 2, 2 3
RDG	6, 1, 8 4	7, 3, 15 5	13, 7, 5 1	16, 4, 5 1	1, 2, 7 5
APEC	1, 0, 14 6	10, 4, 11 3	1, 2, 22 5	4, 4, 17 6	2, 2, 6 4
CSC	13, 0, 2 1	9, 2, 14 4	12, 8, 5 2	15, 0, 10 2	9, 0, 1 1
RAG	4, 0, 11 5	3, 3, 19 6	8, 5, 12 4	9, 4, 12 5	2, 0, 8 4

a single group non-separable, the MCPSO continues to be the winner in the 30 dimension problem group. In non-separable problems, the CSC continues to be the winner in these tests.

4.1.2 Consistency

Table 4.2 shows the consistency for the 30 dimension problems. These results show the CSC algorithm winning in separable, $D/2m$ non-separable, and non-separable problems. The MCPSO algorithm wins in a single group non-separable and the RDG algorithm wins in D/m non-separable.

Table 4.2: 30 dimension consistency statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep**	**S-Non**	**D/2m-Non**	**D/m - Non**	**Non**
	W/T/L	W/T/L	W/T/L	W/T/L	W/T/L
	Rank	Rank	Rank	Rank	Rank
DCPSO	8, 1, 6	16, 5, 4	12, 5, 8	8, 2, 15	7, 0, 3
	3	2	3	5	2
MCPSO	11, 0, 4	19, 5, 1	8, 9, 8	12, 4, 9	6, 2, 2
	2	1	5	3	3
RDG	6, 1, 8	8, 3, 14	13, 7, 5	17, 4, 4	1, 2, 7
	4	5	2	1	5
APEC	1, 0, 14	10, 4, 11	0, 2, 23	3, 4, 18	2, 2, 6
	6	3	6	6	4
CSC	13, 0, 2	9, 2, 14	15, 8, 2	16, 0, 9	9, 0, 1
	1	4	1	2	1
RAG	5, 0, 10	2, 3, 20	9, 5, 11	10, 4, 11	2, 0, 8
	5	6	4	4	4

Table 4.3: Average of 30 runs for dimension of 30

Function	DCPSO		MCPSO		RDG		APEC		CSC		RAG	
	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev
f1	**6.71E-02**	*1.29E-01*	**5.80E-10**	*3.11E-09*	**8.16E-10**	*1.21E-09*	**1.02E+04**	*1.03E+04*	**4.28E-02**	*3.29E-02*	**3.56E+02**	5.52E+02
f2	**6.97E-01**	*5.83E-01*	**2.65E-01**	*5.71E-01*	**1.33E+00**	*1.39E+00*	**1.57E+01**	*8.47E+00*	**1.96E-07**	*9.10E-08*	**2.76E+00**	1.56E+00
f3	**6.33E-04**	*3.12E-04*	**2.13E-02**	*1.14E-01*	**1.07E+01**	*7.23E+00*	**9.63E-01**	*4.33E-01*	**1.35E-04**	*3.16E-05*	**1.06E-01**	2.14E-01
f4	**2.07E+09**	*2.62E+09*	**1.80E+09**	*2.87E+09*	**5.56E+09**	*4.88E+09*	**8.98E+08**	*1.05E+09*	**1.29E+10**	*9.30E+09*	**5.95E+09**	8.18E+09
f5	**9.95E+04**	*2.98E+05*	**3.32E+04**	*1.79E+05*	**2.99E+05**	*5.23E+05*	**1.99E+05**	*7.45E+05*	**2.19E+06**	*1.61E+06*	**1.82E+06**	1.72E+06
f6	**1.87E-02**	*6.28E-02*	**7.39E-01**	*1.19E+00*	**1.48E+01**	*5.79E+00*	**3.73E+01**	*1.99E+01*	**5.39E+01**	*4.05E+01*	**1.87E+02**	3.64E+02
f7	**8.75E-03**	*4.59E-02*	**5.83E-16**	*2.54E-15*	**1.26E+05**	*4.70E+05*	**2.50E+01**	*2.05E+01*	**2.66E-01**	*5.18E-01*	**1.61E+02**	6.42E+02
f8	**1.14E+06**	*3.63E+06*	**8.76E+05**	*2.96E+06*	**2.95E+07**	*3.20E+07*	**6.45E+06**	*1.13E+07*	**1.85E+03**	*3.09E+03*	**5.50E+07**	4.64E+07
f9	**3.77E+04**	*9.41E+04*	**9.28E+03**	*4.98E+04*	**1.02E+05**	*1.34E+05*	**1.08E+06**	*6.10E+05*	**1.95E-01**	*1.29E-01*	**4.97E+04**	1.03E+05
f10	**3.18E+01**	*6.89E+00*	**2.16E+01**	*7.42E+00*	**1.07E+01**	*4.11E+00*	**4.32E+01**	*1.58E+01*	**1.96E+01**	*4.72E+00*	**2.64E+01**	1.01E+01
f11	**4.84E+00**	*6.98E+00*	**6.92E+00**	*5.88E+00*	**8.10E+00**	*8.48E+00*	**1.98E+01**	*1.16E+01*	**4.47E+00**	*2.40E+00*	**9.96E+00**	1.00E+01
f12	**8.60E-05**	*9.88E-05*	**1.66E-01**	*8.97E-01*	**2.56E+00**	*9.33E+00*	**3.11E+01**	*2.97E+01*	**5.59E-06**	*2.95E-06*	**3.42E-01**	1.24E+00
f13	**1.36E+02**	*7.62E+01*	**1.61E+02**	*1.04E+02*	**4.70E+01**	*4.74E+01*	**2.14E+02**	*1.04E+02*	**2.49E+02**	*5.65E+01*	**2.70E+02**	9.38E+01
f14	**7.46E+04**	*2.46E+04*	**8.62E+04**	*2.63E+04*	**4.93E+04**	*1.75E+04*	**5.83E+04**	*1.77E+04*	**1.03E+05**	*2.75E+04*	**6.38E+04**	2.60E+04
f15	**2.18E+01**	*5.45E+00*	**1.38E+01**	*3.77E+00*	**1.45E+01**	*4.18E+00*	**4.09E+01**	*1.13E+01*	**6.90E+00**	*2.16E+00*	**1.88E+01**	5.32E+00
f16	**9.33E+00**	*1.12E+01*	**6.60E+00**	*1.07E+01*	**3.69E+00**	*8.97E+00*	**4.23E+01**	*2.29E+01*	**2.00E-03**	*4.87E-04*	**3.02E+01**	1.84E+01
f17	**1.28E-01**	*1.20E-01*	**1.60E-02**	*5.99E-02*	**2.08E-01**	*8.16E-02*	**3.36E-01**	*1.42E-01*	**1.52E-05**	*5.24E-06*	**1.18E-01**	1.18E-01
f18	**2.71E+02**	*1.04E+02*	**3.49E+02**	*1.16E+02*	**5.60E+01**	*3.88E+01*	**4.20E+02**	*1.55E+02*	**4.93E+02**	*1.27E+02*	**4.06E+02**	1.09E+02
f19	**4.57E+01**	*9.88E+01*	**3.39E+02**	*4.53E+02*	**1.68E+03**	*4.01E+03*	**2.05E+02**	*1.77E+02*	**1.02E-02**	*5.34E-03*	**1.23E+03**	8.00E+02
f20	**7.63E+01**	*3.27E+01*	**2.34E+01**	*1.68E+01*	**1.21E+08**	*2.37E+08*	**1.31E+03**	*1.70E+03*	**5.13E+01**	*3.22E+01*	**1.87E+02**	5.63E+02

4.2 100 Dimensions

4.2.1 Accuracy

The algorithms performance is ranked in Table 4.4 and Table 4.5 for accuracy and consistency respectively. The first table, that is, the accuracy table, reveals that for non-separable problems the CSC algorithm is the winner with the MCPSO algorithm following behind closely. For the partially non-separable problems, the DCPSO and MCPSO win all these tests, with the DCPSO being the winner for a single group non-separable and the MCPSO winning in the other partially separable groups. The RDG algorithm is tied with the RDG algorithm for $D/2m$ non-separable for the first place. In the non-separable benchmark problems, the CSC algorithm is the winner which has the MCPSO algorithm following very close behind.

4.2.2 Consistency

Table 4.5 shows the consistency of the results, in these tests the CSC algorithm is the winner for the separable set of benchmark problems. The DCPSO is the most consistent in a single group non-separable. The MCPSO and RDG tied for $D/2m$ non-separable problems and the MCPSO wins in D/m non-separable. In the non-separable problem set, the CSC algorithm wins with the MCPSO algorithm being close behind.

Table 4.4: 100 dimension consistency statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep**	**S-Non**	**D/2m-Non**	**D/m - Non**	**Non**
	W/T/L	W/T/L	W/T/L	W/T/L	W/T/L
	Rank	Rank	Rank	Rank	Rank
DCPSO	9, 1, 5	18, 6, 1	10, 8, 7	9, 4, 12	6, 1, 3
	3	1	3	4	3
MCPSO	11, 1, 3	9, 10, 6	17, 4, 4	19, 1, 5	8, 1, 1
	2	2	1	1	2
RDG	5, 1, 9	7, 11, 7	17, 3, 5	17, 1, 7	0, 1, 9
	4	3	1	2	6
APEC	2, 0, 13	7, 11, 7	0, 3, 22	0, 3, 22	1, 1, 8
	6	3	5	6	5
CSC	13, 1, 1	4, 10, 11	11, 6, 8	16, 1, 8	9, 0, 1
	1	4	2	3	1
RAG	3, 0, 12	2, 8, 15	5, 6, 14	7, 4, 14	4, 0, 6
	5	2	4	5	4

Table 4.5: 100 consistency dimension statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep**	**S-Non**	**D/2m-Non**	**D/m - Non**	**Non**
	W/T/L	W/T/L	W/T/L	W/T/L	W/T/L
	Rank	Rank	Rank	Rank	Rank
DCPSO	9, 1, 5	18, 6, 1	10, 8, 7	13, 4, 8	6, 1, 3
	3	1	2	3	3
MCPSO	11, 1, 3	7, 10, 8	16, 4, 5	18, 1, 6	8, 1, 1
	2	3	1	1	2
RDG	5, 1, 9	7, 11, 7	16, 3, 6	17, 1, 7	0, 1, 9
	4	3	1	2	6
APEC	1, 0, 14	8, 11, 6	2, 3, 20	1, 3, 21	1, 1, 8
	6	2	4	6	5
CSC	13, 1, 1	4, 10, 11	8, 6, 11	11, 1, 13	9, 0, 1
	1	4	3	4	1
RAG	4, 0, 11	3, 8, 14	8, 6, 11	8, 4, 13	4, 0, 6
	5	5	3	5	4

Table 4.6: Average of 30 runs for dimension of 100

Function	DCPSO		MCPSO		RDG		APEC		CSC		RAG	
	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev
f1	**1.31E+01**	*1.75E+01*	**1.71E+01**	*9.20E+01*	**1.71E+01**	*9.20E+01*	**2.67E+06**	*3.62E+06*	**9.69E-02**	*4.16E-02*	**4.80E+04**	9.44E+04
f2	**2.00E+00**	*1.26E+00*	**8.62E-01**	*9.86E-01*	**3.08E+00**	*1.95E+00*	**1.05E+02**	*3.18E+01*	**7.10E-07**	*2.98E-07*	**1.83E+02**	2.23E+01
f3	**5.89E-03**	*1.77E-03*	**3.61E-08**	*3.31E-08*	**1.90E+01**	*1.50E+00*	**2.37E+00**	*1.04E+00*	**1.49E-04**	*2.11E-05*	**1.09E+00**	2.50E-01
f4	**5.53E+09**	*6.19E+09*	**3.88E+10**	*1.30E+11*	**5.17E+10**	*6.06E+10*	**1.87E+10**	*1.58E+10*	**2.34E+10**	*2.69E+10*	**8.71E+10**	7.16E+10
f5	**1.06E+07**	*3.87E+06*	**1.00E+07**	*5.68E+06*	**1.30E+07**	*6.24E+06*	**1.05E+07**	*4.45E+06*	**1.60E+07**	*7.84E+06*	**1.39E+07**	9.00E+06
f6	**5.49E+04**	*2.96E+05*	**1.32E+05**	*5.02E+05*	**3.77E+05**	*1.41E+06*	**1.15E+03**	*5.45E+02*	**9.51E+06**	*9.12E+06*	**4.93E+06**	7.56E+06
f7	**3.95E-01**	*6.47E-01*	**2.52E+01**	*4.01E+01*	**1.60E+01**	*3.28E+01*	**6.76E+03**	*8.02E+03*	**3.65E+02**	*6.57E+02*	**1.98E+03**	9.95E+03
f8	**1.43E+06**	*4.72E+06*	**3.45E+07**	*3.78E+07*	**5.40E+07**	*1.23E+08*	**3.32E+07**	*2.70E+07*	**4.27E+07**	*3.81E+07*	**5.91E+07**	4.15E+07
f9	**1.39E+07**	*1.06E+07*	**6.00E+06**	*3.14E+06*	**1.40E+06**	*1.71E+06*	**4.38E+07**	*1.72E+07*	**3.87E+07**	*1.39E+07*	**1.20E+07**	9.87E+06
f10	**3.29E+02**	*4.17E+01*	**1.83E+02**	*3.67E+01*	**1.51E+02**	*2.71E+01*	**3.86E+02**	*6.74E+01*	**2.32E+02**	*4.78E+01*	**3.73E+02**	4.86E+01
f11	**7.85E+01**	*2.61E+01*	**6.85E+01**	*2.37E+01*	**4.11E+01**	*1.17E+01*	**1.02E+02**	*2.72E+01*	**6.61E+01**	*2.91E+01*	**9.96E+01**	2.28E+01
f12	**1.14E+01**	*1.56E+01*	**3.47E+01**	*3.88E+01*	**1.03E+03**	*1.37E+03*	**1.50E+03**	*1.00E+03*	**3.31E-01**	*1.28E-01*	**1.94E+02**	1.76E+02
f13	**5.42E+03**	*5.85E+03*	**4.36E+02**	*9.72E+01*	**9.58E+03**	*2.48E+04*	**4.20E+04**	*3.64E+04*	**3.85E+03**	*6.08E+03*	**3.88E+03**	4.94E+03
f14	**3.35E+07**	*2.05E+07*	**2.14E+07**	*1.67E+07*	**1.95E+06**	*1.75E+06*	**9.82E+07**	*2.52E+07*	**8.82E+07**	*2.17E+07*	**3.05E+07**	1.96E+07
f15	**6.08E+02**	*6.08E+01*	**3.59E+02**	*4.43E+01*	**3.13E+02**	*4.08E+01*	**6.36E+02**	*8.83E+01*	**4.58E+02**	*6.14E+01*	**5.58E+02**	6.71E+01
f16	**1.79E+02**	*3.04E+01*	**1.57E+02**	*3.59E+01*	**3.93E+01**	*1.48E+01*	**2.34E+02**	*3.71E+01*	**1.31E+02**	*5.00E+01*	**2.26E+02**	3.07E+01
f17	**1.12E+01**	*9.73E+00*	**2.38E+01**	*5.42E+01*	**4.48E+02**	*8.09E+02*	**1.72E+03**	*1.41E+03*	**6.85E-01**	*2.21E-01*	**3.47E+02**	3.58E+02
f18	**1.77E+04**	*1.61E+04*	**1.83E+03**	*3.11E+03*	**5.55E+05**	*1.38E+06*	**7.70E+05**	*1.41E+06*	**7.23E+03**	*6.85E+03*	**3.04E+04**	3.44E+04
f19	**6.51E+03**	*4.03E+03*	**7.54E+03**	*4.37E+03*	**3.22E+04**	*2.67E+04*	**3.00E+04**	*1.49E+04*	**8.47E-02**	*2.32E-02*	**1.27E+04**	5.75E+03
f20	**5.45E+02**	*1.10E+03*	**5.76E+01**	*4.04E+01*	**2.51E+09**	*1.80E+09*	**5.60E+05**	*7.00E+05*	**1.73E+02**	*5.03E+01*	**2.70E+04**	6.18E+04

4.3 250 Dimensions

4.3.1 Accuracy

The algorithms performance is ranked in Table 4.7 and Table 4.8 for accuracy and consistency respectively. As the dimensionality of the benchmark problems decreases to 250 in these tests, the CSC algorithm is the clear winner in separable benchmark problems, not losing a single test to the other algorithms. The CSC algorithm shows its weakness however in a single group non-separable where it ranks last and the DCPSO ranks first followed by the MCPSO and RAG algorithm. In group sizes of $d/2m$ and d/m non-separable the MCPSO is the winner here followed closely by the RDG algorithm. Showing an increase in performance for the RDG algorithm as the dimensionality decreases. For completely non-separable benchmark problems the MCPSO and CSC algorithm tie with the DCPSO being close behind. The accuracy results overall show the MCPSO being the best performance ranking either first or second across all benchmark problems in the 250 dimension problem set.

4.3.2 Consistency

The consistency tests, shown in Table 4.8 reveal the DCPSO, MCPSO, and CSC algorithm being the most consistent. The DCPSO algorithm either wins or ties for first place in all partially non-separable problem sets and the MCPSO ties with the CSC algorithm for non-separable benchmark problems.

Table 4.7: 250 dimension statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep** W/T/L Rank	**S-Non** W/T/L Rank	**D/2m-Non** W/T/L Rank	**D/m - Non** W/T/L Rank	**Non** W/T/L Rank
DCPSO	9, 1, 5 3	18, 3, 4 1	14, 1, 10 3	13, 2, 10 3	6, 0, 4 2
MCPSO	11, 0, 4 2	13, 7, 5 2	22, 0, 3 1	19, 3, 3 1	9, 0, 1 1
RDG	4, 1, 10 4	6, 4, 15 4	15, 2, 8 2	16, 1, 8 2	2, 0, 8 3
APEC	3, 0, 12 5	9, 6, 10 3	3, 1, 21 6	5, 1, 19 6	2, 0, 8 3
CSC	15, 0, 0 1	4, 3, 18 5	11, 2, 12 4	9, 3, 13 4	9, 0, 1 1
RAG	2, 0, 13 6	13, 1, 11 2	6, 2, 17 5	6, 4, 15 5	2, 0, 8 3

The CSC algorithm is the winner for all non-separable problems being the most consistent in ever problem test.

Table 4.8: 250 dimension consistency statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep**	**S-Non**	**D/2m-Non**	**D/m - Non**	**Non**
	W/T/L	W/T/L	W/T/L	W/T/L	W/T/L
	Rank	Rank	Rank	Rank	Rank
DCPSO	9, 1, 5	14, 3, 8	18, 1, 6	15, 2, 8	6, 0, 4
	3	1	1	1	2
MCPSO	11, 0, 4	14, 7, 4	18, 0, 7	14, 3, 8	9, 0, 1
	2	1	1	2	1
RDG	6, 1, 8	5, 4, 16	9, 2, 14	12, 1, 12	1, 0, 9
	4	5	3	3	5
APEC	2, 0, 13	7, 6, 12	7, 1, 17	10, 1, 14	3, 0, 7
	5	4	4	4	3
CSC	15, 0, 0	11, 3, 11	16, 2, 7	14, 3, 8	9, 0, 1
	1	3	2	2	1
RAG	1, 0, 14	12, 1, 12	3, 2, 20	3, 4, 18	2, 0, 8
	6	2	5	5	4

Table 4.9: Average of 30 runs for dimension of 250

Function	DCPSO		MCPSO		RDG		APEC		CSC		RAG	
	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev
f1	**1.51E+03**	*7.92E+03*	**2.94E+03**	*1.10E+04*	**5.89E+03**	*1.50E+04*	**7.29E+07**	*5.40E+07*	**2.14E-01**	*8.58E-02*	**1.20E+07**	5.49E+06
f2	**4.91E+00**	*2.39E+00*	**2.58E+00**	*1.48E+00*	**5.22E+00**	*1.98E+00*	**3.25E+02**	*7.08E+01*	**1.66E-06**	*4.25E-07*	**1.09E+03**	1.01E+02
f3	**4.54E-02**	*4.53E-02*	**2.51E-02**	*4.32E-02*	**1.94E+01**	*6.02E-02*	**4.58E+00**	*1.46E+00*	**1.45E-04**	*1.64E-05*	**1.75E+01**	1.55E+00
f4	**9.13E+10**	*8.76E+10*	**1.94E+11**	*1.70E+11*	**2.34E+12**	*1.81E+12*	**2.08E+11**	*1.01E+11*	**3.86E+12**	*3.26E+12*	**4.34E+11**	8.41E+11
f5	**8.29E+07**	*2.12E+07*	**8.11E+07**	*2.53E+07*	**7.06E+07**	*2.04E+07*	**9.03E+07**	*2.76E+07*	**1.10E+08**	*4.55E+07*	**7.48E+07**	2.82E+07
f6	**5.70E+06**	*7.73E+06*	**1.67E+07**	*6.94E+06*	**1.05E+07**	*7.55E+06*	**1.19E+07**	*9.11E+06*	**1.86E+07**	*4.38E+06*	**1.37E+07**	7.89E+06
f7	**1.44E+04**	*6.74E+04*	**7.91E+03**	*6.72E+03*	**1.35E+09**	*1.79E+09*	**4.15E+05**	*1.42E+05*	**2.22E+08**	*1.89E+08*	**1.21E+05**	7.68E+04
f8	**6.78E+06**	*1.36E+07*	**4.71E+06**	*1.16E+07*	**5.54E+12**	*1.09E+13*	**2.39E+07**	*2.62E+07*	**3.62E+07**	*8.35E+06*	**7.50E+06**	1.16E+07
f9	**2.83E+07**	*8.03E+06*	**1.04E+07**	*3.84E+06*	**2.09E+07**	*2.91E+07*	**1.52E+08**	*7.47E+07*	**2.94E+07**	*9.79E+06*	**4.89E+07**	2.30E+07
f10	**9.60E+02**	*7.60E+01*	**8.61E+02**	*1.16E+02*	**6.14E+02**	*8.06E+01*	**1.29E+03**	*1.92E+02*	**1.15E+03**	*1.48E+02*	**1.52E+03**	1.00E+02
f11	**1.56E+02**	*1.05E+01*	**1.42E+02**	*2.19E+01*	**1.07E+02**	*2.07E+01*	**1.87E+02**	*7.83E+00*	**1.75E+02**	*7.79E+00*	**1.87E+02**	1.25E+01
f12	**1.28E+03**	*6.76E+02*	**6.96E+02**	*6.36E+02*	**7.88E+03**	*6.98E+03*	**2.36E+04**	*1.05E+04*	**6.26E+03**	*2.43E+03*	**9.75E+03**	5.54E+03
f13	**8.27E+03**	*7.73E+03*	**3.35E+02**	*1.07E+02*	**1.44E+08**	*2.53E+08*	**1.03E+06**	*7.24E+05*	**1.49E+03**	*3.83E+03*	**2.75E+05**	1.96E+05
f14	**1.16E+08**	*3.68E+07*	**4.19E+07**	*1.03E+07*	**3.95E+07**	*2.32E+07*	**2.03E+08**	*6.81E+07*	**1.12E+08**	*2.00E+07*	**1.02E+08**	4.35E+07
f15	**1.89E+03**	*1.12E+02*	**1.72E+03**	*1.77E+02*	**1.19E+03**	*9.06E+01*	**2.08E+03**	*1.43E+02*	**2.61E+03**	*4.25E+02*	**1.89E+03**	1.56E+02
f16	**3.37E+02**	*1.52E+01*	**2.99E+02**	*3.51E+01*	**1.70E+02**	*2.19E+01*	**3.54E+02**	*1.11E+01*	**3.64E+02**	*1.48E+01*	**3.62E+02**	1.10E+01
f17	**6.95E+03**	*2.58E+03*	**3.43E+03**	*2.35E+03*	**2.06E+04**	*9.74E+03*	**5.17E+04**	*1.99E+04*	**1.28E+04**	*3.01E+03*	**4.25E+04**	1.83E+04
f18	**2.20E+04**	*1.00E+04*	**2.32E+03**	*4.19E+03*	**3.02E+08**	*2.70E+08*	**1.20E+07**	*6.31E+06*	**2.16E+03**	*3.78E+03*	**2.39E+06**	1.93E+06
f19	**1.25E+05**	*4.32E+04*	**7.89E+04**	*2.39E+04*	**2.27E+05**	*1.34E+05*	**3.67E+05**	*8.89E+04*	**4.02E-01**	*7.13E-02*	**1.28E+05**	4.72E+04
f20	**9.11E+02**	*9.43E+02*	**1.88E+02**	*1.50E+02*	**1.26E+10**	*7.32E+09*	**1.84E+07**	*1.06E+07*	**3.80E+02**	*1.94E+02*	**2.24E+06**	1.46E+06

4.4 500 Dimensions

4.4.1 Accuracy

The algorithms are ranked in Table 4.10 and Table 4.11 for accuracy and consistency respectively. As the dimensionality of the problem, and therefore the size of separable groups decreases it is clear that for accuracy the DCPSO, MCPSO and CSC algorithms all perform well. The CSC algorithm is the clear winner in separable and non-separable problems winning all comparisons against other algorithms and only tying in a single comparison. The DCPSO and MCPSO compete for the ideal algorithm examined, for all forms of non-separable problems with the DCPSO winning in single group non-separable problems and MCPSO winning in D/2m and D/m non-separable problems. As the dimensionality decreased the DCPSO, MCPSO, RDG, APEC and RAG algorithms stayed relatively consistent between tests while the CSC algorithm took an increased drop in performance when compared to the 1000 dimensionality problem set.

4.4.2 Consistency

The consistency of the algorithms is shown in Table 4.11. In these tests, the results show a similar performance as to dimensionality of 1000 with the DCPSO and MCPSO being ranked well although the MCPSO is optimal in all cases except the non-separable problems where they presented a tie ranking. The CSC algorithm is the clear winner here again with the

consistency being ranked first in separable and non-separable problems and ranking second across all partially non-separable problems falling only behind the MCPSO in these problems. The remaining algorithms all show similar consistencies with the RAG algorithm being the least consistent in nearly all instances.

Table 4.10: 500 dimension statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep**	**S-Non**	**D/2m-Non**	**D/m - Non**	**Non**
	W/T/L	W/T/L	W/T/L	W/T/L	W/T/L
	Rank	Rank	Rank	Rank	Rank
DCPSO	7, 2, 6	18, 5, 2	16, 2, 7	15, 1, 9	6, 1, 3
	3	1	2	2	3
MCPSO	11, 1, 3	12, 4, 9	19, 1, 5	16, 2, 7	7, 2, 1
	2	2	1	1	2
RDG	5, 2, 8	7, 3, 15	13, 2, 10	15, 2, 8	2, 0, 8
	4	4	3	2	4
APEC	3, 0, 12	11, 2, 12	4, 1, 20	7, 1, 17	2, 0, 8
	5	3	6	5	4
CSC	14, 1, 0	5, 2, 18	10, 3, 12	10, 1, 14	9, 1, 0
	1	5	4	3	1
RAG	2, 0, 13	13, 2, 10	8, 1, 16	8, 1, 16	2, 0, 8
	6	2	5	4	4

Table 4.11: 500 dimension consistency statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep**	**S-Non**	**D/2m-Non**	**D/m - Non**	**Non**
	W/T/L	W/T/L	W/T/L	W/T/L	W/T/L
	Rank	Rank	Rank	Rank	Rank
DCPSO	7, 2, 6	15, 5, 5	16, 2, 7	16, 1, 8	6, 1, 3
	3	1	2	3	2
MCPSO	11, 1, 3	15, 4, 6	17, 1, 7	19, 2, 4	6, 2, 2
	2	1	1	1	2
RDG	7, 2, 6	5, 3, 17	10, 2, 13	8, 2, 15	3, 0, 7
	3	3	3	5	3
APEC	0, 0, 15	11, 2, 12	9, 1, 15	9, 1, 15	2, 0, 8
	5	2	4	4	4
CSC	14, 1, 0	11, 2, 12	16, 3, 6	17, 1, 7	9, 1, 0
	1	2	2	2	1
RAG	3, 0, 12	9, 2, 14	2, 1, 22	2, 1, 22	2, 0, 8
	4	4	5	6	4

Table 4.12: Average of 30 runs for dimension of 500

Function	DCPSO		MCPSO		RDG		APEC		CSC		RAG	
	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev
f1	**6.84E+03**	*3.53E+04*	**3.61E+01**	*5.68E+01*	**6.69E+03**	*3.45E+04*	**4.47E+08**	*1.55E+08*	**3.90E-01**	*1.16E-01*	**1.65E+08**	5.80E+07
f2	**8.66E+00**	*3.10E+00*	**5.53E+00**	*2.54E+00*	**8.61E+00**	*3.64E+00*	**8.95E+02**	*1.57E+02*	**2.78E-06**	*4.77E-07*	**3.01E+03**	1.20E+02
f3	**4.21E-02**	*2.45E-02*	**5.01E-02**	*4.14E-02*	**1.96E+01**	*4.51E-02*	**1.45E+01**	*1.58E+00*	**1.37E-04**	*1.53E-05*	**1.91E+01**	1.59E-01
f4	**5.67E+11**	*5.53E+11*	**4.37E+11**	*4.34E+11*	**1.99E+13**	*8.79E+12*	**1.49E+12**	*6.27E+11*	**1.12E+13**	*6.65E+12*	**8.40E+11**	1.04E+12
f5	**1.40E+08**	*2.82E+07*	**2.27E+08**	*6.28E+07*	**1.53E+08**	*2.97E+07*	**1.65E+08**	*3.67E+07*	**3.20E+08**	*8.10E+07*	**1.51E+08**	4.30E+07
f6	**8.12E+06**	*7.41E+06*	**1.89E+07**	*3.59E+06*	**9.46E+06**	*6.49E+06*	**1.79E+07**	*4.95E+06*	**1.98E+07**	*9.42E+04*	**1.66E+07**	6.81E+06
f7	**5.41E+04**	*2.89E+05*	**1.89E+06**	*7.60E+06*	**4.45E+09**	*4.31E+09*	**7.24E+06**	*7.96E+06*	**2.87E+09**	*1.83E+09*	**2.16E+07**	9.06E+07
f8	**3.51E+07**	*5.07E+07*	**1.57E+07**	*3.34E+07*	**1.21E+14**	*1.97E+14*	**1.18E+09**	*3.31E+09*	**8.17E+07**	*3.71E+07*	**3.10E+07**	3.68E+07
f9	**1.43E+08**	*3.85E+07*	**3.86E+07**	*1.00E+07*	**1.36E+08**	*4.06E+07*	**5.59E+08**	*2.10E+08*	**1.30E+08**	*2.12E+07*	**3.38E+08**	1.19E+08
f10	**2.09E+03**	*1.32E+02*	**2.68E+03**	*2.89E+02*	**1.61E+03**	*1.03E+02*	**2.88E+03**	*2.03E+02*	**3.50E+03**	*2.66E+02*	**3.85E+03**	2.32E+02
f11	**1.92E+02**	*4.09E+00*	**1.96E+02**	*4.69E+00*	**1.79E+02**	*1.03E+01*	**2.16E+02**	*7.96E-01*	**1.98E+02**	*4.36E-01*	**2.16E+02**	1.70E+00
f12	**1.93E+04**	*6.66E+03*	**7.06E+03**	*3.52E+03*	**7.15E+04**	*2.03E+04*	**1.45E+05**	*8.24E+04*	**5.80E+04**	*9.70E+03*	**1.20E+05**	3.17E+04
f13	**8.34E+03**	*6.78E+03*	**6.79E+02**	*6.61E+02*	**3.29E+09**	*2.30E+09*	**1.32E+07**	*4.54E+06*	**2.17E+03**	*3.61E+03*	**4.48E+06**	1.92E+06
f14	**3.00E+08**	*6.19E+07*	**1.11E+08**	*1.53E+07*	**2.38E+08**	*7.92E+07*	**5.38E+08**	*1.13E+08*	**2.24E+08**	*2.59E+07*	**3.65E+08**	1.51E+08
f15	**4.24E+03**	*1.37E+02*	**5.49E+03**	*3.84E+02*	**3.20E+03**	*1.74E+02*	**4.63E+03**	*1.75E+02*	**7.22E+03**	*5.45E+02*	**4.43E+03**	3.04E+02
f16	**3.88E+02**	*1.60E+00*	**3.87E+02**	*1.06E+01*	**3.07E+02**	*1.93E+01*	**3.94E+02**	*1.57E+00*	**3.96E+02**	*8.05E-01*	**3.92E+02**	2.21E+00
f17	**7.13E+04**	*2.54E+04*	**3.01E+04**	*1.11E+04*	**1.05E+05**	*3.17E+04*	**1.89E+05**	*6.06E+04*	**1.43E+05**	*2.50E+04*	**2.67E+05**	6.30E+04
f18	**3.24E+04**	*1.88E+04*	**4.39E+03**	*5.23E+03*	**7.06E+09**	*3.69E+09*	**1.31E+08**	*3.92E+07*	**3.39E+03**	*5.56E+03*	**3.53E+07**	1.32E+07
f19	**3.68E+05**	*1.10E+05*	**3.33E+05**	*3.82E+04*	**4.59E+05**	*8.75E+04*	**1.06E+06**	*2.00E+05*	**1.26E+00**	*3.42E-01*	**4.90E+05**	1.23E+05
f20	**1.50E+03**	*7.86E+02*	**6.48E+02**	*8.18E+02*	**2.98E+10**	*1.40E+10*	**1.48E+08**	*3.79E+07*	**6.46E+02**	*3.29E+02*	**4.37E+07**	1.12E+07

4.5 1000 Dimensions

4.5.1 Accuracy

The algorithms are ranked in Table 4.13 for accuracy and Table 4.14 for the consistency of the runs. The results demonstrate the performance of these algorithms on a large scale. Table 4.13 shows the accuracy, that is, which algorithms reached the best fitness value after the designated number of fitness evaluations. In this table, the DCPSO dominated in 3 of the 5 test sets. The DCPSO algorithm ranked first with very similar numbers to the MCPSO in the partially separable set of functions. The MCPSO and DCPSO also performed well in separable and non-separable and were both among the top performers in these sections only being tied, or slightly behind for the DCPSO, in non-separable benchmark problems. Separable benchmark problems show the CSC algorithm performing well with the MCPSO and DCPSO in second and third respectively. The static grouping technique of the RAG algorithm seemed to perform worse in the separable functions although showed some resilience in the s-non separable benchmark problems ranking second.

4.5.2 Consistency

Table 4.14 reveals the consistency of these algorithms, that is, the differences between each of the thirty runs revealing which algorithms vary the least between runs. In this, the CSC algorithm is the clear winner in 3 out of the

Table 4.13: 1000 dimension statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep** W/T/L Rank	**S-Non** W/T/L Rank	**D/2m-Non** W/T/L Rank	**D/m - Non** W/T/L Rank	**Non** W/T/L Rank
DCPSO	8,2,5 3	16, 6, 3 1	18, 0, 7 1	15, 1, 9 1	6, 0, 4 2
MCPSO	11, 1, 3 2	13, 4, 8 2	18, 2, 5 1	15, 2, 8 1	9, 0, 1 1
RDG	4, 1, 10 4	8, 1, 16 5	10, 0, 15 3	11, 3, 11 3	2, 0, 8 3
APEC	3, 0, 12 5	11, 1, 13 3	4, 2, 19 5	9, 0, 16 4	2, 0, 8 3
CSC	15, 0, 0 1	6, 2, 17 4	13, 1, 11 2	12, 2, 11 2	9, 0, 1 1
RAG	2, 0, 13 6	13, 2, 10 2	8, 3, 14 4	8, 2, 15 5	2, 0, 8 3

5 benchmark sets although lags slightly behind the MCPSO in single group non-separable functions and D/M non-separable functions. The MCPSO algorithms overall seem to perform similarly to the CSC algorithm, as they both compete for the first or second ranking in all benchmark sets. Between DCPSO and MCPSO the MCPSO algorithm tends to have slightly better consistency between runs with fewer variations.

Based on these findings for large scale optimization of 1000 dimensions the DCPSO, MCPSO, and CSC algorithms will all make viable choices with DCPSO and MCPSO being the clear winners across all forms of partially separable functions and the CSC algorithm being the ideal choice across non-separable and fully separable.

Table 4.14: 1000 dimension statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep**	**S-Non**	**D/2m-Non**	**D/m - Non**	**Non**
	W/T/L	W/T/L	W/T/L	W/T/L	W/T/L
	Rank	Rank	Rank	Rank	Rank
DCPSO	8, 2, 5	12, 6, 7	17, 0, 8	15, 1, 9	6, 0, 4
	3	4	3	3	3
MCPSO	9, 1, 5	15, 4, 6	18, 2, 5	19, 2, 4	8, 0, 2
	2	1	2	1	2
RDG	8, 1, 6	5, 1, 19	9, 0, 16	7, 3, 15	2, 0, 8
	3	6	4	5	4
APEC	0, 0, 15	13, 1, 11	6, 2, 17	9, 0, 16	2, 0, 8
	5	3	5	4	4
CSC	15, 0, 0	14, 2, 9	19, 1, 5	18, 2, 5	10, 0, 0
	1	2	1	2	1
RAG	3, 0, 12	8, 2, 15	2, 3, 20	2, 2, 21	2, 0, 8
	4	5	6	6	4

Table 4.15: Average of 30 runs for dimension of 1000

Function	DCPSO		MCPSO		RDG		APEC		CSC		RAG	
	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev
f1	**5.49E+04**	*1.36E+05*	**4.87E+02**	*5.11E+02*	**8.12E+04**	*1.65E+05*	**2.77E+09**	*6.02E+08*	**6.77E-01**	*1.50E-01*	**1.47E+09**	3.24E+08
f2	**1.97E+01**	*4.85E+00*	**1.30E+01**	*4.12E+00*	**1.86E+01**	*3.63E+00*	**2.19E+03**	*2.74E+02*	**4.44E-06**	*4.63E-07*	**7.30E+03**	2.05E+02
f3	**3.94E-02**	*2.23E-02*	**5.30E-02**	*4.54E-02*	**1.98E+01**	*3.70E-02*	**1.59E+01**	*7.48E-01*	**1.22E-04**	*5.38E-06*	**1.94E+01**	4.77E-02
f4	**3.00E+12**	*2.67E+12*	**1.03E+12**	*5.93E+11*	**9.64E+13**	*3.51E+13*	**6.58E+12**	*2.56E+12*	**7.90E+12**	*2.15E+12*	**4.60E+12**	4.00E+12
f5	**3.51E+08**	*4.52E+07*	**6.66E+08**	*1.19E+08*	**4.18E+08**	*5.92E+07*	**4.64E+08**	*7.10E+07*	**7.44E+08**	*1.18E+08*	**3.69E+08**	7.08E+07
f6	**1.09E+07**	*6.60E+06*	**1.98E+07**	*6.91E+04*	**1.39E+07**	*5.50E+06*	**1.93E+07**	*4.54E+05*	**1.99E+07**	*7.19E+04*	**1.89E+07**	2.95E+06
f7	**2.18E+08**	*2.28E+08*	**1.22E+08**	*2.19E+08*	**9.48E+10**	*4.46E+10*	**2.53E+08**	*2.11E+08*	**1.16E+10**	*7.63E+09*	**6.03E+08**	8.75E+08
f8	**4.95E+07**	*4.19E+07*	**1.35E+07**	*1.29E+07*	**8.75E+14**	*1.65E+15*	**8.64E+08**	*2.33E+09*	**8.01E+07**	*2.58E+07*	**5.58E+07**	1.98E+08
f9	**2.81E+08**	*4.69E+07*	**7.56E+07**	*1.20E+07*	**1.04E+09**	*1.88E+08*	**2.38E+09**	*5.74E+08*	**1.51E+08**	*1.99E+07*	**2.33E+09**	6.16E+08
f10	**4.56E+03**	*1.48E+02*	**6.99E+03**	*4.44E+02*	**4.14E+03**	*1.70E+02*	**6.54E+03**	*2.89E+02*	**7.65E+03**	*4.30E+02*	**8.32E+03**	2.78E+02
f11	**1.94E+02**	*4.83E-01*	**1.99E+02**	*3.08E-01*	**2.09E+02**	*3.65E+00*	**2.17E+02**	*7.80E-01*	**1.98E+02**	*2.46E-01*	**2.16E+02**	8.08E-01
f12	**1.81E+05**	*4.65E+04*	**4.89E+04**	*1.59E+04*	**4.92E+05**	*6.18E+04*	**6.64E+05**	*2.49E+05*	**2.29E+05**	*2.81E+04*	**8.52E+05**	1.77E+05
f13	**1.77E+04**	*1.01E+04*	**8.98E+02**	*1.54E+03*	**2.32E+10**	*7.13E+09*	**7.71E+07**	*2.26E+07*	**9.45E+02**	*1.05E+03*	**2.77E+07**	1.13E+07
f14	**8.40E+08**	*1.03E+08*	**2.39E+08**	*2.81E+07*	**1.66E+09**	*2.20E+08*	**1.49E+09**	*2.12E+08*	**3.69E+08**	*3.10E+07*	**2.00E+09**	6.09E+08
f15	**8.85E+03**	*2.39E+02*	**1.37E+04**	*5.44E+02*	**7.51E+03**	*1.90E+02*	**9.42E+03**	*2.50E+02*	**1.52E+04**	*4.66E+02*	**9.24E+03**	4.53E+02
f16	**3.90E+02**	*5.62E-01*	**3.97E+02**	*3.93E-01*	**3.83E+02**	*3.75E+00*	**3.96E+02**	*1.01E+00*	**3.97E+02**	*2.77E-01*	**3.93E+02**	1.26E+00
f17	**7.17E+05**	*1.36E+05*	**1.89E+05**	*2.14E+04*	**7.05E+05**	*8.46E+04*	**9.79E+05**	*1.94E+05*	**5.17E+05**	*4.48E+04*	**1.10E+06**	2.27E+05
f18	**4.01E+04**	*1.74E+04*	**2.83E+03**	*2.10E+03*	**5.19E+10**	*1.43E+10*	**1.04E+09**	*1.72E+08*	**4.01E+03**	*4.80E+03*	**2.98E+08**	6.59E+07
f19	**1.60E+06**	*2.81E+05*	**1.21E+06**	*8.62E+04*	**2.16E+06**	*5.62E+05*	**3.76E+06**	*9.82E+05*	**1.24E+02**	*1.25E+01*	**1.61E+06**	2.17E+05
f20	**3.66E+03**	*1.52E+03*	**1.15E+03**	*4.68E+02*	**9.48E+10**	*3.03E+10*	**1.10E+09**	*1.57E+08*	**1.71E+03**	*2.19E+02*	**3.13E+08**	5.09E+07

4.6 2000 Dimensions

4.6.1 Accuracy

The 2000 dimension tests are ranked in Table 4.16 for accuracy and Table 4.17 for consistency. W/T/L refers to the wins, ties and losses from each algorithm across each problem set. These results portray the performance of the algorithms on the largest scale of this study. Table 4.16 discusses the accuracy, in these tests DCPSO ranks the best in $D/m - NonSeparable$ problems, and shows a second place performance for almost all other benchmark sets except completely non-separable where the performance is second from the last. The MCPSO algorithm, performed the best over fully separable, partially non-separable and received a second and third place ranking in non-separable and single group non-separable, respectively. The RDG and APEC algorithms performed poor across almost all benchmark problems tested, ranking in the bottom half of algorithms in all tests. The CSC algorithm, tied in fully separable problems with MCPSO and ranked first in fully non-separable. The CSC algorithm ranked third across both partially non-separable groups and ranked second last in single group non-separable benchmark problems. Finally, the RAG algorithm performed well across single group non-separable benchmark problems, and received a second place ranking in $D/mnon-separable$ problems, the other three rankings all showed a poor performance ranking in the bottom half of algorithms against all.

Table 4.16: 2000 dimensions statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep** W/T/L Rank	**S-Non** W/T/L Rank	**D/2m-Non** W/T/L Rank	**D/m - Non** W/T/L Rank	**Non** W/T/L Rank
DCPSO	9, 2, 4 2	18, 2, 5 2	16, 0, 9 2	15, 2, 8 1	5, 0, 5 4
MCPSO	12, 0, 3 1	12, 4, 9 3	18, 0, 7 1	15, 1, 9 1	7, 1, 2 2
RDG	5, 2, 8 3	2, 1, 22 6	2, 2, 21 6	8, 1, 16 4	1, 0, 9 5
APEC	3, 0, 12 4	11, 1, 13 4	14, 0, 11 4	8, 1, 16 4	1, 0, 9 5
CSC	12, 0, 3 1	5, 3, 17 5	15, 1, 9 3	12, 1, 12 3	9, 1, 0 1
RAG	2, 0, 13 5	21, 1, 3 1	8, 1, 16 5	13, 2, 10 2	6, 0, 4 3

4.6.2 Consistency

The consistency tests across 2000 dimensions are shown in Table 4.17, this table shows the consistency of each test throughout the 30 runs, that is, how much variation is in each result from the other 29 tests of that algorithm. In these tests, the DCPSO performed in the second or third place ranking across all benchmark problems, putting it as one of the better algorithms for consistency in the results. The MCPSO algorithm, ranked first across all three of the partially non-separable function groups. This algorithm performed second in both the separable and fully non-separable benchmark problems. The RDG and APEC algorithm, again show deterioration in the results as the dimensionalty of the problems reaches 2000 and both performed in the bottom

half of the algorithms tested. The CSC algorithm, showed poor consistency performance in the partially non-separable benchmark problems, ranking in the bottom half, however, showed top performance in the fully separable and fully non-separable groups with a rank of first, and either showing a win or tie across all functions in each group. The RAG algorithm, performed poorly with all groups being in the bottom half of the algorithms except single group non-separable which achieved a second place ranking.

Table 4.17: 2000 dimension consistency statistical tests. W/T/L refers to the total Wins, Ties and Losses respectively

Algorithm	**Sep** W/T/L Rank	**S-Non** W/T/L Rank	**D/2m-Non** W/T/L Rank	**D/m - Non** W/T/L Rank	**Non** W/T/L Rank
DCPSO	7, 2, 6 3	13, 2, 10 3	15, 0, 10 2	13, 2, 10 2	5, 0, 5 3
MCPSO	9, 0, 6 2	16, 4, 5 1	20, 0, 5 1	21, 1, 3 1	8, 1, 1 2
RDG	6, 2, 7 4	7, 1, 17 6	5, 2, 18 6	7, 1, 17 4	1, 0, 9 4
APEC	0, 0, 15 5	8, 1, 16 5	12, 0, 13 3	6, 1, 18 5	1, 0, 9 4
CSC	15, 0, 0 1	10, 3, 12 4	11, 1, 13 4	12, 1, 12 3	9, 1, 0 1
RAG	6, 0, 9 4	15, 1, 9 2	10, 1, 14 5	12, 2, 11 3	5, 0, 5 3

Table 4.18: Average of 30 runs for dimension of 2000

Function	DCPSO		MCPSO		RDG		APEC		CSC		RAG	
	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev	Avg	St.Dev
f1	**4.06E+04**	*1.13E+05*	**6.31E+03**	*6.93E+03*	**9.33E+04**	*1.97E+05*	**1.06E+10**	*2.34E+09*	**1.14E+03**	*9.16E+01*	**7.86E+09**	1.14E+09
f2	**3.60E+01**	*5.38E+00*	**2.93E+01**	*3.87E+00*	**3.52E+01**	*6.27E+00*	**6.03E+03**	*3.47E+02*	**4.76E+01**	*3.46E+00*	**1.40E+04**	1.95E+02
f3	**4.15E-02**	*1.85E-02*	**7.73E-02**	*3.51E-02*	**2.01E+01**	*3.14E-02*	**1.68E+01**	*4.40E-01*	**3.79E-03**	*5.45E-05*	**1.93E+01**	1.67E-02
f4	**8.23E+12**	*3.66E+12*	**1.94E+12**	*6.69E+11*	**4.95E+14**	*1.37E+14*	**1.58E+13**	*1.30E+13*	**1.80E+13**	*5.45E+12*	**1.46E+12**	6.86E+11
f5	**6.46E+08**	*6.31E+07*	**1.39E+09**	*1.79E+08*	**9.51E+08**	*4.71E+07*	**9.14E+08**	*1.10E+08*	**1.40E+09**	*2.71E+08*	**8.01E+08**	1.16E+08
f6	**1.90E+07**	*1.49E+06*	**1.99E+07**	*4.08E+04*	**2.08E+07**	*1.21E+05*	**1.96E+07**	*1.40E+05*	**1.99E+07**	*5.54E+04*	**1.98E+07**	1.21E+05
f7	**1.98E+09**	*1.74E+09*	**6.32E+08**	*7.32E+08*	**2.48E+11**	*6.02E+10*	**4.80E+09**	*3.97E+09*	**6.64E+10**	*3.04E+10*	**8.20E+06**	1.44E+07
f8	**2.64E+07**	*3.17E+07*	**7.99E+06**	*1.24E+07*	**3.86E+15**	*2.33E+15*	**1.06E+11**	*2.56E+11*	**2.32E+07**	*1.48E+07*	**7.87E+06**	3.25E+07
f9	**6.41E+08**	*7.86E+07*	**1.64E+08**	*1.73E+07*	**6.05E+09**	*7.39E+08*	**2.79E+06**	*2.85E+06*	**3.53E+08**	*1.05E+08*	**4.35E+08**	4.46E+07
f10	**8.99E+03**	*2.30E+02*	**1.48E+04**	*6.98E+02*	**9.99E+03**	*2.63E+02*	**5.84E+01**	*3.24E+01*	**1.09E+04**	*2.16E+03*	**1.63E+04**	7.09E+02
f11	**1.95E+02**	*2.23E-01*	**1.99E+02**	*1.43E-01*	**2.18E+02**	*4.20E-01*	**2.16E+02**	*4.43E-01*	**1.99E+02**	*1.89E-01*	**2.18E+02**	8.88E-01
f12	**7.85E+05**	*1.71E+05*	**2.72E+05**	*3.29E+04*	**2.23E+06**	*1.75E+05*	**1.82E+06**	*1.27E+06*	**1.20E+06**	*2.18E+05*	**3.98E+05**	5.18E+04
f13	**1.73E+04**	*1.19E+04*	**2.07E+03**	*3.79E+03*	**1.08E+11**	*1.43E+10*	**8.56E+08**	*1.65E+08*	**5.20E+03**	*3.61E+03*	**7.58E+04**	1.52E+04
f14	**1.46E+09**	*1.88E+08*	**5.07E+08**	*3.21E+07*	**8.54E+09**	*6.83E+08*	**2.32E+09**	*2.42E+08*	**8.59E+08**	*2.35E+08*	**1.22E+09**	5.96E+07
f15	**1.79E+04**	*2.79E+02*	**2.97E+04**	*8.91E+02*	**1.78E+04**	*3.71E+02*	**1.89E+04**	*3.84E+02*	**2.36E+04**	*5.20E+03*	**1.86E+04**	1.39E+03
f16	**3.91E+02**	*4.68E-01*	**3.97E+02**	*2.24E-01*	**3.94E+02**	*9.86E-01*	**3.95E+02**	*6.94E-01*	**3.97E+02**	*6.32E-01*	**3.96E+02**	1.86E+00
f17	**2.09E+06**	*3.22E+05*	**1.02E+06**	*7.05E+04*	**3.23E+06**	*2.71E+05*	**2.39E+06**	*4.80E+05*	**1.85E+06**	*1.17E+05*	**1.13E+06**	8.43E+04
f18	**3.25E+04**	*1.47E+04*	**4.74E+03**	*4.13E+03*	**2.21E+11**	*4.21E+10*	**5.25E+09**	*6.79E+08*	**8.96E+03**	*5.99E+03*	**4.27E+04**	8.43E+03
f19	**4.41E+06**	*8.60E+05*	**3.75E+06**	*1.45E+05*	**6.02E+06**	*1.26E+06*	**7.33E+06**	*1.28E+06*	**1.38E+01**	*2.60E+00*	**2.97E+06**	2.84E+05
f20	**7.08E+03**	*1.63E+03*	**2.61E+03**	*7.74E+02*	**3.17E+11**	*4.47E+10*	**5.60E+09**	*4.13E+08*	**2.33E+03**	*1.31E+03*	**3.34E+04**	3.12E+03

Chapter 5

Conclusion

The paper reviewed six current decomposition algorithms and implemented them into a CPSO framework to address large-scale global optimization problems. The DCPSO algorithm decomposed problems at a fixed rate until optimizing as an n-dimensional PSO. The MCPSO algorithm merged the subsets together starting at an n-dimensional PSO and ended as a PSO. The RDG algorithm used prior fitness evaluations to attempt to determine groupings prior to optimizing and then optimized the problems based on the discovered groupings. The CSC algorithm was the only algorithm to adaptively determine the grouping sizes based on prior knowledge as well as spend more time optimizing variables that tended to make bigger improvements to the fitness. The APEC algorithm used a clustering approach that attempted to cluster groups dynamically based on prior knowledge without using any fitness evaluations. Finally, the RAG algorithm is a static algorithm that has

a fixed grouping size and attempts to determine the top-performing groups and randomly mix the groups not performing well.

The results showed the MCPSO, DCPSO, and the adaptive approach of the CSC algorithm to perform best in the majority of cases with each of these algorithms performing very well in almost all cases when compared to the algorithms chosen. The CSC algorithm overall tended to perform best in the separable and non-separable problems due to its adaptive features and ability to spend more time optimizing groupings that perform better and fewer fitness evaluations on undesirable groupings. The MCPSO and DCPSO however, tended to perform best in the majority of partially separable benchmark problems as a result of the constant regrouping. The RDG algorithm seemed to be a better performer in the lower dimension problems but seemed to stagnate as the group sizes got larger and no regrouping of the variables happen to get it past stagnation. The other two algorithms performed mediocrely in most tests and only performed the best in very few if any of the tests.

Given the success of the MCPSO, DCPSO and CSC algorithms future work should investigate hybrid approaches to these. The adaptive features of the CSC algorithm and the constant regrouping of the MCPSO and DCPSO could create better-performing algorithms which are ideal on a vast majority of large and small scale benchmark problems. Research into a larger number of algorithms and a variety of PSO parameters could also be looked into to determine optimal configurations for large scale optimization.

Bibliography

[1] H.-C. Kuo and C.-H. Lin, "A directed genetic algorithm for global optimization," *Applied Mathematics and Computation*, vol. 219, no. 14, pp. 7348–7364, 2013.

[2] J. Douglas, A. Engelbrecht, and B. Ombuki-Berman, "Merging and decomposition variants of cooperative particle swarm optimization: New algorithms for large scale optimization problems," *ISMSI '18: Proceedings of the 2nd International Conference on Intelligent Systems, Metaheuristics Swarm Intelligence*, pp. 70–77, 03 2018.

[3] I. Fister, I. J. Fister, and J. B. Žumer, "Memetic artificial bee colony algorithm for large-scale global optimization," pp. 1–8, 2012.

[4] H. Ismkhan, "Effective heuristics for ant colony optimization to handle large-scale problems," *Swarm and Evolutionary Computation*, vol. 32, pp. 140–149, 2017.

[5] J. Brest, A. Zamuda, I. Fister, and M. S. Maučec, "Large scale global optimization using self-adaptive differential evolution algorithm," pp. 1–8, 2010.

[6] H.-H. Xu and R.-L. Tang, "Particle swarm optimization with adaptive elite opposition-based learning for large-scale problems," pp. 44–49, 2020.

[7] Z. Wu, X. Xia, and B. Wang, "Improving building energy efficiency by multiobjective neighborhood field optimization," *Energy and Buildings*, vol. 87, pp. 45–56, 2015.

[8] R. Bellman, "Dynamic programming," *Princeton University Press, Princeton, NJ, USA.*, 1957.

[9] YewSoon Ong, A. J. Keane, and P. B. Nair, "Surrogate-assisted coevolutionary search," vol. 3, pp. 1140–1145 vol.3, 2002.

[10] E. Sayed, D. Essam, R. Sarker, and S. Elsayed, "Decomposition-based evolutionary algorithm for large scale constrained problems," *Information Sciences*, vol. 316, pp. 457 – 486, 2015.

[11] F. Vandenbergh and A. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, p. 225–239, 2004.

[12] Y.-j. Shi, H.-f. Teng, and Z.-q. Li, "Cooperative co-evolutionary differential evolution for function optimization," pp. 1080–1088, 2005.

[13] Q. Yang, W.-N. Chen, and J. Zhang, "Evolution consistency based decomposition for cooperative coevolution," *IEEE Access*, vol. 6, p. 51084–51097, 2018.

[14] H. Ge, L. Sun, K. Zhang, and C. Wu, "Cooperative differential evolution framework with utility-based adaptive grouping for large-scale optimization," *Advances in Mechanical Engineering*, vol. 11, no. 3, 2019.

[15] Y. Sun, M. Kirley, and S. K. Halgamuge, "A recursive decomposition method for large scale continuous optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 5, p. 647–661, 2018.

[16] E. Sopov and A. Vakhnin, "An investigation of parameter tuning in the random adaptive grouping algorithm for lsgo problems," *Proceedings of the 10th International Joint Conference on Computational Intelligence*, 2018.

[17] Zhenyu Yang, Ke Tang, and Xin Yao, "Self-adaptive differential evolution with neighborhood search," pp. 1110–1116, 2008.

[18] L. Li, W. Fang, Q. Wang, and J. Sun, "Differential grouping with spectral clustering for large scale global optimization," pp. 334–341, 2019.

[19] L. Li, W. Fang, Y. Mei, and Q. Wang, "Cooperative coevolution for large-scale global optimization based on fuzzy decomposition," *Soft Computing*, Nov 2020.

[20] J. Douglas, "Merging and decomposition variants of cooperative particle swarm optimization for large scale problems," *MSc Thesis, Brock University Print*.

[21] E. Sopov, A. Vakhnin, and E. Semenkin, "On tuning group sizes in the random adaptive grouping algorithm for large-scale global optimization problems," *2018 International Conference on Applied Mathematics Computational Science*, pp. 134–13 411, 2018.

[22] S. Mahdavi, M. E. Shiri, and S. Rahnamayan, "Metaheuristics in large-scale global continues optimization: A survey," *Information Sciences*, vol. 295, pp. 407–428, 2015.

[23] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," vol. 2, pp. 1101–1108, 2001.

[24] M. El-Abd, "A cooperative approach to the artificial bee colony algorithm," *2010 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–5, 07 2010.

[25] G. A. Trunfio, "Enhancing the firefly algorithm through a cooperative coevolutionary approach: an empirical study on benchmark optimisation problems," *International Journal of Bio-Inspired Computation*, pp. 108 – 125, 2014.

[26] X. Li and X. Yao, "Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms," pp. 1546–1553, 2009.

[27] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," vol. 4, pp. 1942–1948, 1995.

[28] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," *IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 69–73, 1998.

[29] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[30] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 1110–1116, 2008.

[31] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," *Proceedings of the 2000 Congress on Evolutionary Computation*, vol. 1, pp. 84–88, 2000.

[32] K. R. Harrison, A. P. Engelbrecht, and B. M. Ombuki-Berman, "An adaptive particle swarm optimization algorithm based on optimal pa-

rameter regions," *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, 2017.

[33] K. Tang, L. Xiaodong, P. Suganthan, Y. Zhenyu, and T. Weise, "Benchmark functions for the cec'2010 special session and competition on large-scale global optimization," Nov 2009.

[34] "Holms sequential bonferroni procedure," *Encyclopedia of Research Design.*

[35] "The friedman two-way analysis of variance by ranks," *Handbook of Parametric and Nonparametric Statistical Procedures*, 2003.

Appendix A

Table A.1: Results for MCPSO Velocity Clamping Threshold

Function	**None**		**0.1**		**0.01**	
	Avg	**St.Dev**	**Avg**	**St.Dev**	**Avg**	**St.Dev**
f1	2.57E-02	*3.72E-02*	4.87E+02	*5.11E+02*	6.50E+01	*2.28E+02*
f2	8.29E-01	*8.93E-01*	1.30E+01	*4.12E+00*	8.39E+01	*1.12E+01*
f3	3.06E-06	*1.02E-06*	5.30E-02	*4.54E-02*	1.18E+00	*1.12E-01*
f4	1.12E+12	*5.37E+11*	1.03E+12	*5.93E+11*	5.08E+11	*2.26E+11*
f5	7.12E+08	*1.58E+08*	6.66E+08	*1.19E+08*	6.65E+08	*1.54E+08*
f6	1.99E+07	*4.68E+04*	1.98E+07	*6.91E+04*	1.98E+07	*6.65E+04*
f7	2.78E+05	*1.45E+05*	1.22E+08	*2.19E+08*	3.64E+05	*1.74E+06*
f8	5.40E+07	*4.52E+07*	1.35E+07	*1.29E+07*	5.43E+05	*2.08E+06*
f9	8.96E+07	*9.96E+06*	7.56E+07	*1.20E+07*	7.68E+07	*8.10E+06*
f10	6.90E+03	*3.63E+02*	6.99E+03	*4.44E+02*	7.08E+03	*3.87E+02*
f11	1.98E+02	*2.95E-01*	1.99E+02	*3.08E-01*	2.00E+02	*3.81E-01*
f12	1.74E+04	*2.78E+03*	4.89E+04	*1.59E+04*	2.88E+04	*3.47E+03*
f13	1.56E+03	*4.86E+02*	8.98E+02	*1.54E+03*	8.68E+02	*2.96E+02*
f14	3.10E+08	*2.67E+07*	2.39E+08	*2.81E+07*	2.87E+08	*2.74E+07*
f15	1.40E+04	*5.71E+02*	1.37E+04	*5.44E+02*	1.38E+04	*6.61E+02*
f16	3.97E+02	*3.25E-01*	3.97E+02	*3.93E-01*	3.97E+02	*4.08E-01*
f17	1.17E+05	*1.03E+04*	1.89E+05	*2.14E+04*	1.42E+05	*1.20E+04*
f18	4.47E+03	*1.04E+03*	2.83E+03	*2.10E+03*	3.99E+03	*1.32E+03*
f19	1.12E+06	*7.58E+04*	1.21E+06	*8.62E+04*	1.05E+06	*5.25E+04*
f20	1.42E+03	*1.87E+02*	1.15E+03	*4.68E+02*	1.81E+03	*2.46E+02*

Table A.2: Results for DCPSO Velocity Clamping Threshold

Function	**None**		**0.1**		**0.01**	
	Avg	**St.Dev**	**Avg**	**St.Dev**	**Avg**	**St.Dev**
f1	1.15E+02	*2.95E+01*	5.49E+04	*1.36E+05*	6.59E+02	*5.70E+02*
f2	1.02E+00	*7.66E-01*	1.97E+01	*4.85E+00*	7.55E+01	*1.14E+01*
f3	3.08E-02	*7.30E-03*	3.94E-02	*2.23E-02*	1.55E+00	*3.16E-01*
f4	1.36E+12	*5.48E+11*	3.00E+12	*2.67E+12*	3.80E+11	*2.83E+11*
f5	3.13E+08	*8.21E+07*	3.51E+08	*4.52E+07*	4.18E+08	*4.82E+07*
f6	1.05E+07	*6.20E+06*	1.09E+07	*6.60E+06*	1.91E+07	*1.61E+05*
f7	3.20E+04	*4.14E+04*	2.18E+08	*2.28E+08*	3.48E+05	*7.35E+05*
f8	1.28E+10	*6.80E+10*	4.95E+07	*4.19E+07*	8.99E+06	*2.26E+07*
f9	1.26E+08	*1.66E+07*	2.81E+08	*4.69E+07*	6.03E+07	*6.59E+06*
f10	6.47E+03	*3.59E+02*	4.56E+03	*1.48E+02*	4.18E+03	*9.79E+01*
f11	1.98E+02	*3.68E-01*	1.94E+02	*4.83E-01*	1.96E+02	*6.92E-01*
f12	4.01E+04	*6.63E+03*	1.81E+05	*4.65E+04*	1.18E+04	*1.99E+03*
f13	3.06E+05	*5.15E+05*	1.77E+04	*1.01E+04*	3.67E+03	*2.10E+03*
f14	4.00E+08	*3.67E+07*	8.40E+08	*1.03E+08*	2.04E+08	*1.59E+07*
f15	1.28E+04	*4.57E+02*	8.85E+03	*2.39E+02*	8.00E+03	*1.78E+02*
f16	3.96E+02	*4.86E-01*	3.90E+02	*5.62E-01*	3.87E+02	*3.75E-01*
f17	1.79E+05	*2.36E+04*	7.17E+05	*1.36E+05*	7.69E+04	*6.64E+03*
f18	1.09E+06	*7.67E+05*	4.01E+04	*1.74E+04*	2.93E+04	*1.34E+04*
f19	1.94E+06	*4.59E+05*	1.60E+06	*2.81E+05*	7.35E+05	*7.33E+04*
f20	3.59E+04	*9.21E+04*	3.66E+03	*1.52E+03*	2.50E+03	*1.45E+02*

Table A.3: Results for CSC Velocity Clamping Threshold

Function	**None**		**0.1**		**0.01**	
	Avg	**St.Dev**	**Avg**	**St.Dev**	**Avg**	**St.Dev**
f1	1.67E+00	*4.76E-01*	6.77E-01	*1.50E-01*	5.20E-01	*1.05E-01*
f2	1.08E-05	*1.79E-06*	4.44E-06	*4.63E-07*	5.24E+00	*1.41E+00*
f3	2.02E-04	*2.01E-05*	1.22E-04	*5.38E-06*	1.07E-04	*5.01E-06*
f4	3.05E+13	*9.97E+12*	7.90E+12	*2.15E+12*	9.30E+12	*2.72E+12*
f5	8.55E+08	*1.01E+08*	7.44E+08	*1.18E+08*	7.20E+08	*1.54E+08*
f6	1.98E+07	*4.71E+04*	1.99E+07	*7.19E+04*	1.98E+07	*8.84E+04*
f7	9.83E+10	*7.52E+10*	1.16E+10	*7.63E+09*	2.27E+10	*1.50E+10*
f8	4.69E+09	*4.94E+09*	8.01E+07	*2.58E+07*	7.70E+07	*2.30E+07*
f9	4.09E+08	*5.67E+07*	1.51E+08	*1.99E+07*	1.78E+08	*4.91E+07*
f10	7.63E+03	*4.56E+02*	7.65E+03	*4.30E+02*	5.46E+03	*1.16E+03*
f11	1.99E+02	*2.13E-01*	1.98E+02	*2.46E-01*	1.98E+02	*4.37E-01*
f12	5.56E+05	*1.13E+05*	2.29E+05	*2.81E+04*	3.21E+05	*7.57E+04*
f13	1.34E+06	*1.19E+06*	9.45E+02	*1.05E+03*	4.58E+02	*1.10E+02*
f14	9.18E+08	*1.09E+08*	3.69E+08	*3.10E+07*	4.22E+08	*6.43E+07*
f15	1.52E+04	*5.09E+02*	1.52E+04	*4.66E+02*	1.25E+04	*2.41E+03*
f16	3.97E+02	*2.73E-01*	3.97E+02	*2.77E-01*	3.97E+02	*7.12E-01*
f17	1.24E+06	*2.09E+05*	5.17E+05	*4.48E+04*	5.78E+05	*1.57E+05*
f18	1.94E+06	*1.59E+06*	4.01E+03	*4.80E+03*	1.24E+04	*6.10E+03*
f19	2.71E+01	*3.92E+00*	1.24E+02	*1.25E+01*	3.34E+04	*3.25E+03*
f20	1.57E+03	*5.55E+02*	1.71E+03	*2.19E+02*	8.27E+02	*4.51E+02*

Table A.4: Results for APEC Velocity Clamping Threshold

Function	**None**		**0.1**		**0.01**	
	Avg	**St.Dev**	**Avg**	**St.Dev**	**Avg**	**St.Dev**
f1	8.04E+05	*3.76E+05*	2.77E+09	*6.02E+08*	3.95E+09	*1.20E+09*
f2	1.53E+02	*1.99E+01*	2.19E+03	*2.74E+02*	8.91E+02	*8.32E+01*
f3	1.40E+00	*3.41E-01*	1.59E+01	*7.48E-01*	1.18E+01	*2.16E-01*
f4	1.55E+13	*9.98E+12*	6.58E+12	*2.56E+12*	6.98E+13	*4.58E+13*
f5	4.93E+08	*1.05E+08*	4.64E+08	*7.10E+07*	4.78E+08	*4.58E+13*
f6	1.98E+07	*4.28E+05*	1.93E+07	*4.54E+05*	1.89E+07	*2.37E+08*
f7	4.39E+08	*4.37E+08*	2.53E+08	*2.11E+08*	6.41E+10	*3.68E+10*
f8	4.27E+09	*2.05E+10*	8.64E+08	*2.33E+09*	1.20E+14	*8.49E+13*
f9	7.03E+08	*1.23E+08*	2.38E+09	*5.74E+08*	4.19E+09	*8.49E+13*
f10	6.18E+03	*3.33E+02*	6.54E+03	*2.89E+02*	4.94E+03	*2.30E+09*
f11	2.17E+02	*8.22E-01*	2.17E+02	*7.80E-01*	2.11E+02	*2.37E+03*
f12	3.28E+05	*9.79E+04*	6.64E+05	*2.49E+05*	1.32E+06	*8.53E+05*
f13	2.34E+05	*3.68E+05*	7.71E+07	*2.26E+07*	2.87E+09	*1.60E+09*
f14	1.56E+09	*2.30E+08*	1.49E+09	*2.12E+08*	2.92E+09	*7.93E+08*
f15	1.23E+04	*4.55E+02*	9.42E+03	*2.50E+02*	8.56E+03	*1.50E+09*
f16	3.96E+02	*8.66E-01*	3.96E+02	*1.01E+00*	3.91E+02	*4.09E+03*
f17	1.07E+06	*2.17E+05*	9.79E+05	*1.94E+05*	1.47E+06	*7.89E+05*
f18	1.20E+06	*9.75E+05*	1.04E+09	*1.72E+08*	8.50E+09	*4.36E+09*
f19	8.78E+06	*5.02E+06*	3.76E+06	*9.82E+05*	5.49E+06	*4.36E+09*
f20	6.50E+04	*8.61E+04*	1.10E+09	*1.57E+08*	8.99E+09	*4.66E+09*

Table A.5: Results for RDG Velocity Clamping Threshold

Function	**None**		**0.1**		**0.01**	
	Avg	**St.Dev**	**Avg**	**St.Dev**	**Avg**	**St.Dev**
f1	1.79E-07	*1.40E-07*	8.12E+04	*1.65E+05*	9.40E-03	*6.10E-03*
f2	1.99E-01	*3.98E-01*	1.86E+01	*3.63E+00*	7.87E+01	*9.50E+00*
f3	2.01E+01	*4.59E-01*	1.98E+01	*3.70E-02*	1.97E+01	*2.90E-02*
f4	4.07E+14	*2.48E+14*	9.64E+13	*3.51E+13*	3.64E+14	*1.36E+14*
f5	5.75E+08	*8.60E+07*	4.18E+08	*5.92E+07*	5.18E+08	*8.87E+07*
f6	1.53E+07	*4.09E+06*	1.39E+07	*5.50E+06*	1.92E+07	*2.38E+05*
f7	2.63E+11	*6.74E+10*	9.48E+10	*4.46E+10*	5.88E+11	*2.85E+11*
f8	2.51E+15	*1.52E+15*	8.75E+14	*1.65E+15*	7.76E+15	*2.79E+15*
f9	2.11E+09	*3.21E+08*	1.04E+09	*1.88E+08*	2.26E+09	*5.15E+08*
f10	5.26E+03	*2.03E+02*	4.14E+03	*1.70E+02*	4.03E+03	*1.73E+02*
f11	2.18E+02	*1.97E+00*	2.09E+02	*3.65E+00*	2.13E+02	*7.83E-01*
f12	8.37E+05	*9.67E+04*	4.92E+05	*6.18E+04*	6.48E+05	*6.81E+04*
f13	1.78E+10	*9.98E+09*	2.32E+10	*7.13E+09*	1.82E+10	*4.53E+09*
f14	2.04E+09	*3.48E+08*	1.66E+09	*2.20E+08*	1.19E+09	*1.46E+08*
f15	9.63E+03	*4.06E+02*	7.51E+03	*1.90E+02*	7.27E+03	*2.16E+02*
f16	3.96E+02	*1.86E+00*	3.83E+02	*3.75E+00*	3.86E+02	*8.68E-01*
f17	7.94E+05	*5.66E+04*	7.05E+05	*8.46E+04*	4.72E+05	*4.33E+04*
f18	1.32E+09	*1.44E+09*	5.19E+10	*1.43E+10*	1.94E+09	*9.14E+08*
f19	4.68E+06	*1.22E+06*	2.16E+06	*5.62E+05*	7.76E+05	*1.04E+05*
f20	8.12E+12	*1.13E+12*	9.48E+10	*3.03E+10*	5.23E+10	*1.09E+10*

Table A.6: Results for RAG Velocity Clamping Threshold

Function	None		0.1		0.01	
	Avg	**St.Dev**	**Avg**	**St.Dev**	**Avg**	**St.Dev**
f1	8.38E+01	*1.58E+02*	1.47E+09	*3.24E+08*	1.81E+04	*1.22E+04*
f2	5.89E+03	*2.99E+02*	7.30E+03	*2.05E+02*	6.71E+03	*1.41E+02*
f3	1.98E+01	*3.30E-02*	1.94E+01	*4.77E-02*	1.93E+01	*2.95E-02*
f4	6.85E+11	*3.23E+11*	4.60E+12	*4.00E+12*	3.75E+11	*1.60E+11*
f5	3.56E+08	*6.34E+07*	3.69E+08	*7.08E+07*	3.99E+08	*1.08E+08*
f6	1.87E+07	*4.12E+06*	1.89E+07	*2.95E+06*	1.95E+07	*7.42E+05*
f7	5.25E+03	*4.99E+03*	6.03E+08	*8.75E+08*	5.32E+04	*3.94E+04*
f8	1.63E+08	*4.62E+08*	5.58E+07	*1.98E+08*	8.95E+06	*2.94E+07*
f9	1.61E+08	*2.44E+07*	2.33E+09	*6.16E+08*	6.85E+07	*6.42E+06*
f10	9.64E+03	*2.90E+02*	8.32E+03	*2.78E+02*	8.18E+03	*2.91E+02*
f11	2.18E+02	*2.98E-01*	2.16E+02	*8.08E-01*	2.18E+02	*6.65E-01*
f12	4.80E+04	*1.31E+04*	8.52E+05	*1.77E+05*	2.75E+04	*6.72E+03*
f13	1.28E+04	*2.81E+04*	2.77E+07	*1.13E+07*	2.44E+03	*1.86E+03*
f14	6.28E+08	*9.26E+07*	2.00E+09	*6.09E+08*	2.83E+08	*2.78E+07*
f15	1.32E+04	*6.25E+02*	9.24E+03	*4.53E+02*	9.37E+03	*7.81E+02*
f16	3.97E+02	*4.24E-01*	3.93E+02	*1.26E+00*	3.96E+02	*1.08E+00*
f17	2.73E+05	*4.19E+04*	1.10E+06	*2.27E+05*	1.26E+05	*1.78E+04*
f18	1.58E+05	*2.52E+05*	2.98E+08	*6.59E+07*	7.98E+03	*7.18E+03*
f19	1.39E+06	*2.31E+05*	1.61E+06	*2.17E+05*	7.09E+05	*9.03E+04*
f20	1.71E+04	*4.95E+04*	3.13E+08	*5.09E+07*	4.03E+03	*3.52E+02*