Neural Processing Letters (2020) 51:759–796
https://doi.org/10.1007/s11063-019-10112-x

Random Regrouping and Factorization in Cooperative Particle Swarm Optimization Based Large-Scale Neural Network Training

Cody Dennis[1] · Beatrice M. Ombuki-Berman[1] · Andries P. Engelbrecht[2]

Published online: 13 September 2019

Abstract
Previous studies have shown that factorization and random regrouping significantly improve the performance of the cooperative particle swarm optimization (CPSO) algorithm. However, few studies have examined whether this trend continues when CPSO is applied to the training of feed forward neural networks. Neural network training problems often have very high dimensionality and introduce the issue of saturation, which has been shown to significantly affect the behavior of particles in the swarm; thus it should not be assumed that these trends hold. This study identifies the benefits of random regrouping and factorization to CPSO based neural network training, and proposes a number of approaches to problem decomposition for use in neural network training. Experiments are performed on 11 problems with sizes ranging from 35 up to 32,811 weights and biases, using a number of general approaches to problem decomposition, and state of the art algorithms taken from the literature. This study found that the impact of factorization and random regrouping on solution quality and swarm behavior depends heavily on the general approach to problem decomposition. It is shown that a random problem decomposition is effective in feed forward neural network training. A random problem decomposition has the benefit of reducing the issue of problem decomposition to the tuning of a single parameter.

Keywords Feed forward neural network · Particle swarm optimization · Random regrouping · Factorization · Variable interdependence · Saturation

✉ Beatrice M. Ombuki-Berman
bombuki@brocku.ca

Cody Dennis
cd15oy@brocku.ca

Andries P. Engelbrecht
engel@sun.ac.za

1 Department of Computer Science, Brock University, St. Catharines, Canada

2 Department of Industrial Engineering and Computer Science Division, Stellenbosch University, Stellenbosch, South Africa

1 Introduction

Artificial neural networks are simple structures inspired by the mammalian brain [38]. These structures provide a powerful mathematical model, capable of identifying patterns in data. This ability has led to the widespread use of neural networks in fields such as speech recognition and computer vision [38], amongst other applications. A number of earlier works have identified particle swarm optimization (PSO), and cooperative particle swarm optimization (CPSO) as suitable algorithms for neural network training. These algorithms have been successfully applied to a number of neural network training problems [9,28,52]. Artificial neural network training is often relatively difficult due to the dimensionality of the data involved. Unfortunately, the performance of the PSO algorithm is known to suffer in high dimensional problems [31,53]. The CPSO algorithm was introduced to address this issue [53]. Since its introduction, a number of techniques have been developed to further improve the performance of CPSO [27,46,49]. Two common strategies, among others, which are used in high dimensional spaces are random regrouping [27] and factorization [45].

In early works, CPSO was shown to improve performance relative to PSO, particularly as problem dimensionality increased [53]. Later works found that repeatedly randomly decomposing the problem throughout the search further improved performance in high dimensional problems [26,27]; this was referred to as random regrouping. Similarly, it was found that allowing multiple sub-swarms to optimize the same dimensions of the problem also offered improved performance in high dimensional spaces [36,46]; this was referred to as factorization. The success of these two techniques led to their use as the basis for a wide variety of CPSO based optimization algorithms [18,45,49,60]. Large scale optimization with such algorithms is typically restricted to problems with no more than 2000 dimensions. Often works examining the use of PSO and CPSO to train neural networks restrict their experimentation to similarly sized problems [36,39,55–57]. However, neural network training problems can have much higher dimensionality. It is not uncommon for a neural network training problem to have tens of thousands of dimensions. Such an increase in dimensionality is significant because the hyper-volume of the search space increases exponentially with problem dimensionality. This means that the potential difficulty of the problem increases exponentially as well. It has yet to be determined whether random regrouping or factorization will continue to be effective in such high dimensional problem spaces.

Previous works have assumed that factorization, which has been effective in the case of general function optimization, will continue to be effective in neural network training [36,45]. However, when training neural networks PSO and CPSO must contend with the issue of saturation [57] , which is not present in the case of general function optimization. This issue significantly influences the behavior of particles in the swarm [56]. When saturation occurs, particles may experience an explosion in velocity, resulting in divergent behavior of the swarm [53,57]. As a result, a swarm typically behaves differently during neural network training than it does for other types of optimization. For this reason it should not be assumed that factorization or random regrouping will continue to be effective when CPSO is used to train neural networks. In fact, previous works have suggested that random regrouping may harm performance in neural network training, due to the high degree of interdependence among the weights of the network [38].

This paper provides five primary contributions related to the training of artificial neural networks with CPSO. First, a number of approaches to problem decomposition, based on factorization and random regrouping, are proposed for use in CPSO based neural network training. Second, these decomposition methods are used to conduct an empirical evaluation of the effect of random regrouping and factorization on solution quality and swarm behavior in neural network training. A particular focus is placed on the applicability of these techniques to large scale problems. For this evaluation experiments are performed on 11 problems with neural network sizes ranging from 35 up to 32,811 weight and biases. From these 11 problems two were scaled to different sizes, ranging from 8335 to 32,095 dimensions, such that the performance of the tested algorithms on problems of different dimensionality can be directly compared. The performance of these problem decompositions is also compared to the performance achieved by a state of the art decomposition algorithm. Third, it is shown that the effect of factorization and random regrouping on swarm performance and behavior is dependent on the overall approach to problem decomposition. In particular, the results indicate that sub-swarm size plays a larger role in performance than sub-swarm membership. Fourth, it is shown that CPSO, both with and without factorization and random regrouping, continues to be effective for very high dimensional neural network training problems. Increasing problem dimensionality from 8335 to 32,095 did not significantly influence performance when factorization and random regrouping were used together, when only factorization was used, and when neither factorization or random regrouping were used. Finally, based on the results of this study recommendations are given regarding problem decomposition for neural network training. Notably, an entirely random problem decomposition is shown to outperform a state of the art problem decomposition algorithm.

The rest of this paper is organized as follows: Sect. 2 provides background information on neural networks, PSO, CPSO, and the issues encountered in high dimensional spaces. Section 3 describes the general approaches to problem decomposition used for this study. Section 4 describes the method of mini-batch training used in this work to reduce the cost of neural network training. Section 5 describes the experiments performed in this study and the algorithms which have been tested. Section 6 presents the results of the empirical study. Section 7 provides concluding remarks and directions for future work.

2 Background

This section provides background information on the topics discussed in this work. Sections 2.1 and 2.2 provide an overview of the PSO and CPSO algorithms. Section 2.3 discusses some of the issues encountered when applying CPSO to high dimensional problems. Section 2.4 gives a brief introduction to feed forward neural networks, and Sect. 2.5 discusses the use of CPSO and PSO to train artificial neural networks.

2.1 Particle Swarm Optimization

PSO is a stochastic population based search algorithm originally introduced for the optimization of boundary constrained continuous non-linear functions. The algorithm explores possible solutions to a problem by simulating the movement of a swarm of particles in some n-dimensional space. Each position in the search space is represented by an n-dimensional vector of real numbers, $\mathbf{x} = (x_1, x_2, x_3, x_4, \ldots, x_n)$, and corresponds to a potential solution. Potential solutions are evaluated using an objective function, $f : \mathbf{x} \mapsto \mathbb{R}$. The result, $f(\mathbf{x})$, is referred to as the quality of the solution represented by $\mathbf{x}$. The goal of PSO is to minimize or maximize f. Typically a particle's initial position is a randomly chosen point within the search space [21].

At every time step, each particle's position is first evaluated and then updated using its velocity vector. A particle's velocity directs it to promising areas of the search space. The velocity of a particle is calculated using Eq. (1) [44].

$$\mathbf{v}_i(t+1) = \omega \mathbf{v}_i(t) + c_1 \mathbf{r}_1(t)(\mathbf{y}_i(t) - \mathbf{x}_i(t)) + c_2 \mathbf{r}_2(t)(\hat{\mathbf{y}}_i(t) - \mathbf{x}_i(t)) \tag{1}$$

The coefficient ω is the inertia weight, controlling the influence of a particle's previous velocity on its current velocity; $\mathbf{y}_i(t)$ is the particle's personal best, representing the best position found by particle i up to time t; $\hat{\mathbf{y}}_i(t)$ is the particle's neighborhood best, representing the best position found by any particle in i's neighborhood up to time t. The coefficients c_1 and c_2 are referred to as the cognitive and social coefficients respectively. They control the influence of a particle's personal best position and the neighborhood best position on the particle's velocity. The vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ are n-dimensional vectors whose elements are sampled uniformly from [0, 1] [33]. These vectors are resampled each time the velocity is updated, and add a stochastic element to a particle's movement. A particle's position at time $t+1$ is given by Eq. (2) [44].

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{2}$$

A particle's neighborhood consists of a subset of the particles in the swarm. Particles in the same neighborhood share information about their personal best positions so that the neighborhood can be directed toward the best personal best found thus far [22]. The set of neighborhoods within a swarm is referred to as a neighborhood topology. The way in which these neighborhoods overlap controls the flow of information throughout the swarm. *Von Neumann* is one commonly used neighborhood topology [22]. This topology arranges particles in a two dimensional grid. A particle's neighborhood consists of itself as well as the particles above, below, to the left, and to the right of it. In this grid, the leftmost edge is joined with the rightmost edge, and the upper edge is joined to the lower edge, thus each particle has exactly four neighbors [22].

If the neighborhood best position is only updated after all particles have updated their personal best positions, then the PSO is referred to as synchronous. If the neighborhood best position is updated whenever a particle finds a better personal best position, then the PSO is referred to as asynchronous [4].

2.2 Cooperative Particle Swarm Optimization

The standard PSO algorithm has been shown to suffer from the curse of dimensionality [53]. The total hyper-volume of the search space increases exponentially with problem dimensionality. In general, as the hyper-volume of the search space grows, the performance of PSO tends to degrade [31]. One issue which contributes to this is known as two steps forward, one step back [53]. This refers to situations where improvement in some dimensions overshadows degradation in others. When this occurs, overall solution quality may improve even though a particle has actually moved away from the optimum in some dimensions [53].

The CPSO algorithm was proposed to address the curse of dimensionality [53]. CPSO reduces a large problem to a number of simpler sub-problems, each with fewer dimensions than the original. Each sub-problem is then optimized by a dedicated swarm, and complete solutions are produced by combining partial solutions from each sub-swarm [53]. This divide and conquer approach introduces two new issues: First, a method is needed to determine the degree to which a partial solution is responsible for overall solution quality. Secondly, a method is needed for selecting which partial solutions will be combined to create a complete solution. CPSO addresses these issues using an n-dimensional vector known as the context vector [53]. The context vector represents a complete solution to the problem. The components of each sub-problem correspond to specific components of the context vector. When a sub-problem is evaluated, the components of the sub-problem replace the corresponding components of the context vector. This new vector can then be evaluated using the objective function. Whenever the quality of the new vector surpasses that of the context vector, the context vector can be updated. Typically, the context vector is initialized using randomly chosen partial solutions from each of the sub-swarms [53].

2.3 Issues with Cooperative Particle Swarm Optimization

Although the CPSO algorithm has been shown to improve performance for large problem spaces, the curse of dimensionality continues to be an issue. Two major contributors to this phenomenon are particle roaming and variable interdependence. Particle roaming refers to a tendency for particles to leave the search space [13]. As problem dimensionality increases, it becomes increasingly likely for roaming to occur within the first few iterations [17]. Often, velocity can become unfavorably large [56], preventing particles from returning to the feasible range. A number of techniques have been proposed to address this issue. Velocity clamping [32] is one such technique, which restricts the maximum step size of a particle [32]. Another approach involves restricting the behavior of particles through the cognitive, social, and inertia coefficients [31]. It is also possible to manipulate the behavior of the swarm through particle initialization strategies [31]. For example, it has been shown that restricting the initial positions of particles to a small sub-space within the search space can prevent roaming, and improve performance [31,61].

Previous studies have shown that the performance of CPSO tends to improve when interdependent variables are optimized together [46,53]. At any particular iteration only a small part of the search space is available to each sub-swarm. This means that multiple intermediary solutions may need to be accepted for the context vector to arrive at a high performing position. When intermediary solutions fail to improve the quality of the context vector, CPSO may fail to reach a high performing position in the search space, since the context vector will not be updated. If correlated variables are optimized within the same sub-swarm, the need to accept intermediary positions can be reduced [53].

Unfortunately, variable interdependencies are generally not known in advance, and thus can not be used to determine an appropriate problem decomposition. Two notable strategies for mitigating this issue are random regrouping [26] and factorization [46]. In a random regrouping CPSO, problem decomposition is performed randomly, and repeated throughout the search. This increases the probability that a favorable decomposition will be used for at least part of the search [26]. This strategy has been shown to be effective for a number of CPSO variants, for many different problems [18,26,27,49,60].

Many real world problems contain complex interdependencies, possibly involving all variables of the problem [46]. As a result, it may be difficult or impossible to reduce a problem to sub-problems while ensuring that interdependent variables are optimized together. A factorized CPSO allows multiple sub-swarms to optimize the same variable in an attempt to address this issue [46]. Suppose some problem, P, has variables A, B, and C, such that A and B are interdependent, and B and C are interdependent. By allowing overlap between sub-swarms this problem can be decomposed into two sub-problems. One containing A and B and the other containing B and C. This views P as a collection of sub-problems in fewer dimensions, while ensuring that interdependent variables are optimized together. Past research has shown this approach to be effective for a number of optimization algorithms in many problems [36,45,46].

2.4 Feed Forward Neural networks

A feed forward neural network (FFNN) is a mathematical model inspired by the mammalian brain [3]. This model consists of a collection of nodes organized into layers. The first layer is referred to as the input layer, the last layer is referred to as the output layer, and all other layers are referred to as hidden layers. Nodes in the network have an associated activation function. The activation function serves a similar purpose to the action potential found in biological neurons [56]. This function defines the node's behavior with respect to an input. The input to a node depends on the output of the preceding layer. A node's net input is found by multiplying the output of each node in the previous layer by a distinct connection weight and taking the sum. The node's output is found by passing the net input through the activation function.

A FFNN is simply a structure capable of representing non-linear mappings from inputs to outputs. The form of this mapping is completely defined by the weights and biases of the network [3]. It has been theoretically proven that a three layer network is capable of representing any non-linear mapping [23]. The process of adjusting the weights and biases of the network is referred to as training. This paper considers supervised training, where a set of inputs and desired outputs is known in advance. Training of the network is thus an optimization problem with the goal of minimizing the difference between the network's output, and the desired output across all known examples. When a network is successfully trained, it becomes capable of accurately approximating outputs for new data patterns not found in the set of examples [56]. This ability to generalize has led to the extensive use of neural networks in real world problems such as classification, pattern recognition, forecasting, medical, financial, and other fields [38]. Previous work has established that PSO and CPSO can be successfully used to optimize the weights within a neural network [9,28,52].

2.5 Training FFNNs with CPSO

When PSO is used to train a FFNN, the position of a particle encodes all weights and biases within the neural network [52]. This is be accomplished by simply serializing the weights and biases in the network to produce a single vector. The number of weights in a three layer FFNN is given by $W = (I + 1)H + (H + 1)K$ where I is the number of input nodes, H is the number of hidden nodes, K is the number of output neurons, assuming that bias nodes are added to the input and hidden layer [52]. Thus a particle's position is a point in some W-dimensional space. When CPSO is used to train a FFNN, a similar encoding is used. The W dimensional vector corresponds to the context vector, and each sub-swarm optimizes a sub-set of all dimensions [52].

Often the mean squared error (MSE) is used in neural network training to measure the difference between the network's actual output, and desired output. The MSE is used directly as f, the objective function, and is given by Eq. (3)

$$MSE = \frac{\sum_{p=1}^{P}\sum_{k=1}^{K}(t_{kp} - o_{kp})^2}{PK} \tag{3}$$

where K is the number of output neurons, P is the number of examples, t_{kp} is the desired output of neuron k on example p, and o_{kp} is the actual output of neuron o on example p.

Previous work has shown that both PSO and CPSO exhibit divergent behavior when used to train neural networks[55,57]. Activation function saturation has been identified as a major contributor to this problem [57]. Saturation occurs when bounded activation functions are used in the hidden layer. When the output of a bounded function approaches a boundary of the function, the gradient of the function becomes extremely shallow. When the gradient becomes shallow, the change in function output can become insignificant relative to the change in input [57]. When this occurs, the magnitude of the values in a particle's position vector grow arbitrarily, with little impact on solution quality. As a result, particles may become trapped, and solution quality may fail to improve any further [57]. A number of techniques have been proposed to control saturation, and improve PSO and CPSO performance when training neural networks [38,39,56].

First, the rectified linear function can be used as the activation function in the hidden layers [56]. This function is given by $f_{ReL}(x) = max(0, x)$. Since this function has only one boundary, the potential for saturation is decreased. Furthermore, it has been shown that this function performs comparably too other popular functions [56]. When the net input of f_{ReL} is in the range $(-\infty, 0]$, the gradient of the function is zero. When the gradient of the function is zero, it may be possible for the magnitude of a weight to become arbitrarily large without impacting solution quality; thus, other techniques for preventing divergence may be necessary.

Weight decay is a form a weight regularization which reduces the overall magnitude of weights within a neural network [39]. The weight decay term is calculated as $WD = \sum_{l=1}^{W} w_l^2$ where W is the total number of weights and biases in the network and w_l is weight l. The weight decay term is added to the objective function, thus solution quality is given by $f(\mathbf{x}) + \lambda WD$, where λ is referred to as the penalty coefficient. The penalty coefficient controls the strength of regularization [39]. It was shown in [39] that weight regularization helps to reduce saturation and to control the magnitude of the weights in CPSO based neural network training. The saturation problem is also lessened by imposing bounds on the search space [38]. When a search space is unbounded the magnitude of the net input can become arbitrarily large, contributing to saturation. Particles may also obtain an extremely large velocity. When this occurs, particles exit the boundaries of the search space [8]. Velocity clamping has been used to address this issue [56]. In this study, velocity clamping is applied after a particle's velocity is updated. A particle's new velocity is calculated according to Eq. (4)

$$v_{ij}(t+1) = \begin{cases} v_{ij}(t+1) & if |v_{ij}(t+1) < V_{max} \\ -V_{max} & if\, v_{ij}(t+1) < -V_{max} \\ V_{max} & if\, v_{ij}(t+1) > V_{max} \end{cases} \tag{4}$$

where $v_{ij}(t+1)$ is the velocity of particle i in dimension j at time $t+1$, and V_{max} is a parameter of the algorithm which controls the maximum step size of a particle in any one dimension.

3 Problem Decomposition

It has been shown that the performance of CPSO depends heavily on how a problem is decomposed [46,53]. For this reason it is necessary to examine the effect of random regrouping and factorization on multiple approaches to problem decomposition. It is worth noting that for this study a problem decomposition does not need to be optimal since in general an optimal problem decomposition is not known [46]. Furthermore, this study does not attempt

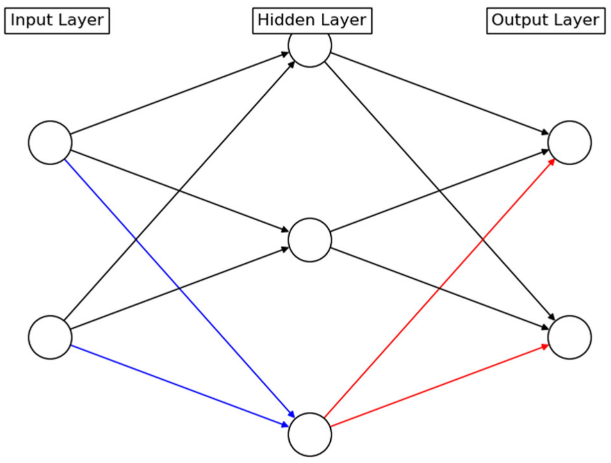

Fig. 1 An illustration of the node based decomposition. Weights in blue illustrate one sub-problem of the node based decomposition. Weights in blue and red illustrate a sub-problem of the factorized node decomposition. Black weights are not optimized in the illustrated sub-problem. (Color figure online)

to achieve optimal performance, only to examine the effect of factorization and random regrouping in a variety of different situations. This section describes the problem decomposition methods considered for this study. Section 3.1 describes node based decompositions. Section 3.2 describes layer based decompositions. Section 3.3 describes the random decompositions. Section 3.4 describes other decomposition methods which have been used recently in the literature.

3.1 Node Based Decomposition

Node based decomposition creates a sub-swarm for each node in the hidden and output layers of the network. The blue weights in Fig. 1 provide an example of one sub-problem with the node decomposition. Each sub-swarm contains all the weights, and the bias, used when calculating the net input of that node. Note that this may only be a subset of all weights on which the net input depends. For example, the net input to an output node depends on both the weights connecting the output node to the nodes in the hidden layer, and the output of the hidden nodes. The output of the hidden nodes depends on the weights connecting the hidden nodes to the input layer, thus the net input to an output node depends on these weights as well. In the past this architecture has been used to successfully optimize the weights of a neural network [52].

The factorized version of this decomposition optimizes all the weights, and the bias, on which the node's net input depends, and all weights associated with the node's output, within the same sub-swarm. The blue and red weights in Figure 1 provide an example of one sub-problem with the factorized node decomposition. For a three layer network, a sub-swarm dedicated to a node in the hidden layer includes all weights and biases connecting that node to either the input layer or the output layer. A sub-swarm dedicated to a node in the output layer contains only the weights and bias connecting that node to the hidden layer. As a result, overlap occurs only among the weights and bias connecting the hidden layer to the output layer. This factorized architecture was chosen based on the assumption that correlation is

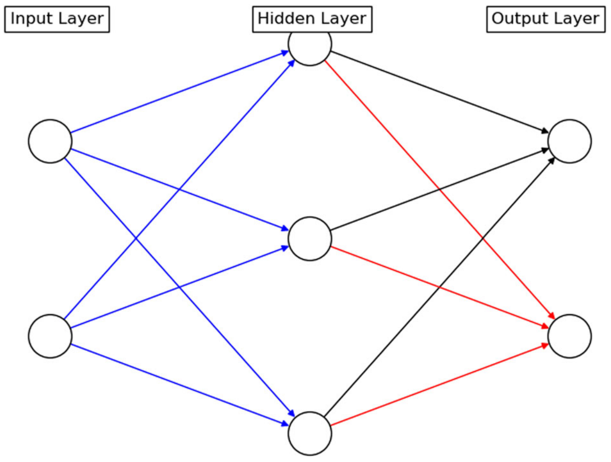

Fig. 2 An illustration of the layer based decomposition. Weights in blue illustrate one sub-problem of the layer based decomposition. Weights in blue and red illustrate a sub-problem of the factorized layer decomposition. Black weights are not optimized in the illustrated sub-problem. (Color figure online)

likely to exist between the weights going into a node, and the weights associated with a node's output.

3.2 Layer Based Decomposition

Layer based decomposition creates a sub-swarm for the hidden layer and output layer. A single sub-problem is illustrated by the blue weights in Fig. 2. Each sub-swarm contains all the weights and biases used to calculate the net input of any node in the layer. This architecture has been used previously in the literature with mixed results [52]. This decomposition method is not expected to provide optimal performance. It is used in this study to examine how differences in the number of sub-swarms and sub-swarm size may influence the effects of random regrouping and factorization. The layer decomposition produces exactly two sub-swarms of size $(I + 1)H$ and $(H + 1)K$ respectively. These two sub-swarms may or may not evenly divide the weights and biases in the network, depending on network architecture. In contrast, node based decomposition produces H sub-swarms of size $I + 1$, and K sub-swarms of size $H + 1$. This means that in general as problem size increases, the number of sub-swarms increases as well. In addition, the sub-swarms created by node decomposition tend to be smaller relative to those created by the layer decomposition.

The factorized version of this decomposition increases the number of sub-swarms, as well as their dimensionality. For a three layer network, a sub-swarm is created for each output node in the network. For an output node O, the associated sub-swarm contains all the weights and biases on which O's output depends, and thus all weights and biases on which the error at O depends. A single sub-problem is illustrated by the red and blue weights in Fig. 1. This includes all weights and biases used to calculate the net input of any hidden node, as well as all weights and the bias used to calculate O's net input. An additional sub-swarm is used to optimize all the weights and biases connecting the hidden layer to the output layer. This additional sub-swarm was included based on the assumption that interdependencies may exist between the weights within a layer.

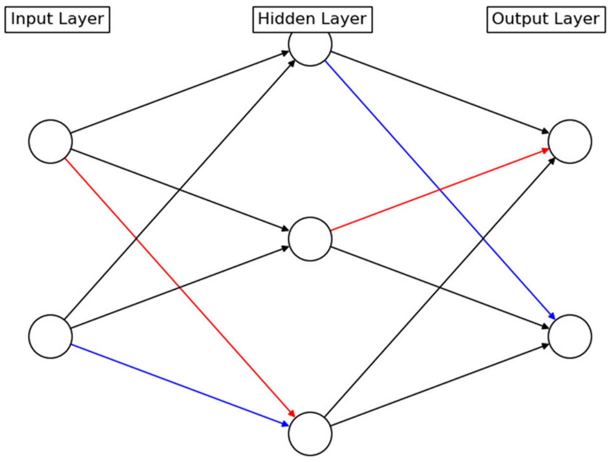

Fig. 3 An illustration of the random decomposition. Weights in blue illustrate one sub-problem of the random decomposition. Weights in blue and red illustrate a sub-problem of the factorized random decomposition. Black weights are not optimized in the illustrated sub-problem. (Color figure online)

3.3 Random Decomposition

Both node and layer decomposition implicitly assume certain properties of an appropriate problem decomposition. For example, both of these methods assume that an appropriate decomposition can be derived from the network architecture. Both node and layer decomposition also assume that the number of sub-swarms and sub-swarm sizes should be fixed for a particular architecture. Random decomposition avoids these assumptions. Previous works have shown that varying the number of sub-swarms, and sub-swarm size, can be an effective approach to problem decomposition [27,49]. To perform the random decomposition, first randomly select a number of sub-swarms. Then, randomly assign a size to each sub-swarm such that the total number of dimensions optimized is equal to the total number of weights and biases in the network. To complete the decomposition, randomly distribute the problem dimensions among the sub-swarms ensuring that each dimension is optimized at least once. An illustration of a single randomly composed sub-problem can be found in Fig. 3. When this decomposition method is used with random regrouping, this process is repeated at every iteration, otherwise the decomposition process is performed only once, at initialization. Note that by the definition above, each dimension of the problem will be optimized by exactly one sub-swarm. A factorized version of this decomposition is obtained by increasing the total number of dimensions optimized. In the factorized variant each dimension is optimized by at least one sub-swarm, and at most two sub-swarms.

3.4 Other Decompositions

Previous works have proposed and examined a method of problem decomposition for neural network training based on using all unique paths from the input layer to the output layer as sub-problems [36,37]. This approach creates a sub-swarm for each unique series of weights which connects the input layer to the output layer and is referred to as the PATHS based decomposition. For a 3-layered neural network this results in IHK sub-swarms. With this approach, the required number of sub-swarms grows exponentially with problem dimensionality. As a result, this approach to decomposition simply does not scale to very high dimensional problems. For example, for the MNIST-196 dataset used in this study and a standard CPSO, this decomposition method would require $196 * 155 * 10 = 303{,}800$ sub-swarms. With 20 particles per sub-swarm 6,076,000 function evaluations would be required to perform a single iteration. A more recent work has attempted to reduce the number of function evaluations required by this decomposition method [37]. This work dynamically samples a sub-set of all sub-swarms to train at each iteration. It was shown that similar performance to the original decomposition method could be achieved with a linear reduction in the number of function evaluations per iteration. However, this linear reduction does not offset the exponential growth in the required number of function evaluations as problem size increases.

Recently, a number of works have focused on automatically deriving an appropriate problem decomposition [6,34,35,41,47]. These works have been based on differential grouping and are primarily focused on reducing the number of function evaluations required to decompose a problem. Differential grouping attempts to identify interacting variables and to reduce a problem to a set of separable sub-problems in fewer dimensions than the original [34]. This is accomplished by holding most problem variables constant while pairs of variables are perturbed. If no interaction exists between a pair of variables, then the same change in the first variable should produce the same change in solution quality for different values of the second variable, assuming that all other variables are held constant [34]. A number of algorithms based on differential grouping have been shown to achieve near optimal decomposition and state of the art performance for many common benchmark functions [6,35,47].

4 Mini-Batch Training

When a data set is large, evaluating an objective function over all training examples can become prohibitively expensive. One solution to this issue is mini-batch training [2]. In this style of training only a sub-set of the training examples are used to evaluate the quality of a position. Each time a position is evaluated, a new mini-batch is chosen by drawing examples from a random permutation of all training examples. Once all examples have been used, a new random permutation is generated.

When MSE is used as the objective function, the quality of a position depends on the training examples which are used to calculate the MSE. Since two different mini-batches may contain different training examples, those two mini-batches may yield different MSE values for the same position. As a result, a particle's personal best position may be updated based on the performance in one mini-batch. However, this new personal best position may not perform well in all mini-batches.

To account for this, particle evaluation was modified as follows: At each iteration the particle's personal best position is re-evaluated with respect to the current mini-batch. Next, the particle's current position is evaluated with respect to the current mini-batch. If necessary, the particle's personal best position is updated. Finally, the global best position is evaluated with respect to the current mini-batch. If necessary, the global best position is updated. Once particle evaluation is complete, the position can be updated normally.

The factored evolutionary algorithms (FEA) framework [45] has been modified for use in this study to account for a similar issue. During the competition step of this framework, the algorithm iterates through a random permutation of all dimensions of the problem. For each dimension, d, the algorithm iterates through a random permutation of all sub-swarms optimizing d. For each sub-swarm the algorithm selects the particle with the best personal best position. The modified version of this algorithm selects a new mini-batch at this point and re-evaluates the current global best. The algorithm then copies the value associated with dimension d from the selected particle into the global best position, and evaluates this new vector with respect to the current mini-batch. If the global best quality improves, the global best position is updated.

During an iteration any modifications made to the global best position reflect an improvement in quality with respect to some mini-batch. However, such modifications may not improve quality with respect to all training examples. To prevent the global best position from degrading the global best position was evaluated with respect to all training examples at the end of each iteration. If the quality of the global best position, with respect to all training examples, has improved, the new global best position is kept, otherwise the global best position from the previous iteration is reinstated.

Note that this procedure does not re-evaluate a potential solution at every possible opportunity. For example, during position updates the neighborhood best is selected with no consideration given to the mini-batch on which position quality was calculated. Additional function evaluations are performed only at the points described above. The procedure described above is intended to balance performance gained from additional function evaluations with performance lost due to decreased iterations.

5 Experimental Setup

This section describes the experiments performed in this study. Section 5.1 describes the implementation of the algorithms used in this study. Section 5.2 gives the data sets used in this study, and the neural network architectures used with each problem. Section 5.3 gives the measures used to evaluated performance and behavior of the swarm.

5.1 Algorithmic Details and Control Parameters

Unless otherwise stated, the FEA framework proposed by Strasser and Fortier [45] was used for the simulations in this study, with the method of mini-batch training described in Sect. 4. The underlying optimizer is a standard CPSO as proposed by Van den Bergh and Engelbrecht [53]. For all experiments the mini-batch size used is 100. Ideally, a mini-batch should be representative of the dataset from which it is drawn. For example, for a classification problem the distribution of classes in the mini-batch should be similar to the distribution of classes across all training examples. A minibatch size of 100 was selected to balance this goal with the need to reduce the computational cost of training on large datasets. For the smaller datasets this size allows a significant portion of all training examples to be used at each iteration. For example, 66% of the Iris dataset and 33% of both the Soybean and Heart datasets. For datasets with a large number of training examples and a relatively large number of classes, such as MNIST and FMNIST, this size is large enough to permit a number of examples from each class to be present in a mini-batch.

Seven variations of the CPSO algorithm have been tested in this study. The first four variations are factorized and random regrouping (fr), only random regrouping (nfr), only factorized (fnr), and neither factorized nor random regrouping ($nfnr$). When random regrouping is used, each sub-swarm is repopulated with dimensions by drawing from a random permutation of all dimensions. In the context of neural network training, a dimension refers to a weight or bias within the network. Unless otherwise stated, the number of dimensions optimized by each sub-swarm does not change throughout the search, and each dimension of the problem is guaranteed to be optimized by at least one sub-swarm. Each of these variations has been tested with node decomposition (N), layer decomposition (L), and random decomposition (R) as described in Sect. 3. For the rest of this paper algorithms are referred to using the abbreviations above. For example, N_{nfnr} refers to the non-factored, non-regrouping variant using node decomposition, and R_{fr} refers to the factored and regrouping variant using random decomposition. The fifth variation is the CPSOS algorithm described in [53]. This simple approach to CPSO serves as a baseline and does not permit factorization or random regrouping. The sixth algorithm examined is the Historical interdependency based differential grouping (HIDG) algorithm for large scale problem decomposition [6]. This is a state of the art algorithm based on differential grouping which can be used to decompose a problem for CPSO. More information regarding differential grouping and similar algorithms can be found in Sect. 3.4. When HIDG is used for problem decomposition, the function evaluations used for decomposing the problem are counted in the total number of function evaluations performed. The final algorithm examined is the Adaptive Multi-Context Cooperatively Coevolving Particle Swarm Optimization (AMCCPSO) algorithm for large-scale problems [49]. AMCCPSO is a state of the art extension to Cooperative Coevolving PSO2, the classic random regrouping CPSO. AMCCPSO has been shown to perform competitively with a number of other state of the art algorithms for large scale optimization on a variety of problems [11,48,50]. AMCCPSO was implemented as described in [49], with the method of mini-batch training described in Sect. 4. Since AMCCPSO is a complete algorithm, and not a method of decomposition, it has not been used with the FEA framework. A description of the parameter values used for all algorithms in this study follows.

A number of techniques from the literature have been used to prevent saturation, and encourage convergent behavior. The first technique is velocity clamping. Velocity clamping has been applied in a component wise manner as described in Sect. 2.5 with $V_{max} = 0.14286$. Oldewage et al and Oldewage performed a detailed study of velocity clamping [31,32]. These studies examined the relationship between problem dimensionality and optimal V_{max} using 20 benchmark functions with dimensionality ranging from 10 to 10,000. It was shown that a relationship between dimensionality and optimal V_{max} exists, and that as dimensionality increases, optimal V_{max} tends to decrease. It was also found that as dimensionality became large, the optimal V_{max} converged. As problem size was increased beyond 1000 dimensions, further reducing V_{max} did not improve performance. The value of V_{max} used in this study is drawn from the range of optimal values identified in [31] and does not unfairly favor either large or small problems. Note that since AMCCPSO does not use velocity, velocity clamping has not been used with AMCCPSO.

The next technique is weight decay. For all experiments a weight decay term has been added to particle quality. The weight decay term used in this study is $\lambda \frac{WD}{WD+1}$, where $\lambda = 0.001$. WD has been normalized in this way to gain greater control over the strength of weight regularization. The magnitude of WD is highly problem dependent. Its magnitude is unbounded and depends on the dimensionality of the problem and the magnitude of the weights and biases in the neural network. The magnitude of the normalized weight decay term still depends on problem dimensionality and the magnitude of weights and biases of the network, however it has an upper bound of λ. This means that given two solutions, normalized weight decay can only influence which solution has higher quality if the difference in the error of the solutions is greater than $-\lambda$ and less than λ. The result is that the effect of the normalized weight decay term is limited relative to the non-normalized term. The normalized weight decay term is only significant when the error of the solutions is similar, where the threshold for similarity is bounded by λ. In a way, the non-normalized weight decay term still requires that the error of two solutions be similar, however the threshold for similarity depends primarily on the weights and biases of the network and has no upper bound. The penalty coefficient was not optimized on a per-problem basis, so the normalized weight decay term was preferable since its effects are less problem dependent.

Search space boundaries have been used to prevent the weights in the network from becoming arbitrarily large. In neural network training, the use of a bounded activation function can allow the magnitude of a particle's position to increase arbitrarily with little or no effect on solution quality [57]. If the search space is unbounded, or the boundaries of the search space are not restrictive enough, this can create large plateaus in the search space. A Particle can become trapped in these plateaus, and fail to improve its solution any further. Restricting the search space can prevent particles' position vectors from becoming arbitrarily large and help with this issue [38]. For this study the search space has been restricted to the range $[-1, 1]$. This range was taken from [38], where it was shown to reduce saturation relative to an unbounded search space.

For all problems the global best and neighborhood best positions are updated asynchronously, and a *Von Neumann* neighborhood topology was used. This topology has been shown to regularly outperform other common topologies, including *ring* and *star* [22]. Recent work suggests that the *Von Neumann* topology performs competitively in neural network training [55]. Finally, 20 particles were used in each sub-swarm, and the parameter values $c_1 = c_2 = 1.49618$ and $\omega = 0.729844$ were used in all problems. These values have been proven to produce convergent behavior [51] and have been shown to perform well empirically for a variety of problems [12,51]. As a result,these values are commonly used in the literature [1,25,30].

5.2 Problems

Table 1 lists the data sets used in this study. This table also provides the neural network architecture and overall dimensionality for each problem. For all of these problems the rectified linear function was used. Experiments were performed for 11 benchmark data sets with problem sizes ranging from 35 dimensions up to 32,811 dimensions. The citations listed in the Hidden column of Table 1 indicate the work from which the hidden layer size was taken. In all cases, bias nodes were added to the input layer and the hidden layer. All experiments were allowed to continue for 5,000,000 function evaluations. No attempt was made to stop training before overfitting occurred, thus training error provides the best indication of performance. For this study, one calculation of the MSE is considered as one function evaluation, irrespective of the number of training training examples used in the calculation. Note that, for classification problems, network output was normalized such that the sum of the output of all output nodes was 1 before MSE was calculated. To achieve this, the output of each output node was divided by the sum of the output of all output nodes.

For all problems, if missing values were present they were replaced with the column average. All input data has been normalized to the range [0, 1], except for the CNAE-9 data set. The MNIST and FMNIST data sets were scaled to a number of different sizes. This was achieved by dividing example images into an appropriate number of blocks and assigning the average of all pixels in the block to the corresponding pixel in the scaled image. Finally, for the COIL 2000 data set number of car policies was used as the target variable instead of number of mobile home policies.

Table 1 The data sets and neural network architectures used in this study

Problem	Input	Hidden	Output	Dimensionality	Instances	Source
MNIST-100	100	75	10	8335	70,000	[24]
MNIST-144	144	105	10	16,285	70,000	[24]
MNIST-169	169	135	10	24,310	70,000	[24]
MNIST-196	196	155	10	32,095	70,000	[24]
FMNIST-100	100	75	10	8335	70,000	[59]
FMNIST-144	144	105	10	16,285	70,000	[59]
FMNIST-169	169	135	10	24,310	70,000	[59]
FMNIST-196	196	155	10	32,095	70,000	[59]
COIL 2000	85	85	1	7396	9000	[54]
Crime	122	122	1	15,129	1994	[40]
CNAE-9	856	30	9	25,989	1080	[7]
SLICE	384	85	1	32,811	53,500	[16]
Soybean (large)	35	12 [43]	19	679	307	[29]
Iris	4	4 [39]	3	35	150	[14]
Heart	13	6 [5]	4	98	303	[20]
Breast cancer (BRCA)	30	25 [56]	2	827	699	[58]
Wine	13	10 [55]	3	173	178	[15]

5.3 Performance Measures

The MSE has been used to evaluate neural network performance. The MSE over a training set is referred to as E_T, and the MSE over a testing set is referred to as E_G. The reported MSE values were calculated over all examples, and averaged over 30 independent experiments. The error over the training set, and the error over the testing set have been used together as an indicator of overfitting. The indicator used is referred to as the generalization factor, $\rho_F = \frac{E_G}{E_T}$ [42]. The generalization factor is interpreted as follows: $\rho_F \leq 1$ indicates that overfitting has not occurred and $\rho > 1$ indicates that overfitting has occurred.

Swarm diversity has also been used to examine swarm behavior. The diversity measure used was the average distance around the swarm center [19] calculated using Eq. (5)

$$D = \frac{1}{|S|} \sum_{i=1}^{|S|} \sqrt{\sum_{j=1}^{W} (x_{ij} - \bar{x}_j)^2} \tag{5}$$

where $|S|$ is the total number of particles, W is the dimensionality of the problem, x_{ij} is the position of particle i in dimension j, and $\bar{x}_j$ is the position of the swarm center in dimension j. This formula requires a complete solution for each particle. This solution was generated using the context vector in the same manner that complete solutions were generated for the objective function. This measure of diversity was shown to be valid in [19].

6 Results and Discussion

Table 2 presents the 30 run average of E_T for each tested algorithm. Table 2 also includes the p values of Friedman tests for difference in training performance between CPSO variants using the same decomposition method. The p values found in Table 2 indicate a significant differ-

Table 2 Average E_T of each algorithm over 30 runs for each decomposition method

	CPSOS	N_{nfnr}	N_{nfr}	N_{fnr}	N_{fr}
BRCA-30	0.025(0.009)	0.01(0.0)	0.01(0.002)	0.01(0.001)	**0.01(0.001)**
CNAE-856	0.029(0.005)	0.008(0.002)	**0.008(0.001)**	0.01(0.001)	0.009(0.002)
CRIME-122	0.078(0.019)	0.024(0.002)	0.022(0.002)	0.023(0.002)	**0.022(0.002)**
HEART-13	0.12(0.012)	0.097(0.009)	0.101(0.005)	**0.097(0.009)**	0.1(0.007)
IRIS-4	0.028(0.041)	0.015(0.027)	0.012(0.027)	**0.006(0.003)**	0.008(0.006)
COIL-85	0.204(0.153)	0.113(0.024)	0.101(0.019)	**0.092(0.01)**	0.108(0.024)
SLICE-384	70.952(5.548)	38.763(1.456)	39.564(1.163)	**37.292(1.448)**	40.182(1.401)
SOY-35	0.009(0.002)	**0.002(0.001)**	0.002(0.001)	0.003(0.001)	0.003(0.002)
WINE-13	0.01(0.036)	0.0(0.0)	0.005(0.026)	**0.0(0.0)**	0.0(0.001)

Friedman test p value: 0.0008778

	CPSOS	L_{nfnr}	L_{nfr}	L_{fnr}	L_{fr}
BRCA-30	0.025(0.009)	**0.01(0.001)**	0.01(0.002)	0.011(0.002)	0.011(0.002)
CNAE-856	**0.029(0.005)**	0.044(0.021)	0.04(0.02)	0.041(0.008)	0.037(0.01)
CRIME-122	0.078(0.019)	0.043(0.014)	**0.039(0.016)**	0.046(0.014)	0.042(0.019)
HEART-13	0.12(0.012)	0.102(0.011)	0.101(0.008)	**0.101(0.006)**	0.102(0.007)
IRIS-4	0.028(0.041)	0.011(0.027)	**0.007(0.004)**	0.007(0.004)	0.009(0.004)
COIL-85	0.204(0.153)	**0.105(0.02)**	0.107(0.024)	0.119(0.028)	0.125(0.034)
SLICE-384	**70.952 (5.548)**	100.157(112.855)	100.288(95.271)	410.582(185.526)	206.001(142.484)
SOY-35	**0.009(0.002)**	0.013(0.003)	0.014(0.003)	0.015(0.003)	0.014(0.003)
WINE-13	0.01(0.036)	0.012(0.045)	0.01(0.036)	**0.001(0.003)**	0.005(0.026)

Friedman test p value: 0.569

Table 2 continued

	CPSOS	R_{nfnr}	R_{nfr}	R_{fnr}	R_{fr}
BRCA-30	0.025(0.009)	0.01(0.002)	0.01(0.001)	**0.01(0.001)**	0.01(0.0)
CNAE-856	0.029(0.005)	0.033(0.03)	**0.013(0.003)**	0.021(0.008)	0.016(0.003)
CRIME-122	0.078(0.019)	0.026(0.005)	**0.023(0.002)**	0.024(0.003)	0.024(0.002)
HEART-13	0.12(0.012)	0.102(0.009)	0.104(0.006)	**0.1(0.008)**	0.102(0.005)
IRIS-4	0.028(0.041)	0.017(0.033)	0.016(0.031)	**0.006(0.003)**	0.008(0.004)
COIL-85	0.204(0.153)	**0.097(0.02)**	0.106(0.023)	0.102(0.015)	0.126(0.028)
SLICE-384	70.952(5.548)	80.45(21.988)	**50.599(3.75)**	67.223(16.373)	51.336(5.308)
SOY-35	0.009(0.002)	0.01(0.005)	0.008(0.002)	0.009(0.004)	**0.007(0.002)**
WINE-13	0.01(0.036)	0.01(0.036)	0.011(0.042)	0.0(0.001)	**0.0(0.0)**
Friedman test p value: 0.002288					

Bold values indicate the minimum value of a row. Standard deviation is listed in parentheses

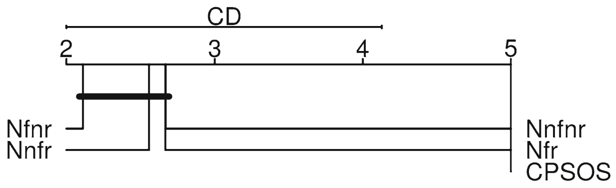

Fig. 4 A critical difference plot [10] of E_T when the node based decomposition is used. A horizontal bar across algorithms indicates that no significant difference was detected between those algorithms

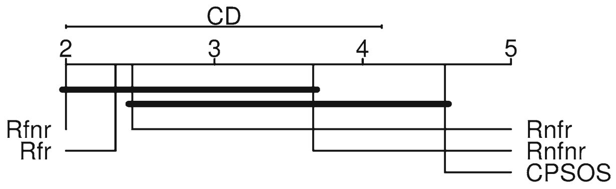

Fig. 5 A critical difference plot [10] of E_T when the random decomposition is used. A horizontal bar across algorithms indicates that no significant difference was detected between those algorithms

ence in training performance between CPSO variants when node decomposition and random decomposition were used. Figure 4 shows that all four algorithms significantly outperformed CPSOS on training data when node based decomposition was used. Figure 5 shows that R_{fnr} and R_{fr} significantly outperformed CPSOS on training data. For both the random decomposition and the node based decomposition no significant difference exists between the $nfnr$, nfr, fnr and fr variants. Table 3 presents the 30 run average E_G for each algorithm, as well as the p values of the Friedman tests for difference in generalization performance between CPSO variants using the same decomposition method. The p values found in Table 3 do not indicate a significant difference in generalization performance between CPSO variants for any decomposition method.

The CPSOS algorithm has been shown to suffer from stagnation [53]. Furthermore, the number of function evaluations per iteration is dependent on the number of sub-swarms. If CPSOS is used as the underlying optimizer for a problem of d dimensions, then $2(d) + (3 \times 20)(d)$ function evaluations are required at each iteration, since CPSOS optimizes each dimension of the problem independently. For a problem where $d = 30{,}000$, only two iterations could complete before the maximum number of function evaluations is reached. Both of these issues likely contributed to the poor performance of CPSOS. Despite these issues no significant difference was detected between CPSOS, L_{nfnr}, L_{nfr}, L_{fnr}, or L_{fr} in either training or generalization performance.

Table 4 presents the 30 run average of ρ_F for each tested algorithm. Table 4 shows that overfitting occurred in a number of experiments. Experiments on the Wine and Iris data sets in particular produced extremely large values of ρ_F. These large values for ρ_F occurred when an algorithm was able to completely memorize a training set for a least 1 run, resulting in a training error of 0. When calculating ρ_F for a run, an E_T of 0 was replaced with an arbitrary small value to avoid division by zero. This resulted in the extremely large values of ρ_F for Iris and Wine. Table 4 also includes the results of Friedman tests for difference in ρ_F between CPSO variants using the same decomposition method. The p values presented in Table 4 indicate a significant difference in ρ_F between CPSO variants when the node and random decompositions were used. This indicates a significant difference in overfitting behavior between the tested CPSO variants. Subsequent critical difference plots indicate

Table 3 Average E_G of each algorithm over 30 runs for each decomposition method

	CPSOS	N_{nfnr}	N_{nfr}	N_{fnr}	N_{fr}
BRCA-30	0.067(0.032)	**0.035(0.012)**	0.036(0.013)	0.036(0.011)	0.04(0.012)
CNAE-856	0.04(0.007)	0.027(0.003)	**0.026(0.002)**	0.027(0.003)	0.027(0.003)
CRIME-122	0.085(0.022)	0.033(0.003)	**0.027(0.004)**	0.029(0.003)	0.027(0.004)
HEART-13	**0.19(0.029)**	0.197(0.025)	0.199(0.02)	0.193(0.029)	0.201(0.027)
IRIS-4	0.041(0.06)	0.023(0.041)	0.028(0.042)	0.019(0.024)	**0.015(0.015)**
COIL-85	0.212(0.148)	0.134(0.028)	0.118(0.02)	**0.109(0.011)**	0.125(0.025)
SLICE-384	70.623(5.651)	38.162(1.453)	38.897(1.196)	**36.791(1.561)**	39.635(1.448)
SOY-35	**0.072(0.002)**	0.075(0.001)	0.075(0.002)	0.075(0.001)	0.074(0.002)
WINE-13	0.043(0.034)	0.036(0.014)	0.043(0.03)	0.044(0.015)	**0.029(0.014)**
Friedman test p value: 0.4433					
	CPSOS	L_{nfnr}	L_{nfr}	L_{fnr}	L_{fr}
BRCA-30	0.067(0.032)	**0.036(0.014)**	0.039(0.017)	0.037(0.011)	0.038(0.017)
CNAE-856	**0.04(0.007)**	0.056(0.017)	0.053(0.016)	0.052(0.008)	0.048(0.009)
CRIME-122	0.085(0.022)	0.055(0.017)	**0.049(0.019)**	0.055(0.019)	0.055(0.021)
HEART-13	0.19(0.029)	**0.183(0.024)**	0.2(0.022)	0.189(0.026)	0.197(0.026)
IRIS-4	0.041(0.06)	0.025(0.041)	0.022(0.018)	0.023(0.024)	**0.019(0.017)**
COIL-85	0.212(0.148)	**0.122(0.023)**	0.124(0.026)	0.136(0.03)	0.142(0.037)
SLICE-384	**70.623**	99.623	99.518	411.147	206.713
	(5.651)	(112.84)	(94.716)	(185.888)	(143.352)
SOY-35	0.072(0.002)	0.07(0.002)	0.07(0.003)	**0.069(0.003)**	0.07(0.003)
WINE-13	0.043(0.034)	0.052(0.052)	0.047(0.036)	0.038(0.013)	**0.036(0.03)**
Friedman test p value: 0.6626					

Table 3 continued

	CPSOS	R_{nfnr}	R_{nfr}	R_{fnr}	R_{fr}
BRCA-30	0.067(0.032)	0.039(0.007)	0.035(0.008)	0.04(0.014)	**0.034(0.012)**
CNAE-856	0.04(0.007)	0.047(0.024)	**0.029(0.004)**	0.036(0.008)	0.03(0.004)
CRIME-122	0.085(0.022)	0.036(0.008)	0.033(0.004)	**0.03(0.004)**	0.031(0.003)
HEART-13	0.19(0.029)	0.2(0.029)	**0.19(0.024)**	0.198(0.027)	0.195(0.02)
IRIS-4	0.041(0.06)	0.034(0.05)	0.027(0.043)	**0.012(0.016)**	0.013(0.015)
COIL-85	0.212(0.148)	**0.114(0.022)**	0.125(0.028)	0.118(0.017)	0.141(0.028)
SLICE-384	70.623(5.651)	79.655(21.549)	**50.164(3.899)**	66.797(16.38)	50.886(5.338)
SOY-35	0.072(0.002)	**0.071(0.003)**	0.074(0.003)	0.072(0.003)	0.072(0.003)
WINE-13	0.043(0.034)	0.048(0.04)	0.048(0.05)	**0.036(0.016)**	0.038(0.014)

Friedman test p value: 0.16

Bold values indicate the minimum value of a row. Standard deviation is listed in parentheses

Table 4 Average ρ_F of each algorithm over 30 runs for each decomposition method

	CPSOS	N_{nfnr}	N_{nfr}	N_{fnr}	N_{fr}
BRCA-30	2.749(1.033)	3.518(1.26)	3.609(1.153)	3.549(1.196)	4.069(1.371)
CNAE-856	1.404(0.143)	3.295(0.769)	3.127(0.486)	2.626(0.332)	3.088(0.563)
CRIME-122	1.08(0.08)	1.382(0.096)	1.248(0.088)	1.307(0.076)	1.261(0.08)
HEART-13	1.602(0.285)	2.047(0.337)	1.982(0.241)	2.017(0.392)	2.017(0.309)
IRIS-4	2.286	45.187	4.796	2,469,139.132	4,938,274.254
	(4.214)	(235.686)	(6.466)	(13,524,013.137)	(27,048,027.031)
COIL-85	1.07(0.061)	1.19(0.052)	1.169(0.05)	1.178(0.048)	1.166(0.051)
SLICE-384	0.995(0.009)	0.985(0.008)	0.983(0.007)	0.986(0.009)	0.986(0.008)
SOY-35	8.642(1.947)	44.04(14.297)	36.106(17.393)	31.401(6.664)	25.167(9.731)
WINE-13	129,171,994	247,208,534	257,143,313	938,090,409	244,608,626
	(174,834,650)	(284,607,500)	(561,253,036)	(2,687,208,906)	(286,759,823)
Friedman test p value: 0.01014					
	CPSOS	L_{nfnr}	L_{nfr}	L_{fnr}	L_{fr}
BRCA-30	2.749(1.033)	3.708(1.455)	3.923(1.998)	3.358(1.082)	3.632(1.612)
CNAE-856	1.404(0.143)	1.375(0.28)	1.431(0.284)	1.284(0.159)	1.323(0.158)
CRIME-122	1.08(0.08)	1.29(0.119)	1.279(0.102)	1.208(0.139)	1.328(0.138)
HEART-13	1.602(0.285)	1.816(0.316)	1.998(0.318)	1.881(0.299)	1.95(0.294)
IRIS-4	2.286	46.094	11,932,898.997	34,529,105.69	2.421
	(4.214)	(235.031)	(53,104,384.876)	(189,123,494.014)	(2.574)
COIL-85	1.07(0.061)	1.166(0.036)	1.165(0.042)	1.143(0.043)	1.147(0.043)
SLICE-384	0.995(0.009)	0.993(0.011)	0.992(0.009)	1.0(0.01)	1.0(0.011)
SOY-35	8.642(1.947)	5.77(1.375)	5.311(1.594)	4.566(0.749)	5.356(1.094)
WINE-13	129,171,994	169,790,830	89,757,028	129,561,702	295,007,640
	(174,834,650)	(198,798,649)	(159,639,593)	(207,246,587)	(666,668,390)
Friedman test p value: 0.2164					

Table 4 continued

	CPSOS	R_{nfnr}	R_{nfr}	R_{fnr}	R_{fr}
BRCA-30	2.749(1.033)	3.885(0.983)	3.523(0.918)	4.144(1.584)	3.404(1.23)
CNAE-856	1.404(0.143)	1.981(0.755)	2.3(0.347)	1.822(0.325)	1.897(0.223)
CRIME-122	1.08(0.08)	1.391(0.103)	1.403(0.105)	1.274(0.117)	1.274(0.095)
HEART-13	1.602(0.285)	1.976(0.333)	1.843(0.316)	1.991(0.319)	1.913(0.241)
IRIS-4	2.286	2.9	306.693	19,992,708.059	1.996
	(4.214)	(3.254)	(1661.719)	(109,502,026.68)	(2.739)
COIL-85	1.07(0.061)	1.186(0.049)	1.182(0.046)	1.158(0.041)	1.126(0.046)
SLICE-384	0.995(0.009)	0.991(0.011)	0.991(0.008)	0.993(0.009)	0.991(0.009)
SOY-35	8.642(1.947)	9.917(9.652)	9.686(3.047)	10.692(6.844)	11.777(3.991)
WINE-13	129,171,994	148,941,753	137,099,398	49,785,055,742	229,422,815
	(174,834,650)	(337,966,648)	(208,228,281)	(271,754,659,409)	(207,162,996)
Friedman test p value: 0.01328					

Standard deviation is listed in parentheses

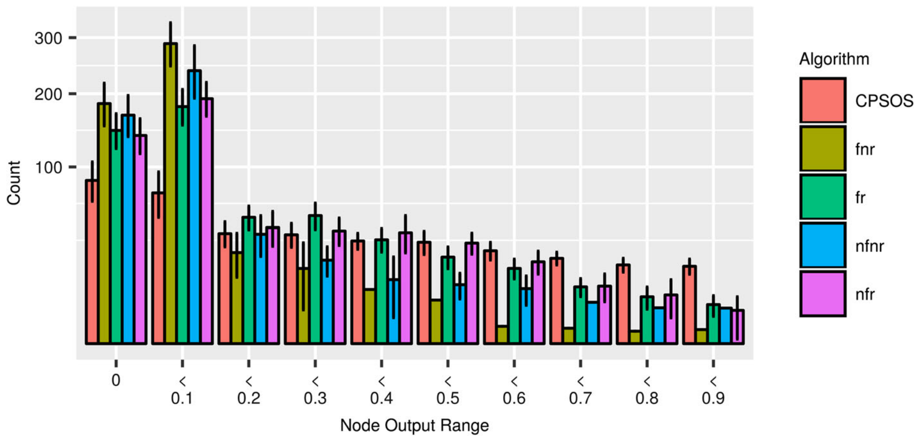

Fig. 6 A bar plot indicating the frequency with which values were output by nodes in the hidden layer on the Wine data set. The error bars indicate a 95% CI. (Color figure online)

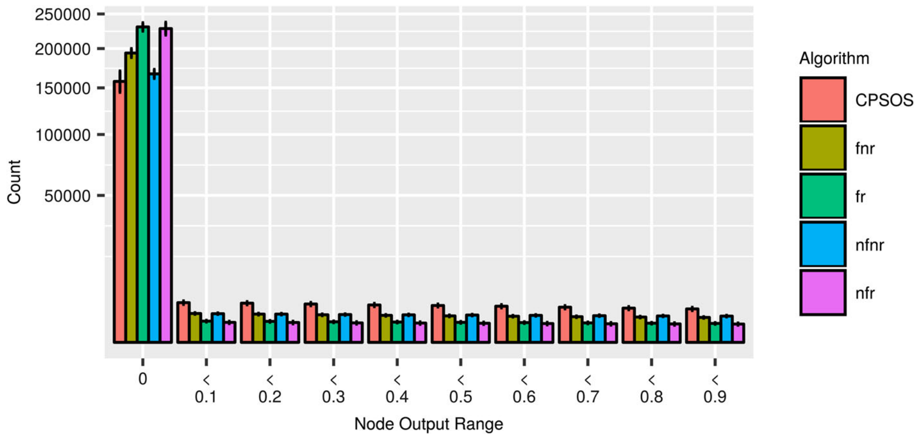

Fig. 7 A bar plot indicating the frequency with which values were output by nodes in the hidden layer on the COIL 2000 data set. The error bars indicate a 95% CI. (Color figure online)

that CPSOS exhibited significantly lower ρ_F than N_{fnr}, N_{nfnr}, and R_{fnr}. The resistance of CPSOS to overfitting is believed to be a result of the relatively low number of iterations performed by CPSOS. For example, on the Iris data set, CPSOS was able to complete 2304 iterations, whereas the N_{nfnr} was able to complete 6849 iterations.

Figures 6 and 7 present bar graphs describing the distribution of outputs in the hidden layer for the Wine and COIL 2000 data sets when the node based decomposition was used. The leftmost group of bars in these plots describe the frequency with which 0 was output by a node in the hidden layer. Since the rectified linear activation function has been used, an output of 0 corresponds to a gradient of 0 and frequent 0 outputs indicate saturation of the node. From these figures it can be seen that all five variants exhibited saturation, and that CPSOS produced low levels of saturation relative to the node based variants. This occurred consistently across all data sets, with the exception of SLICE where CPSOS and the node based variants produced a similar number of 0 outputs in the hidden layer. Figures 8 and

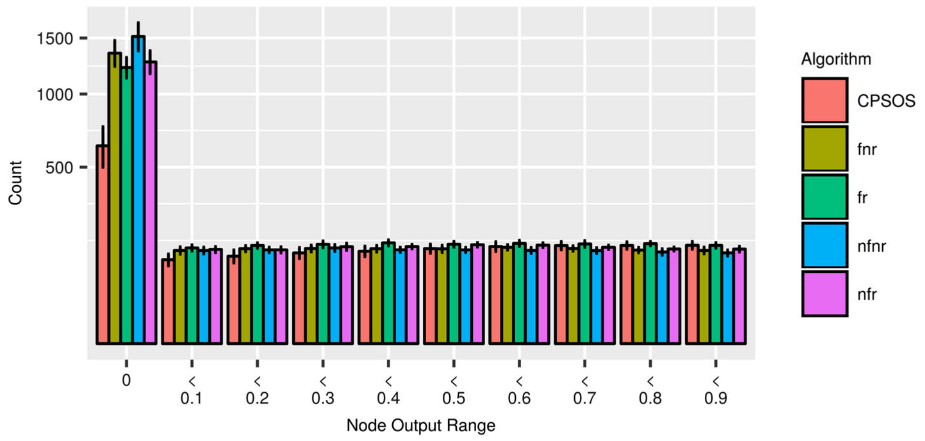

Fig. 8 A bar plot indicating the frequency with which values were output by nodes in the hidden layer on the Soybean data set. The error bars indicate a 95% CI. (Color figure online)

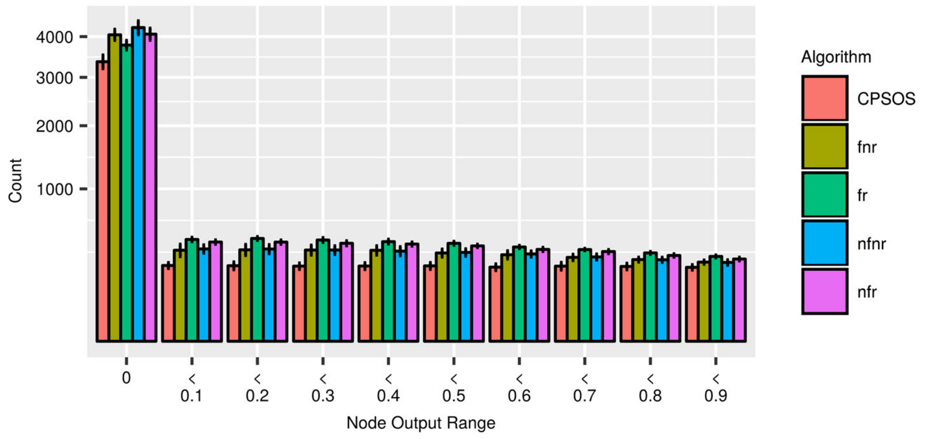

Fig. 9 A bar plot indicating the frequency with which values were output by nodes in the hidden layer on the CNAE-9 data set. The error bars indicate a 95% CI. (Color figure online)

9 illustrate the distribution of outputs in the hidden layer for the Soybean and CNAE-9 data sets when the random decomposition was used. From Figs. 8 and 9 it can be seen that CPSOS produced low levels of saturation relative to the random variants. This trend was again consistent across all data sets except SLICE.

The CPSOS algorithm was found to consistently produce low levels of saturation in the hidden layer. Although a number of previous works have suggested a link between reduced saturation of the hidden layer and improved performance [38,39,57], this effect was not observed is this study. The CPSOS algorithm performed consistently poorly on training data and did not perform significantly better than any other on test data. Furthermore, CPSOS produced the worst testing performance for a number of data sets including BRCA, Crime, Iris, and COIL 2000. While CPSOS performed poorly, it also exhibited resistance to overfitting. This may suggest a link between overfitting behavior and saturation.

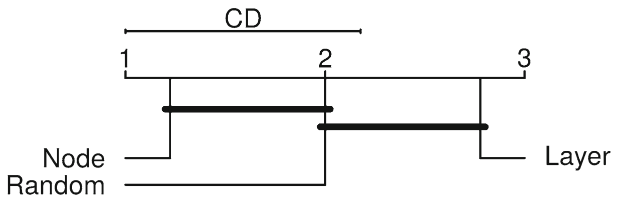

Fig. 10 A critical difference plot [10] comparing E_T of fr with each decomposition method. A horizontal bar across algorithms indicates that no significant difference was detected between those algorithms

Additional Friedman tests were performed to determine if the general approach to decomposition influences the performance of factorization or random regrouping. A separate test was performed for each variant, $nfnr$, nfr, fnr, and fr, comparing performance with the node, layer, and random decomposition methods. When E_T of the $nfnr$ variants was tested, a p value of 0.1211 was produced. Tests of E_T for the nfr, fnr, and fr algorithms yielded p values of 0.007742, 0.0003002, and 0.00432 respectively. Similar tests of E_G for $nfnr$, nfr, fnr, and fr produced p values of 0.2636, 0.3679, 0.3679, and 0.2636 respectively, indicating no significant difference in E_G between decomposition methods. Similarly, no significant difference was detected in ρ_F between decomposition methods. The difference in E_T suggests that both factorization and random regrouping are sensitive to the general approach to decomposition used. Figure 10 shows that, when factorization and random regrouping were used, node decomposition significantly outperforms layer decomposition on training data. When either factorization or random regrouping were used, similar results were obtained. It is likely that the difference in performance between these two decomposition methods was partially due to the difference in sub-swarm dimensionality.

Figure 10 also shows that there was no significant difference in training performance between R_{fr} and its node and layer based counterparts. Similar results were obtained for R_{fnr} and R_{nfr}. This may suggest that a random problem decomposition is viable in CPSO based neural network training when combined with either factorization, regrouping, or both. Similar approaches to problem decomposition were shown to be successful for many optimization problems [26,27,49]. The benefit to this type of approach is that it simplifies the issue of problem decomposition. Experimentally determining an appropriate problem decomposition is computationally expensive, with cost increasing with problem size. If a random decomposition is viable, then this challenge can be avoided entirely. Alternatively, it could be reduced to tuning a single parameter such as maximum sub-swarm size, along with the standard PSO parameters.

The images in the MNIST and FMNIST data sets were scaled to four different sizes producing problems with 8335, 16,285, 24,310, and 32,095 dimensions. This scaling makes it possible to directly compare the performance obtained on problems of different dimensionality. By comparing the performance obtained on problems of different sizes, it becomes possible to examine how problem dimensionality may effect random regrouping and factorization.

Table 5 presents the 30 run average E_T of the $nfnr$, nfr, fnr, and fr variants for each tested problem size. Table 5 also provides the p values produced by Friedman tests for a difference in training performance between the $nfnr$, nfr, fnr, and fr variants. Table 6 presents the 30 run average E_G, and the p values produced by Friedman tests for a difference in performance between the $nfnr$, nfr, fnr, and fr variants. Table 7 presents the 30 run average ρ_F, and the p values produced by Friedman tests for a difference in ρ_F between the $nfnr$, nfr, and fnr, and fr variants.

Table 5 Average E_T of each algorithm over 30 runs on MNIST and FMNIST in 8335, 16,285, 24,310, and 32,095 dimensions

	nfnr	nfr	fnr	fr
MNIST-100-N	**0.021(0.0)**	0.021(0.001)	0.021(0.001)	0.022(0.0)
FMNIST-100-N	**0.03(0.0)**	0.03(0.0)	0.03(0.0)	0.031(0.001)
MNIST-100-L	0.042(0.009)	**0.038(0.007)**	0.047(0.006)	0.047(0.007)
FMNIST-100-L	0.046(0.007)	**0.045(0.006)**	0.047(0.005)	0.05(0.005)
MNIST-100-R	0.032(0.007)	**0.025(0.005)**	0.029(0.005)	0.026(0.001)
FMNIST-100-R	0.039(0.006)	**0.034(0.003)**	0.036(0.003)	0.034(0.002)
Friedman test p value: 0.1577				
MNIST-144-N	**0.022(0.001)**	0.023(0.0)	0.023(0.001)	0.023(0.001)
FMNIST-144-N	**0.031(0.0)**	0.032(0.0)	0.032(0.0)	0.032(0.0)
MNIST-144-L	**0.041(0.008)**	0.043(0.009)	0.046(0.009)	0.044(0.008)
FMNIST-144-L	0.049(0.007)	0.049(0.007)	0.049(0.005)	**0.049(0.005)**
MNIST-144-R	0.034(0.009)	**0.026(0.003)**	0.031(0.006)	0.028(0.002)
FMNIST-144-R	0.043(0.008)	0.036(0.004)	0.039(0.006)	**0.036(0.001)**
Friedman test p value: 0.7055				
MNIST-169-N	**0.023(0.001)**	0.025(0.01)	0.023(0.001)	0.023(0.001)
FMNIST-169-N	**0.031(0.0)**	0.031(0.0)	0.032(0.001)	0.032(0.002)
MNIST-169-L	**0.043(0.008)**	0.044(0.009)	0.045(0.006)	0.044(0.007)
FMNIST-169-L	0.047(0.007)	0.047(0.007)	0.047(0.004)	**0.046(0.004)**
MNIST-169-R	0.037(0.013)	**0.027(0.002)**	0.032(0.005)	0.03(0.003)
FMNIST-169-R	0.044(0.01)	**0.035(0.003)**	0.039(0.005)	0.036(0.002)
Friedman test p value: 0.5319				
MNIST-196-N	**0.024(0.001)**	0.025(0.001)	0.025(0.001)	0.025(0.001)
FMNIST-196-N	**0.031(0.0)**	0.032(0.0)	0.032(0.001)	0.032(0.001)
MNIST-196-L	**0.044(0.009)**	0.046(0.01)	0.045(0.005)	0.044(0.005)
FMNIST-196-L	0.046(0.008)	0.048(0.007)	**0.046(0.005)**	0.046(0.004)
MNIST-196-R	0.038(0.01)	**0.029(0.003)**	0.036(0.007)	0.032(0.003)
FMNIST-196-R	0.047(0.011)	**0.036(0.003)**	0.039(0.005)	0.037(0.002)
Friedman test p value: 0.8013				

Bold values indicate the minimum value of a row. Standard deviation is listed in parentheses

The obtained p values did not indicate a significant difference in E_T, E_G, or ρ_F between $nfnr$, nfr, fnr, and fr within any tested problem size. Additional Friedman tests were performed to determine if the performance of any of these four algorithms differs between problem sizes. One Friedman test was performed for each of $nfnr$, nfr, fnr, and fr. In these Friedman tests there are four columns and six rows. Each column contains one problem size, i.e. 8335, 16,285, 24,310, or 32,095 dimensions. The six rows correspond to all possible combinations of either MNIST or FMNIST, and one particular problem decomposition. For example, one row corresponds to performance on MNIST with node decomposition, another row corresponds to performance on FMNIST with random decomposition.

The tests of both E_T and E_G for the nfr variants produced a p value of 0.008887, indicating that a significant difference in performance exists. Figures 11 and 12 show that both E_T and E_G of the nfr variants was significantly lower on problems with 8335 dimensions than

Table 6 Average E_G of each algorithm over 30 runs on MNIST and FMNIST in 8335, 16,285, 24,310, and 32,095 dimensions

	nfnr	nfr	fnr	fr
MNIST-100-N	**0.019(0.001)**	0.02(0.001)	0.02(0.001)	0.021(0.001)
FMNIST-100-N	**0.031(0.0)**	0.032(0.0)	0.032(0.001)	0.032(0.001)
MNIST-100-L	0.041(0.009)	**0.037(0.007)**	0.046(0.007)	0.046(0.007)
FMNIST-100-L	0.047(0.007)	**0.046(0.006)**	0.048(0.005)	0.051(0.005)
MNIST-100-R	0.031(0.008)	**0.024(0.005)**	0.027(0.005)	0.025(0.001)
FMNIST-100-R	0.04(0.006)	**0.035(0.003)**	0.037(0.003)	0.036(0.002)
Friedman test p value: 0.1577				
MNIST-144-N	**0.021(0.001)**	0.022(0.001)	0.022(0.001)	0.022(0.001)
FMNIST-144-N	**0.032(0.0)**	0.033(0.0)	0.033(0.0)	0.034(0.001)
MNIST-144-L	**0.04(0.008)**	0.042(0.009)	0.044(0.009)	0.043(0.008)
FMNIST-144-L	0.05(0.007)	0.05(0.007)	0.05(0.005)	**0.049(0.005)**
MNIST-144-R	0.033(0.009)	**0.025(0.003)**	0.03(0.006)	0.027(0.002)
FMNIST-144-R	0.044(0.007)	0.038(0.004)	0.04(0.006)	**0.037(0.001)**
Friedman test p value: 0.4575				
MNIST-169-N	**0.021(0.001)**	0.024(0.01)	0.022(0.001)	0.022(0.001)
FMNIST-169-N	**0.032(0.001)**	0.033(0.0)	0.033(0.001)	0.033(0.001)
MNIST-169-L	**0.042(0.009)**	0.044(0.009)	0.044(0.007)	0.043(0.007)
FMNIST-169-L	0.047(0.007)	0.048(0.007)	0.048(0.004)	**0.047(0.004)**
MNIST-169-R	0.036(0.014)	**0.025(0.002)**	0.031(0.005)	0.029(0.003)
FMNIST-169-R	0.045(0.01)	**0.036(0.003)**	0.04(0.005)	0.038(0.002)
Friedman test p value: 0.5319				
MNIST-196-N	**0.023(0.001)**	0.023(0.001)	0.023(0.001)	0.023(0.001)
FMNIST-196-N	**0.032(0.0)**	0.033(0.0)	0.033(0.001)	0.033(0.001)
MNIST-196-L	0.043(0.009)	0.045(0.009)	0.044(0.005)	**0.043(0.005)**
FMNIST-196-L	0.047(0.007)	0.049(0.006)	**0.047(0.004)**	0.047(0.004)
MNIST-196-R	0.037(0.01)	**0.028(0.003)**	0.035(0.007)	0.03(0.003)
FMNIST-196-R	0.048(0.011)	**0.037(0.003)**	0.04(0.005)	0.038(0.002)
Friedman test p value: 0.9776				

Bold values indicate the minimum value of a row. Standard deviation is listed in parentheses

on problems with 32,095 dimensions. Interestingly, this was the only significant difference detected. The tests for difference in E_T between problem sizes for the $nfnr$, fnr, and fr variants yielded p values of 0.06018, 0.2407, and 0.4575 respectively. The tests for difference in E_G between problems sizes for the $nfnr$, fnr, and fr variants yielded p values of 0.0618, 0.334, and 0.5725 respectively. No significant difference in ρ_F was detected between problem sizes for the $nfnr$, fnr, nfr, or fr variants.

This suggests that random regrouping is sensitive to the dimensionality of the problem. It is worth noting that each size of MNIST and FMNIST requires a different neural network architecture. Since the network architecture was not optimized for the different sizes of MNIST and FMNIST, the difference in architecture may influence any difference, or lack thereof, in performance. The issue of different neural network architectures is only present in the Friedman tests for difference in performance between problem sizes. For all other

Table 7 Average ρ_F of each algorithm over 30 runs on MNIST and FMNIST in 8335, 16,285, 24,310, and 32,095 dimensions

	nfnr	nfr	fnr	fr
MNIST-100-N	0.934(0.014)	0.947(0.013)	0.942(0.015)	0.945(0.019)
FMNIST-100-N	1.053(0.007)	1.049(0.01)	1.049(0.006)	1.047(0.009)
MNIST-100-L	0.969(0.017)	0.969(0.017)	0.979(0.017)	0.979(0.011)
FMNIST-100-L	1.024(0.009)	1.022(0.01)	1.021(0.01)	1.019(0.009)
MNIST-100-R	0.956(0.024)	0.949(0.019)	0.951(0.014)	0.945(0.016)
FMNIST-100-R	1.034(0.011)	1.043(0.009)	1.035(0.009)	1.039(0.008)
Friedman test p value: 0.753				
MNIST-144-N	0.945(0.018)	0.946(0.019)	0.949(0.017)	0.947(0.014)
FMNIST-144-N	1.051(0.008)	1.046(0.006)	1.048(0.007)	1.044(0.008)
MNIST-144-L	0.975(0.014)	0.969(0.017)	0.972(0.017)	0.975(0.014)
FMNIST-144-L	1.022(0.01)	1.021(0.009)	1.02(0.008)	1.019(0.008)
MNIST-144-R	0.967(0.016)	0.95(0.017)	0.955(0.018)	0.955(0.014)
FMNIST-144-R	1.026(0.013)	1.038(0.009)	1.032(0.012)	1.034(0.009)
Friedman test p value: 0.6149				
MNIST-169-N	0.951(0.015)	0.954(0.02)	0.953(0.017)	0.954(0.013)
FMNIST-169-N	1.047(0.007)	1.044(0.009)	1.046(0.008)	1.046(0.008)
MNIST-169-L	0.975(0.013)	0.979(0.015)	0.976(0.016)	0.979(0.013)
FMNIST-169-L	1.022(0.011)	1.022(0.009)	1.018(0.009)	1.022(0.009)
MNIST-169-R	0.969(0.02)	0.96(0.017)	0.965(0.015)	0.963(0.019)
FMNIST-169-R	1.027(0.014)	1.039(0.01)	1.029(0.012)	1.035(0.01)
Friedman test p value: 0.7055				
MNIST-196-N	0.95(0.017)	0.951(0.018)	0.949(0.018)	0.945(0.013)
FMNIST-196-N	1.048(0.009)	1.048(0.009)	1.046(0.008)	1.048(0.008)
MNIST-196-L	0.978(0.017)	0.978(0.012)	0.977(0.016)	0.968(0.014)
FMNIST-196-L	1.026(0.011)	1.021(0.011)	1.022(0.008)	1.021(0.009)
MNIST-196-R	0.968(0.017)	0.961(0.017)	0.962(0.016)	0.96(0.017)
FMNIST-196-R	1.023(0.016)	1.033(0.011)	1.033(0.012)	1.035(0.009)
Friedman test p value: 0.1447				

Standard deviation is listed in parentheses

Friedman tests in this study, each row contains only one neural network architecture. Since each row contains only one architecture, differences due to architecture can not influence the ranking of algorithms within a row.

To further evaluate the results in this work, plots of average diversity and E_T have been generated to compare the $nfnr$, nfr, fnr, and fr variants. The way in which a problem is decomposed directly influences the positions in the search space which are accessible to individual particles. As a result, problem decomposition shapes the behavior of the swarm as a whole. For this reason, each problem decomposition was considered separately when examining the behavior of the $nfnr$, nfr, fnr, and fr variants. Furthermore, by considering each problem decomposition separately, it becomes possible to examine the difference in performance between decomposition methods.

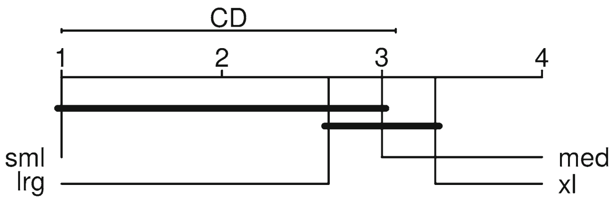

Fig. 11 A critical difference plot [10] comparing E_T of nfr for each problem size. A horizontal bar across algorithms indicates that no significant difference was detected between those algorithms

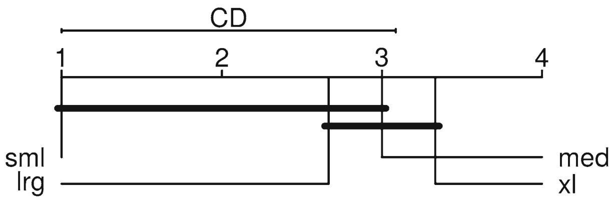

Fig. 12 A critical difference plot [10] comparing E_G of nfr for each problem size. A horizontal bar across algorithms indicates that no significant difference was detected between those algorithms

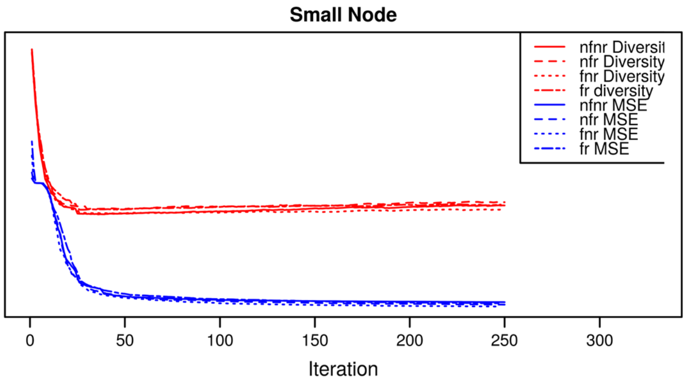

Fig. 13 A plot of average diversity and E_T at each iteration of the $nfnr$, nfr, fnr, and fr variants using the node based decomposition. E_T and diversity have been averaged across MNIST and FMNIST in 8335 dimensions and COIL 2000

When generating the line plots in this paper, diversity was normalized to the range [0, 1] according to the equation $d'_z = \frac{d_z - D_{min}}{D_{max} - D_{min}}$ where d_z is average diversity at iteration i, D_{min} is the minimum average diversity of any algorithm, and D_{max} is the maximum average diversity of any algorithm. E_T was normalized similarly. After normalization, diversity was shifted upward. This allows the ranking of each algorithm to be easily observed at each iteration with respect to either diversity or E_T. This also makes it possible to examine how diversity relates to the performance of an algorithm.

Figures 13 and 14 present average diversity and E_T of the $nfnr$, nfr, fnr, and fr variants at each iteration when node decomposition was used. Figures 13 and 14 show that problem size had little effect on swarm behavior when node decomposition was used. For the smaller problems a greater difference in swarm diversity between CPSO variants can be observed. With node decomposition, for both the factored and non-factored variants, the number of sub-swarms and the size of those sub-swarms depends on the number of nodes

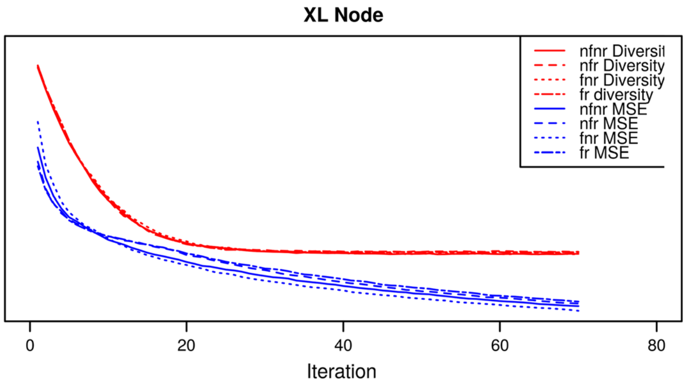

Fig. 14 A plot of average diversity and E_T at each iteration of the $nfnr$, nfr, fnr, and fr variants using the node based decomposition. E_T and diversity have been averaged across MNIST and FMNIST in 32,095 dimensions and SLICE

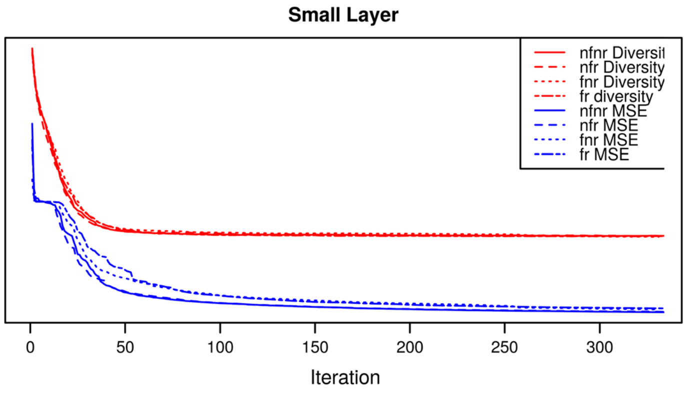

Fig. 15 A plot of average diversity and E_T at each iteration of the $nfnr$, nfr, fnr, and fr variants using the layer based decomposition. E_T and diversity have been averaged across MNIST and FMNIST in 8335 dimensions and COIL 2000

in the network. The difference in sub-swarm size between the factored and non-factored variants depends only on the number of output nodes. However, the number of output nodes is the same for MNIST and FMNIST in both 8335 and 32,095 dimensions. As a result, an increase in problem dimensionality reduces the relative difference in sub-swarm size between the factored and non-factored variants. It is likely that this contributes to the similar behavior observed in Figs. 13 and 14.

Figures 15 and 16 present average diversity and E_T of the $nfnr$, nfr, fnr, and fr variants at each iteration when layer decomposition was used. Figure 15 shows all four algorithms behaved similarly in the smaller problems. Figure 16 shows that the factored variants exhibited lower diversity than the non-factored variants early in the search. Figure 16 also shows that the factored and non-regrouping variant exhibited consistently low diversity throughout the search for larger problems. This is notable because factorization was expected to increase diversity. For layer decomposition, factorization increases both the number of sub-swarms and sub-swarm dimensionality. It has been shown that, as the number of sub-swarms increases, diversity tends to increase as well [19]. Similarly, an increase in sub-

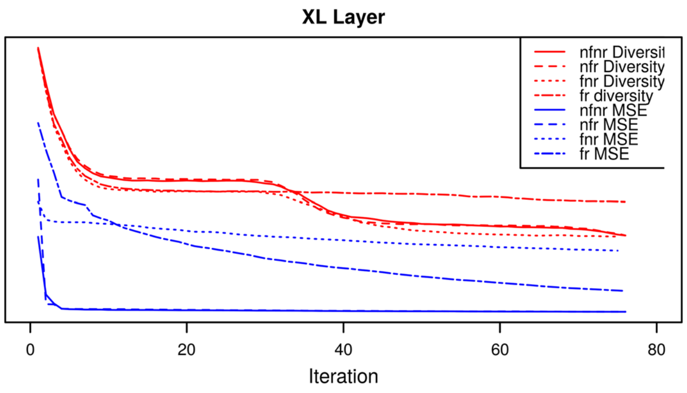

Fig. 16 A plot of average diversity and E_T at each iteration of the $nfnr$, nfr, fnr, and fr variants using the layer based decomposition. E_T and diversity have been averaged across MNIST and FMNIST in 32,095 dimensions and SLICE

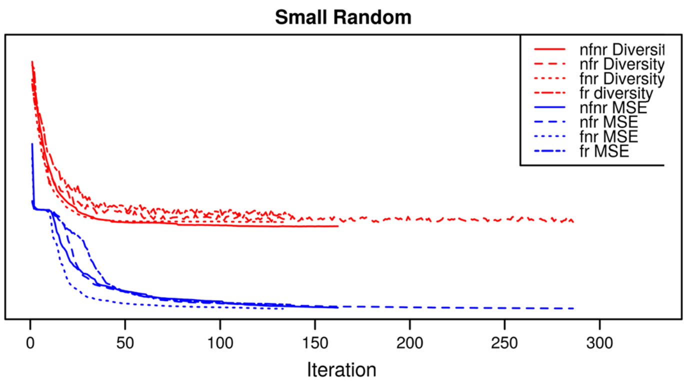

Fig. 17 A plot of average diversity and E_T at each iteration of the $nfnr$, nfr, fnr, and fr variants using the random decomposition. E_T and diversity have been averaged across MNIST and FMNIST in 8335 dimensions and COIL 2000

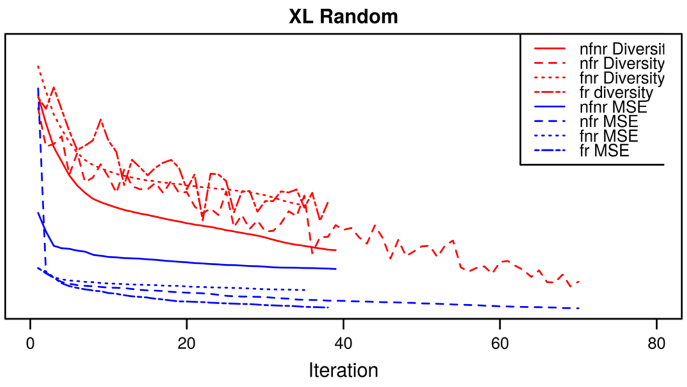

Fig. 18 A plot of average diversity and E_T at each iteration of the $nfnr$, nfr, fnr, and fr variants using the random decomposition. E_T and diversity have been averaged across MNIST and FMNIST in 32,095 dimensions and SLICE

Table 8 Training error, generalization error, ρ_F, and diversity of the decompositions produced by HIDG over all problems

	E_T	E_G	ρ_F	Diversity
BRCA-30	0.0113(0.002)	0.0382(0.0144)	3.4802(1.3499)	16.175(0.0693)
CNAE-856	0.0389(0.0079)	0.0508(0.0071)	1.3284(0.1571)	42.0595(25.6617)
CRIME-122	0.0416(0.017)	0.0542(0.0229)	1.3035(0.116)	69.2059(0.0674)
FMNIST-100	0.048(0.0062)	0.0489(0.0061)	1.0187(0.008)	45.1628(4.1341)
FMNIST-144	0.0504(0.0128)	0.0514(0.0125)	1.0207(0.0098)	59.7777(6.2909)
FMNIST-169	0.0458(0.0101)	0.0468(0.0099)	1.0225(0.0099)	73.057(6.4953)
FMNIST-196	0.0479(0.0139)	0.0489(0.0136)	1.023(0.0122)	71.7866(12.5666)
HEART-13	0.1045(0.0077)	0.1919(0.0276)	1.857(0.3573)	5.573(0.0571)
IRIS-4	0.0072(0.0032)	0.0247(0.0231)	5.8956(10.8479)	3.3145(0.0584)
MNIST-100	0.0452(0.0053)	0.0441(0.0056)	0.9746(0.0142)	30.1343(9.4617)
MNIST-144	0.0428(0.0062)	0.0418(0.0062)	0.9749(0.0155)	29.5284(10.0046)
MNIST-169	0.0483(0.0163)	0.0475(0.0166)	0.9803(0.0144)	31.4199(9.4924)
MNIST-196	0.0431(0.0116)	0.042(0.0117)	0.9722(0.0162)	5.8612(3.4884)
POLIC-85	0.1211(0.035)	0.1417(0.0383)	1.1793(0.0595)	47.0037(3.1623)
SLICE-384	702(643)	705.4012(648.0228)	1.0008(0.0073)	101.9152(0.058)
SOY-35	0.0171(0.0032)	0.0682(0.0033)	4.1295(0.8357)	14.6509(0.0596)
WINE-13	0.0264(0.0551)	0.0544(0.0476)	54,359,817(150,668,668)	7.4059(0.0571)

Standard deviation is listed in parentheses

Table 9 Training error, generalization error, ρ_F, and diversity of the decompositions produced by AMCCPSO over all problems

	E_T	E_G	ρ_F	Diversity
BRCA-30	0.0124(0.0042)	0.0458(0.0174)	4.1671(2.6406)	4.7235(3.419)
CNAE-856	0.0127(0.011)	0.0319(0.0069)	4.8904(3.3738)	5.4863(4.3037)
CRIME-122	0.0175(0.0039)	0.0209(0.0028)	1.2148(0.1312)	6.4008(3.6826)
FMNIST-100	0.0337(0.0038)	0.0351(0.0037)	1.0406(0.009)	4.9044(3.5893)
FMNIST-144	0.034(0.0037)	0.0355(0.0036)	1.0454(0.0114)	5.904(3.6997)
FMNIST-169	0.0348(0.0043)	0.036(0.0041)	1.0349(0.0128)	5.2453(4.4377)
FMNIST-196	0.0343(0.0041)	0.0357(0.0039)	1.0413(0.014)	5.623(4.0113)
HEART-13	0.1062(0.0084)	0.1855(0.0278)	1.7603(0.3175)	3.8304(1.9642)
IRIS-4	0.0078(0.0043)	0.0194(0.0219)	150.0882(804.015)	2.4313(1.1513)
MNIST-100	0.0262(0.0078)	0.025(0.0077)	0.9508(0.0151)	5.3219(3.7251)
MNIST-144	0.0281(0.0078)	0.0268(0.0075)	0.9521(0.0122)	4.6487(3.705)
MNIST-169	0.0254(0.0069)	0.0244(0.0069)	0.9574(0.017)	6.4518(4.2301)
MNIST-196	0.0284(0.0064)	0.0272(0.0062)	0.9585(0.0166)	5.0832(3.4688)
POLIC-85	0.0691(0.022)	0.0853(0.0207)	1.2475(0.0573)	5.9496(4.2963)
SLICE-384	42.2353(10.9841)	41.8125(11.1678)	0.9885(0.0096)	5.1585(3.7741)
SOY-35	0.009(0.0022)	0.0717(0.003)	8.6299(2.7213)	4.9376(3.9072)
WINE-13	0.0(0.0)	0.0322(0.0151)	120,293,922(167,563,402)	4.175(3.0122)

Standard deviation is listed in parentheses

swarm dimensionality should also produce an increase in diversity, since the chosen diversity measure is based on Euclidean distance, and each additional dimension adds a term under the square root in Euclidean distance. Although the factored and regrouping variant exhibited low diversity early in the search, it maintained diversity longer than any other algorithm and displayed consistent improvement in performance. The non-factored variants achieved better performance on average. However, they also become trapped within the first few iterations, and failed to improve further.

Figures 17 and 18 present average diversity and E_T of the $nfnr$, nfr, fnr, and fr variants at each iteration when random decomposition was used. Both figures show that the regrouping variants tended to maintain diversity longer than the $nfnr$ variant. Figure 18 shows that factorization encourages diversity as problem size grows. For random decomposition, random regrouping changes both the number of sub-swarms and the dimensionality of sub-swarms. This is likely to be the cause of the observed fluctuation in diversity across the search. In effect, when random regrouping is used with random decomposition, the swarm regularly alternates between explorative and exploitative behavior. Figure 18 also shows that, when random regrouping was used, E_T improves more consistently throughout the search. In Figure 17, the random regrouping variants delay gains in E_T, relative to the non-regrouping variants.

Finally, the best performing variants of the node, random, and layer decompositions, N_{fnr}, L_{nfr}, and R_{fnr} respectively, have been compared with the decompositions produced by HIDG, and with the AMCCPSO algorithm. The best performing variants were selected based on training performance, since no significant differences in generalization performance were detected. Table 8 presents the average E_T, E_G, ρ_F, and diversity of the decompositions produced by HIDG over 30 runs for all problems. Table 9 presents the average E_T, E_G, ρ_F, and diversity of AMCCPSO over 30 runs for all problems. Friedman tests comparing E_T, E_G, and ρ_F of N_{fnr}, L_{nfr}, R_{fnr}, and HIDG over all problems produced p values < 0.00001, < 0.00001, and 0.7314 respectively. The presented p values indicate that a significant difference in both training and generalization performance was detected between these five algorithms. Figures 19 and 20 present critical difference plots comparing the training and generalization performance of these five algorithms.

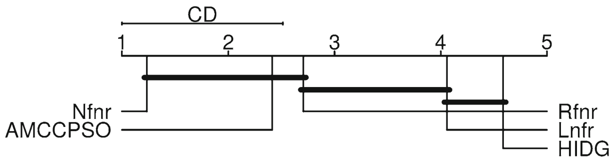

Fig. 19 A critical difference plot [10] comparing E_T of N_{fnr}, L_{nfr}, R_{fnr}, AMCCPSO and HIDG over all problems. A horizontal bar across algorithms indicates that no significant difference was detected between those algorithms

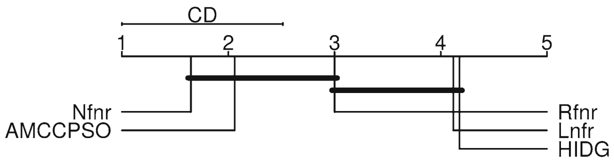

Fig. 20 A critical difference plot [10] comparing E_G of N_{fnr}, L_{nfr}, R_{fnr}, AMCCPSO and HIDG over all problems. A horizontal bar across algorithms indicates that no significant difference was detected between those algorithms

From Figure 19 it can be seen that when training error is considered N_{fnr}, AMCCPSO and R_{fnr} significantly outperformed HIDG and that N_{fnr} and AMCCPSO significantly outperformed L_{nfr}. Again the random decomposition has not performed significantly worse than any other algorithm. From Figure 20 it can be seen that when generalization error is considered N_{fnr} and AMCCPSO significantly outperformed L_{nfr} and HIDG, and R_{fnr} did not perform significantly worse than any algorithm. Figures 19 and 20 show that HIDG performed poorly when decomposing a neural network training problem. In general, HIDG had difficulty identifying separable groups. HIDG often decomposed the problem into a small number of sub-problems, usually with most variables in a single sub-problem. This produced sub-problems with thousands, or tens of thousands of variables, which sub-swarms could not effectively optimize. This is likely a result of the high degree of dependencies present between problem variables. In contrast, AMCCPSO, which uses random regrouping, performed fairly well. This is consistent with the previous results in this study since a random decomposition has been shown to be effective, and the sub-problems of AMCCPSO are reasonably small.

7 Conclusion

This study examined the use of factorization and random regrouping to address the issue of variable interdependence in CPSO based neural network training. Three methods of problem decomposition, based on factorization and random regrouping, have been proposed. These methods have been used to examine the effect of factorization and random regrouping on swarm behavior and performance for 11 benchmark problems with dimensionality ranging from 35 to 32,811.

In this work, a significant difference in E_T was detected between general decomposition methods. However, neither factorization, random regrouping, or both significantly impacted performance within a single general decomposition method. This implies that solution quality depends more on the general approach to problem decomposition than whether or not that approach incorporates factorization or random regrouping. When either factorization, random regrouping, or both were used, random decomposition did not perform significantly worse than node or layer decomposition. In addition, both R_{fr} and R_{fnr} significantly outperformed CPSOS on training data, although CPSOS exhibited significantly less overfitting than R_{fnr}. This suggests that factorization and random regrouping together can improve performance when a random problem decomposition is used.

It was found that the way in which factorization and random regrouping influence swarm behavior depends on the general approach to problem decomposition. This makes it difficult to identify desirable properties of a decomposition and exemplifies the difficulty of determining an appropriate decomposition. For these reasons, a random approach to problem decomposition may be desirable. This study has found that a random problem decomposition performs competitively in CPSO based neural network training. It should be noted that the random regrouping variants performed comparably with the non-regrouping variants for all general approaches to problem decomposition. When random regrouping is used the original problem decomposition is lost after the first iteration. For the node and layer decompositions, only sub-swarm size is preserved between regroupings. Thus the difference in performance between these two decompositions, when random regrouping is used, is likely due to sub-swarm size, and not the overall approach to decomposition. Furthermore, evidence has been presented suggesting that a random approach to problem decomposition continues to be effective as problem size increases. It was also found that the random decomposition significantly outperformed the decompositions produced by HIDG, a state of the art decomposition algorithm. CPSO variants similar to the random decomposition have been found to be effective for many optimization problems [26,27,49], and reduce the issue of problem decomposition to tuning a single parameter. AMCCPSO, a state of the art CPSO which uses random regrouping, has been shown to perform comparable with both the node and random decompositions. This suggest that some form of random decomposition may provide a general purpose CPSO for neural network training. This would significantly reduce the cost of determining an appropriate problem decomposition.

Future work will include an investigation of the effectiveness of R_{fr}, R_{fnr} and similar CPSO variants in neural network training. Such variants will be compared with a wider variety of problem decompositions on a number of suitably complex data sets. This could provide additional insights into the effects of random regrouping and factorization on neural network training, and may identify cases where a random problem decomposition is not appropriate.

References

1. Bai X, Gao X, Xue B (2018) Particle swarm optimization based two-stage feature selection in text mining. In: Proceedings of the congress on evolutionary computation. IEEE, pp 1–8
2. Baraldi A, Blonda P (1999) A survey of fuzzy clustering algorithms for pattern recognition. IEEE Trans Syst Man Cybern Part B 29(6):778–785. https://doi.org/10.1109/3477.809032
3. Bishop C (1995) Neural networks for pattern recognition. Oxford University Press, Oxford
4. Carlisle A, Dozier G (2001) An off-the-shelf pso. In: Proceedings of the workshop on particle swarm optimization, vol 1. Technology IUPUI, Indianapolis, IN, USA, pp 1–6
5. Chen A, Huang S, Hong P, Cheng C, Lin E (2011) HDPS: heart disease prediction system. In: Computing in cardiology, pp 557–560
6. Chen A, Ren Z, Yang Y, Liang Y, Pang B (2018) A historical interdependency based differential grouping algorithm for large scale global optimization. In: Proceedings of the genetic and evolutionary computation conference companion, GECCO '18. ACM, New York, NY, USA, pp 1711–1715. https://doi.org/10.1145/3205651.3208278
7. Ciarelli P, Oliveira E (2009) CNAE-9 data set. https://archive.ics.uci.edu/ml/datasets/CNAE-9. Accessed 2 Aug 2018
8. Clerc M, Kennedy J (2002) The particle swarm—explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evolut Comput 6(1):58–73. https://doi.org/10.1109/4235.985692
9. Das M, Dulger L (2009) Signature vecification (SV) toolbox: applications of PSO-NN. Eng Appl Artif Intell 22(4):688–694
10. Demsar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
11. Douglas J (2018) Efficient merging and decomposition variants of cooperative particle swarm optimization for large scale problems. Master's thesis, Brock University
12. Eberhart R, Shi Y (2000) Comparing inertia weights and constriction factors in particle swarm optimization. In: Proceedings of the congress on evolutionary computation, vol 1. IEEE, pp 84–88
13. Engelbrecht AP (2013) Roaming behavior of unconstrained particles. In: Proceedings of the Brazilian congress on computational intelligence, pp 104–111. https://doi.org/10.1109/BRICS-CCI-CBIC.2013.28
14. Fisher R (1936) Iris data set. https://archive.ics.uci.edu/ml/datasets/Iris. Accessed 2 Aug 2018
15. Forina M, et al. (1991) Wine data set. https://archive.ics.uci.edu/ml/datasets/Wine. Accessed 2 Aug 2018
16. Graf F, Kriegel H, Schubert M, Poelsterl S, Cavallaro A (2011) Relative location of ct slices on axial axis data set. https://archive.ics.uci.edu/ml/datasets/Relative+location+of+CT+slices+on+axial+axis#. Accessed 2 Aug 2018
17. Helwig S, Wanka R (2008) Theoretical analysis of initial particle swarm behavior. In: Rudolph G, Jansen T, Beume N, Lucas S, Poloni C (eds) Parallel Problem Solving from Nature—PPSN X. Springer, Berlin, pp 889–898
18. Hu C, Wu X, Wang Y, Xie F (2009) Multi-swarm particle swarm optimizer with cauchy mutation for dynamic optimization problems. In: Cai Z, Li Z, Kang Z, Liu Y (eds) Advances in Computation and Intelligence. Springer, Berlin, pp 443–453

19. Ismail A, Engelbrecht AP (2012) Measuring diversity in the cooperative particle swarm optimizer. In: Dorigo M, et al (eds) Proceedings of the international conference on swarm intelligence. Springer, Berlin, pp 97–108
20. Janosi A, Steinbrunn W, Pfisterer M, Detrano R (1989) Heart disease data set. https://archive.ics.uci.edu/ml/datasets/Heart+Disease. Accessed 2 Aug 2018
21. Kennedy J, Eberhart R (1995) Particle swarm optimization. Proc IEEE Int Conf Neural Netw 4:1942–1948. https://doi.org/10.1109/ICNN.1995.488968
22. Kennedy J, Mendes R (2002) Population structure and particle swarm performance. In: Proceedings of the international congress on evolutionary computation, vol 2. IEEE Computer Society, Washington, DC, USA, pp 1671–1676
23. Lawrence S, Tsoi A, Back A (1996) Function approximation with neural networks and local methods: bias, variance and smoothness. In: Proceedings of the australian conference on neural networks, vol 1621. Australian National University
24. LeCun Y, Cortes C, Burges J (1999) MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/. Accessed 2 Aug 2018
25. Lensen A, Xue B, Zhang M (2017) Using particle swarm optimisation and the silhouette metric to estimate the number of clusters, select features, and perform clustering. In: Proceedings of the European conference on the applications of evolutionary computation. Springer, pp 538–554
26. Li X, Yao X (2009) Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In: Proceedings of the international congress on evolutionary computation, pp 1546–1553. https://doi.org/10.1109/CEC.2009.4983126
27. Li X, Yao X (2012) Cooperatively coevolving particle swarms for large scale optimization. IEEE Trans Evolut Comput 16(2):210–224. https://doi.org/10.1109/TEVC.2011.2112662
28. Mendes R, Cortez P, Rocha M, Neves J (2002) Particle swarms for feedforward neural network training. Proc IEEE Int Conf Neural Netw 2:1895–1899. https://doi.org/10.1109/IJCNN.2002.1007808
29. Michalski R, Chilausky R (1980) Soybean (large) data set. https://archive.ics.uci.edu/ml/datasets/Soybean+%28Large%29. Accessed 2 Aug 2018
30. Mikula M, Gao X, Machová K (2017) Adapting sentiment analysis system from english to slovak. In: Proceedings of the symposium series on computational intelligence, pp 1–8. https://doi.org/10.1109/SSCI.2017.8285313
31. Oldewage E (2018) The perils of particle swarm optimization in high dimensional problem spaces. Master's thesis, University of Pretoria
32. Oldewage E, Engelbrecht AP, Cleghorn C (2017) The merits of velocity clamping particle swarm optimisation in high dimensional spaces. In: Symposium series on computational intelligence, pp 1–8. https://doi.org/10.1109/SSCI.2017.8280887
33. Oldewage E, Engelbrecht A, Cleghorn C (2018) The importance of component-wise stochasticity in particle swarm optimization. In: International conference on swarm intelligence. Springer, pp 264–276
34. Omidvar MN, Li X, Mei Y, Yao X (2014) Cooperative co-evolution with differential grouping for large scale optimization. IEEE Trans Evol Comput 18(3):378–393. https://doi.org/10.1109/TEVC.2013.2281543
35. Omidvar MN, Yang M, Mei Y, Li X, Yao X (2017) DG2: a faster and more accurate differential grouping for large-scale black-box optimization. IEEE Trans Evol Comput 21(6):929–942. https://doi.org/10.1109/TEVC.2017.2694221
36. Pillai K, Sheppard J (2011) Overlapping swarm intelligence for training artificial neural networks. In: Proceedings of the Symposium on Swarm Intelligence, pp 1–8. https://doi.org/10.1109/SIS.2011.5952566
37. Qureshi S, Sheppard JW (2016) Dynamic sampling in training artificial neural networks with overlapping swarm intelligence. In: Proceedings of the congress on evolutionary computation, pp 440–446. https://doi.org/10.1109/CEC.2016.7743827
38. Rakitianskaia A, Engelbrecht AP (2014a) Training high-dimensional neural networks with cooperative particle swarm optimiser. In: Proceedings of the international joint conference on neural networks, pp 4011–4018. https://doi.org/10.1109/IJCNN.2014.6889933
39. Rakitianskaia A, Engelbrecht AP (2014b) Weight regularisation in particle swarm optimisation neural network training. In: Proceedings of the symposium on swarm intelligence, pp 1–8. https://doi.org/10.1109/SIS.2014.7011773
40. Redmond M (2009) Communities and crime data set. https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime. Accessed 2 Aug 2018
41. Ren Z, Chen A, Wang L, Liang Y, Pang B (2017) An efficient vector-growth decomposition algorithm for cooperative coevolution in solving large scale problems. In: Proceedings of the genetic and evolutionary computation conference companion, GECCO '17, ACM, New York, NY, USA, pp 41–42. https://doi.org/10.1145/3067695.3082048

42. Röbel A (1994) The dynamic pattern selection algorithm: effective training and controlled generalization of backpropagation neural networks. Technical report, Technische Universität Berlin
43. Sexton RS, Dorsey RE (2000) Reliable classification using neural networks: a genetic algorithm and backpropagation comparison. Decis Support Syst 30(1):11–22
44. Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In: Proceedings of the international congress on evolutionary computation, pp 69–73. https://doi.org/10.1109/ICEC.1998.699146
45. Strasser S, Sheppard J, Fortier N, Goodman R (2017) Factored evolutionary algorithms. IEEE Trans Evol Comput 21(2):281–293. https://doi.org/10.1109/TEVC.2016.2601922
46. Sun L, Yoshida S, Cheng X, Liang Y (2012) A cooperative particle swarm optimizer with statistical variable interdependence learning. Inf Sci 186(1):20–39
47. Sun Y, Kirley M, Halgamuge SK (2018) A recursive decomposition method for large scale continuous optimization. IEEE Trans Evol Comput 22(5):647–661
48. Tang R, Li X (2018) Adaptive multi-context cooperatively coevolving in differential evolution. Appl Intell 48(9):2719–2729
49. Tang R, Wu Z, Fang Y (2017) Adaptive multi-context cooperatively coevolving particle swarm optimization for large-scale problems. Soft Comput 21(16):4735–4754
50. Tang R, Li X, Lai J (2018) A novel optimal energy-management strategy for a maritime hybrid energy system based on large-scale global optimization. Appl Energy 228:254–264
51. Van den Bergh F (2001) An analysis of particle swarm optimizers. PhD thesis, University of Pretoria
52. Van den Bergh F, Engelbrecht AP (2000) Cooperative learning in neural networks using particle swarm optimizers. S Afr Comput J 2000(26):84–90
53. Van den Bergh F, Engelbrecht AP (2004) A cooperative approach to particle swarm optimization. IEEE Trans Evol Comput 8(3):225–239
54. Van der Putten P, Van Someren M (eds) (2000) Insurance company benchmark (coil 2000) data set. https://archive.ics.uci.edu/ml/datasets/Insurance+Company+Benchmark+%28COIL+2000%29. Accessed 02 Aug 2018
55. Van Wyk A, Engelbrecht AP (2010) Overfitting by PSO trained feedforward neural networks. In: Proceedings of the congress on evolutionary computation, pp 1–8. https://doi.org/10.1109/CEC.2010.5586333
56. Van Wyk A, Engelbrecht AP (2016) Analysis of activation functions for particle swarm optimised feedforward neural networks. In: Proceedings of the congress on evolutionary computation, pp 423–430. https://doi.org/10.1109/CEC.2016.7743825
57. Volschenk A, Engelbrecht AP (2016) An analysis of competitive coevolutionary particle swarm optimizers to train neural network game tree evaluation functions. In: Tan Y, Shi Y, Niu B (eds) Advances in Swarm Intelligence. Springer, Cham, pp 369–380
58. Wolberg W (1990) Breast cancer Wisconsin (original) data set. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29. Accessed 2 Aug 2018
59. Xiao H, Rasul K, Vollgraf R (2017) Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. https://arxiv.org/abs/1708.07747. Accessed 2 Aug 2018
60. Xu X, Tang Y, Li J, Hua C, Guan X (2015) Dynamic multi-swarm particle swarm optimizer with cooperative learning strategy. Appl Soft Comput 29:169–183
61. Zyl E, Engelbrecht AP (2015) A subspace-based method for PSO initialization. In: Symposium series on computational intelligence, pp 226–233. https://doi.org/10.1109/SSCI.2015.42