# Simplifying Particle Swarm Optimization

**By**
**Magnus Erik Hvass Pedersen, Andrew John Chipperfield**
**School of Engineering Sciences, University of Southampton, UK**
**Published in Applied Soft Computing, 2009**
http://dx.doi.org/10.1016/j.asoc.2009.08.029

**Abstract**

The general purpose optimization method known as Particle Swarm Optimization (PSO) has received much attention in past years, with many attempts to find the variant that performs best on a wide variety of optimization problems. The focus of past research has been with making the PSO method more complex, as this is frequently believed to increase its adaptability to other optimization problems. This study takes the opposite approach and simplifies the PSO method. To compare the efficacy of the original PSO and the simplified variant here, an easy technique is presented for efficiently tuning their behavioural parameters. The technique works by employing an overlaid meta-optimizer, which is capable of simultaneously tuning parameters with regard to multiple optimization problems, whereas previous approaches to meta-optimization have tuned behavioural parameters to work well on just a single optimization problem. It is then found that the PSO method and its simplified variant not only have comparable performance for optimizing a number of Artificial Neural Network problems, but the simplified variant appears to offer a small improvement in some cases.

**Keywords:** Numerical optimization, stochastic, swarm, tuning, simplifying.

## 1 Introduction

The general purpose optimization method known as Particle Swarm Optimization (PSO) is due to Kennedy and Eberhart [1], and works by maintaining a swarm of particles that move around in the search-space influenced by the improvements discovered by the other particles. The advantage of using an optimization method such as PSO is that it does not rely explicitly on the gradient of the problem to be optimized, so the method can be readily employed for a host of optimization problems. This is especially useful when the gradient is too laborious or even impossible to derive. A good example of this is found in the history of Artificial Neural Networks (ANN), which are dataflow-models that can approximate arbitrary mappings between input and output by optimizing various parameters of the ANN. The devising of the ANN model is reported by Haykin [2, Section 1.9] to have taken several decades, apparently because each change to the ANN model also required the mathematical deriving of its

gradient, thus making it difficult for the researchers to quickly try out new ideas for the ANN model. Optimization methods such as PSO have an advantage in such prototyping, as they do not require the mathematical gradient in order to optimize a problem, but can be employed directly on the fitness measure to be optimized. This versatility comes at a price, however, as the disadvantage of general purpose optimization methods such as PSO is that they do not always work well, and often require some form of tuning to the problem at hand [3] [4].

## 1.1 Parameter Selection

Numerous studies exist on how to make the PSO method perform better. A traditional and easy way of trying to improve the PSO method is by manually changing its behavioural parameters. Various studies have been reported in the literature, for example one by Shi and Eberhart [5] [6] regarding the use of velocity boundaries for the movement of the particles and the choice of the so-called inertia weight which is believed to influence the degree of exploration versus exploitation of the PSO particles. A comprehensive survey of such studies is due to Carlisle and Dozier [7] on everything from the selection of behavioural parameters to the neighbourhood topology determining how particles influence each other, as well as other algorithmic details. However, such studies have all been limited in scope by the fact that they are either conducted manually, or at best in a systematic but very coarse way, in contrast to the automatic fine-tuning made possible by the meta-optimization approach given later in this paper.

## 1.2 Mathematical Analysis

Various attempts have been made by van den Bergh [3], Trelea [4], and Clerc and Kennedy [8], at mathematically analyzing the PSO by considering how the behavioural parameters influence contraction of its particles to a single point. But there are questions about the wider validity of these analyses. First is the preconceived notion that optimization performance and swarm contraction are so tightly connected, as it is important to stress that convergence of the optimizing particles to some point in the search-space, does not automatically guarantee the point of convergence is also the actual optimum of the problem at hand. A number of limiting assumptions are also made to facilitate the mathematical analyses in [3] [4] [8]. The particles' points of attraction are assumed to remain constant. In other words, the improvements discovered by particles have no influence on their further progress under this assumption. This also means just a single particle is studied, and not an entire swarm of particles. Stochasticity is also eliminated by using expectancies instead, thus considering an averaged and deterministic behaviour of the particle. But do these features not seem to be what makes the PSO work at all?

## 1.3 Optimizer Variants

Another approach often taken in an effort to improve the original PSO method is the development of new variants. There are two common trends: Trying to control contraction of the swarm's particles (see e.g. [9] [10]), and combining different optimization methods into a hybrid one (see e.g. [11]). Other variants of the original PSO method have been developed specifically for ANN weight optimization. For example a PSO variant that has its inertia weight decrease linearly during an optimization run is used by Eberhart et al. [12] to not only optimize ANN weights, but also other features of the ANN. Another PSO variant for optimizing ANN weights is introduced by Fan [13], and makes its particles velocity-boundaries more narrow for each iteration of the algorithm, in an attempt to focus the search more towards the end of the optimization run. Several variants of the PSO method are developed by van den Bergh [3] and tested on different kinds of ANNs. These PSO variants were developed under the notion that contraction of the swarm's particles is all-important. Although mathematical analysis was used to guide development of these PSO variants, the analysis was argued above to be ill-founded and too limited in its assumptions. Another PSO variant was introduced by Ge et al. [14] for optimizing a recurrent ANN, in which the output of some ANN nodes feedback into the nodes of previous layers, as opposed to the pure feedforward ANN studied here. The PSO variant in [14] tries to identify PSO particles that have become inactive, because they have collapsed prematurely to a local optimum. Those particles are then moved slightly in the search-space by adding Gaussian noise with zero mean and a small fixed variance. While these PSO variants do show improvements empirically they are also more complicated to describe and implement, which is probably undesirable as the original version is not well understood to begin with.

## 1.4 Optimizer Simplification & Meta-Optimization

This study takes a different approach altogether, in that it does not make any assumptions about what causes an optimization method to work well, and instead of devising more complex variants to the original PSO method, it is instead sought simplified. This is more in accord with the original paradigm of self-organization: That simple individuals cooperate and complex collective behaviour emerges. This is also in vein of Occam's Razor; *lex parsimoniae*, or the Law of Parsimony which popularly states that simplicity is usually better.

To compare the performance of optimization methods, it is important to tune their behavioural parameters for the optimization problems at hand, so as to make both optimization methods perform their best on that specific problem. One can consider the task of finding the best performing behavioural parameters as an optimization task in its own right, and solve it by using another overlaid optimization method. This is known as Meta-Optimization or Meta-Evolution. But due to the vast requirements for computational power, only few and limited studies have been conducted until recently, see for example Mercer

[15], Grefenstette [16], Bäck [17] [18], and Keane [19]. A more recent study is due to Meissner et al. [20] who attempt to tune the PSO parameters by using another instance of the PSO method as overlaid meta-optimizer as well. But their study is limited in terms of the accuracy and quality of the results obtained, as their approach lacks the proper choice of the overlaid meta-optimizer, it lacks time-saving features, and it does not have the ability to meta-optimize the PSO parameters with regard to their performance on multiple optimization problems. The latter is important because the PSO method is particularly susceptible to overtuning of its behavioural parameters, making it perform well on the problem for which the parameters were tuned, but perform poorly on problems for which the parameters were not specifically tuned. All of these issues are addressed in this study, which in some regards is an extension to our earlier work on meta-optimization [21], whose results are also compared to the work presented here.

Another approach which might be called Meta-Adaptation is sometimes taken for tuning behavioural parameters, where the parameters are being tuned by an overlaid meta-optimizer during the optimization run in an online manner. An example of meta-adaption is found in the study by Parsopoulos and Vrahatis [22, Section 10]. However, meta-adaptation is not only more complicated to implement, but also raises a number of open questions, such as how many iterations should be made during each change of the behavioural parameters, how to rate the performance incurred by this change, etc. Instead, the meta-optimization technique presented here is used for tuning the behavioural parameters in an offline manner, by only altering the behavioural parameters between the individual optimization runs of the PSO method. This is not only simpler to describe and implement, but also has the added benefit that whatever behavioural parameters are being discovered for the PSO, they can be used directly by other practitioners in their own implementation of the PSO algorithm, without having to implement a more complicated meta-adaptive scheme such as the one proposed in [22].

In short, the approach to meta-optimization taken here is generally simpler, less time-consuming, and yields better results than previous approaches in the literature. We have previously published work on meta-optimization [21] and the technique presented here extends directly on the previous work. The main difference is that meta-optimization with regard to multiple problems is now supported, in an attempt to avoid overtuning the behavioural parameters to just a single optimization problem, and thereby hopefully making the behavioural parameters generalize better to other optimization problems as well.

Using the simple and efficient technique of meta-optimization presented here, allows us to find the behavioural parameters of both the PSO method and its simplified variant. It is found that the two methods have comparable performance on a number of ANN problems, and that the simplified variant is even a slight improvement in some cases.

The paper is organized as follows. Section 2 gives a brief overview of the class of ANNs to be optimized. Section 3 describes the PSO method. Section 4 describes an optimization method especially suited for meta-optimization, and

section 5 describes how to employ it as an overlaid meta-optimizer. Section 6 describes the experimental settings and results, and conclusive remarks are found in section 7.

## 2 Artificial Neural Networks

This section briefly introduces the particular kind of ANN that is to be used in the study below, and is the actual problem the PSO method and its simplified variant must ultimately be used for optimizing. An ANN is a dataflow model inspired by the complex connections of cells in a biological brain. The type of ANN used in this study is a fully connected feedforward network with a single hidden layer having four nodes. This means the data flows from an input layer, through a single hidden layer, and finally reaches the output layer. This kind of ANN has its origin in research dating back several decades (see [2, Section 1.9] for a detailed historical account). More recent textbooks on this and other kinds of ANN can be found in Haykin [2] and Bishop [23]. The ANN has parameters, usually called weights, that can be adjusted to provide different mappings from input to output. The optimization task is then to change these weights so a desired correspondence between inputs and outputs of the ANN is achieved. Under certain circumstances, and provided the input-output mapping is well-behaved, this approximation can be achieved arbitrarily well [24] [25] [26], simply by adding more hidden nodes to an ANN having a single hidden layer whose nodes use bounded, compressive functions such as the Sigmoid function described below. However, the datasets used in the experiments here are not necessarily well-behaved as they may contain contradictory entries or have missing features needed to properly distinguish them.

To briefly describe the computation of this kind of ANN, let the $j$'th node of the $i$'th layer be denoted by $n_{ij}$. The node has a bias-value $b_{ij}$ for offsetting the influence of its inputs. The weight denoted by $w_{ijk}$ connects the $k$'th node from the previous layer to node $n_{ij}$. For an ANN having just a single hidden layer, the computation of the $j$'th element of the ANN output $\vec{o}$ can be expressed as a single, flattened formula:

$$o_j = b_{3,j} + \sum_{k=1}^{N_2} w_{3,j,k} \cdot \sigma \left( b_{2,k} + \sum_{l=1}^{N_1} w_{2,k,l} \cdot n_{1,l} \right) \tag{1}$$

Where $\vec{n}_1 \in \mathbb{R}^{N_1}$ is the first layer of the network containing its input, and $N_1$ is the number of nodes in this first layer. The number of nodes in the hidden layer is $N_2$, and the number of nodes in the output layer is $N_3$. The parameters subject to optimization are all the bias-values and weights for the entire ANN, that is, $b_{ij}$ and $w_{ijk}$ for all valid combinations of $i$, $j$, and $k$. The $\sigma(\cdot)$ part of this formula represents the calculation of the $k$'th node in the hidden layer of the ANN, where the compressive function used here is called the Sigmoidal function $\sigma : \mathbb{R} \to (0, 1)$, defined as: $\sigma(x) = 1/(1 + e^{-x})$

A fitness measure must first be defined in order to optimize the weights and bias-values of an ANN. Let $T$ be the set of input-output data the ANN must be

trained to mimic. An element in the dataset $T$ consists of input data $\vec{t_i}$ and its associated output data $\vec{t_o}$. Passing the input data $\vec{t_i}$ to the ANN and computing the actual ANN output using Eq.(1) yields $\vec{o}$. Accumulating the error between desired and actual ANN output for all data pairs in $T$ gives a measure for how well the ANN mimics a desired input-output mapping. This is known as the Mean Squared Error (MSE), defined as:

$$\text{MSE} = \frac{1}{N_3 \cdot |T|} \sum_{(\vec{t_i},\vec{t_o})\in T} \sum_{j=1}^{N_3} (o_j - t_{oj})^2 \tag{2}$$

where normalization is done with both the number of input-output pairs $|T|$, and the number of ANN output-nodes $N_3$. This is not standard in the literature, but has the advantage of giving a uniform fitness measure that can be compared independently of the given dataset and ANN sizes. The MSE measure is also sometimes halved in the literature to facilitate a cleaner derivation of the gradient, but that is ignored here. It is customary to split the dataset $T$ and use one part during optimization (or training) of the ANN, and the other part to test the generalization ability of the ANN, as a simple form of statistical cross-validation [27]. However, this is ignored here as this study is primarily concerned with the performance of PSO rather than that of the ANN.

The traditional way of optimizing ANN weights is to use a gradient-based optimization method to follow the path of steepest descent of the MSE measure. This is known as BackPropagation (BP) because it first makes a forward pass and computes the node values for the entire ANN, and then propagates the MSE errors backwards through the ANN to determine adjustments for the weights and bias-values. The BP method was developed independently by Werbos and Rumelhart et al. [28] [29] [30]. The stochastic gradient is used in this study, where a single input-output pair is picked randomly from the dataset $T$, and the gradient is computed for just that data pair. This requires much less computational time than accumulating the gradient for all data pairs in $T$, and taking small enough steps yields satisfactory results [31].

Once the ANN weights and bias-values have been optimized for the MSE measure, the ANN can be used for classification tasks as well, as proven by Richard and Lippmann [32]. This study uses 1-of-$m$ classification [2] [31] [33], where each ANN output represents a class, and the output with the highest value is chosen as the classification. The Classification Error (CLS) is computed as the number of data-pairs in $T$, for which the ANN output classification does not match that of the desired output, divided by the total number of data pairs $|T|$. This yields a CLS value in the range $[0, 1]$, where a lower CLS value means a better classification rate.

# 3 Particle Swarm Optimization

Another optimization method which can be used instead of BackPropagation for optimizing ANN weights, is the population-based method known as Particle

Swarm Optimization (PSO) due to Kennedy and Eberhart [1]. The PSO method was originally intended for simulating the social behaviour of a bird flock, but the algorithm was simplified and it was realized that the individuals – here typically called particles – were actually performing optimization.

In the PSO method the particles are initially placed at random positions in the search-space, moving in randomly defined directions. The direction of a particle is then gradually changed so it will start to move in the direction of the best previous positions of itself and its peers, searching in their vicinity and hopefully discovering even better positions with regard to some fitness measure $f : \mathbb{R}^n \to \mathbb{R}$.

Let the position of a particle be denoted by $\vec{x} \in \mathbb{R}^n$, and let $\vec{v}$ be its velocity. Both are initially chosen randomly and then iteratively updated according to two formulae. The following formula for updating the particle's velocity is by Shi and Eberhart [34]:

$$\vec{v} \leftarrow \omega\vec{v} + \phi_p r_p(\vec{p} - \vec{x}) + \phi_g r_g(\vec{g} - \vec{x}) \tag{3}$$

Where the user-defined behavioural parameter $\omega \in \mathbb{R}$ is called the inertia weight and controls the amount of recurrence in the particle's velocity. The particle's previous best position is $\vec{p}$, and $\vec{g}$ is the swarm's previous best position through which the particles communicate implicitly with each other. These are weighted by the stochastic variables $r_p$, $r_g \sim U(0, 1)$ and the user-defined behavioural parameters $\phi_p$, $\phi_g \in \mathbb{R}$. Adding the velocity to the particle's current position causes the particle to move to another position in the search-space, regardless of any improvement to its fitness:

$$\vec{x} \leftarrow \vec{x} + \vec{v} \tag{4}$$

In addition to enforcing search-space boundaries after updating a particle's position, it is also customary to impose limitations on the distance a particle can move in a single step [35]. This is done by bounding a particle's velocity $\vec{v}$ to the full dynamic range of the search-space, so the particle can at most move from one search-space boundary to the other in one step. The PSO algorithm is shown in figure 1.

## 3.1 The Simplification

To simplify the PSO method one can informally start by considering what parts of its mathematical description and algorithm can be eliminated. The velocity update formula in Eq.(3) is essentially a weighted sum of user-defined behavioural parameters and various vectors relating to the state of the swarm and its indvidual particles. Eliminating the part containing the swarm's best known position $\vec{g}$ by setting $\phi_g = 0$ would mean there is no longer any communication amongst the particles. And setting the inertia weight $\omega$ to zero would eliminate direct recurrence in Eq.(3), causing the particle to have no persistence in the path it follows.

- Initialize the particles with random positions and velocities.
- Until a termination criterion is met (e.g. a given number of fitness evaluations have been executed, observed fitness stagnates, or a fitness threshold is met), repeat:
  - For each particle with position $\vec{x}$ and velocity $\vec{v}$ do:
    - Update the particle's velocity using Eq.(3)
    - Enforce velocity boundaries.
    - Move the particle to its new position using Eq.(4)
    - Enforce search-space boundaries on the particle's position by moving it back to the boundary value if it has been exceeded.
    - If $(f(\vec{x}) < f(\vec{p}))$ then update the particle's best known position:

      $$\vec{p} \leftarrow \vec{x}$$

      And similarly for the entire swarm's best known position $\vec{g}$.

Figure 1: The PSO algorithm.

Instead, the simplification that will be tested here consists of eliminating the use of a particle's own previous best known position $\vec{p}$ by setting $\phi_p = 0$. The velocity update formula from Eq.(3) thus becomes:

$$\vec{v} \leftarrow \omega\vec{v} + \phi_g r_g(\vec{g} - \vec{x}) \tag{5}$$

Where $\omega$ is still the inertia weight, and $r_g \sim U(0,1)$ is a stochastic variable weighted by the user-defined behavioural parameter $\phi_g$. The particle's current position is still denoted by $\vec{x}$ and updated as in the PSO method, and the entire swarm's best known position is known as $\vec{g}$ as well. The algorithm is also identical to that of the PSO in figure 1, with the exception that it too can be simplified somewhat by randomly choosing the particle to update, instead of iterating over the entire swarm. This simplified PSO is called Many Optimizing Liaisons (MOL) to make it easy to distinguish from the original PSO.

A somewhat similar PSO simplification was studied by Clerc [36] on two benchmark problems, although the algorithm had a number of other features that made it more complicated in other ways, and it was also made to be purely deterministic. Another PSO simplification that is more similar to the MOL method has been suggested by Kennedy [37] who called it the "social only" PSO, and used it for optimizing a small ANN. Kennedy's results suggest the simplification does improve on the performance of the basic PSO method, something which shall be tested more rigorously in this study.

# 4 Local Unimodal Sampling

We introduced Local Unimodal Sampling (LUS) in [38], so named because it was designed to deplete unimodal optima, although it has been found to work well for harder optimization problems. The LUS method is used here as the overlaying meta-optimizer for finding the behavioural parameters of PSO and its variants in an offline manner. LUS is often able to locate the optimum in comparatively few iterations, which is required due to the computational time needed for each iteration in meta-optimization.

For the sampling done by the LUS method, the new potential solution $\vec{y} \in \mathbb{R}^n$ is chosen randomly from the neighbourhood of the current solution $\vec{x}$ by adding a random vector $\vec{a}$. When the LUS method is used for meta-optimization these vectors represent different choices of behavioural parameters for the PSO method, meaning that the dimensionality is $n = 4$, and $\vec{x} = [S, \omega, \phi_p, \phi_g]$ with $S$ being the number of particles in the PSO swarm, and $\omega$, $\phi_p$, and $\phi_g$ are the behavioural parameters from Eq.(3). The potential new choice of behavioural parameters $\vec{y}$ is hence found from the current choice $\vec{x}$ as follows:

$$\vec{y} = \vec{x} + \vec{a}$$

where the vector $\vec{a}$ is picked randomly and uniformly from the hypercube bounded by $\pm\vec{d}$, that is:

$$\vec{a} \sim U\left(-\vec{d}, \vec{d}\right)$$

with $\vec{d}$ being the current search-range, initially chosen as the full range of the search-space and decreased during an optimization run as described next. The search-space in meta-optimization constitutes the valid choices of behavioural parameters, and will be detailed later.

When a sample fails to improve the fitness, the search-range $\vec{d}$ is decreased for all dimensions simultaneously by multiplying with a factor $q$ for each failure to improve the fitness:

$$\vec{d} \leftarrow q \cdot \vec{d}$$

with the decrease factor $q$ being defined as:

$$q = \sqrt[n/\beta]{1/2} = 2^{-\beta/n} \tag{6}$$

Here $0 < \beta < 1$ causes slower decrease of the search-range, and $\beta > 1$ causes more rapid decrease. Note that applying this $n$ times yields a search-range reduction of $q^n = 1/2^\beta$, and for $\beta = 1$ this would mean a halving of the search-range for all dimensions. For the experiments in this paper, a value of $\beta = 1/3$ is used as it has been found to yield good results on a broad range of problems. The algorithm for the LUS optimization method is shown in figure 2.

# 5 Meta-Optimization

There are a number of parameters controlling the behaviour and efficacy of the PSO method. To properly compare the performance of the PSO method to that

- Initialize $\vec{x}$ to a random solution in the search-space:

$$\vec{x} \sim U\left(\vec{b}_{lo}, \vec{b}_{up}\right)$$

  Where $\vec{b}_{lo}$ and $\vec{b}_{up}$ are the search-space boundaries.

- Set the initial sampling range $\vec{d}$ to cover the entire search-space:

$$\vec{d} \leftarrow \vec{b}_{up} - \vec{b}_{lo}$$

- Until a termination criterion is met, repeat the following:
  - Pick a random vector $\vec{a} \sim U\left(-\vec{d}, \vec{d}\right)$
  - Add this to the current solution $\vec{x}$, to create the new potential solution $\vec{y}$:

$$\vec{y} = \vec{x} + \vec{a}$$

  - If $(f(\vec{y}) < f(\vec{x}))$ then update the solution:

$$\vec{x} \leftarrow \vec{y}$$

    Otherwise decrease the search-range by multiplication with the factor $q$ from Eq.(6):

$$\vec{d} \leftarrow q \cdot \vec{d}$$

    Note that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the meta-fitness algorithm from figure 5.

Figure 2: LUS algorithm.

of the MOL method, it is essential that good and fair choices of behavioural parameters can be made for both of these optimization methods. While a number of studies have been presented in the literature for properly choosing behavioural parameters for the PSO method, they have been disputed in the introductory review above. Furthermore, no such studies exist for choosing behavioural parameters for the MOL method, so another technique is needed for choosing well-performing behavioural parameters for both the PSO and MOL methods.

The problem of finding the best choice of behavioural parameters for a given optimization method can be considered as an optimization problem in its own right. This is called Meta-Optimization. In other words, the idea is to have an optimization method act as an overlaying meta-optimizer, trying to find the best performing behavioural parameters for another optimization method, which in turn is used to optimize one or more ANN problems. The overall concept is depicted in figure 3 for the PSO method having its behavioural parameters

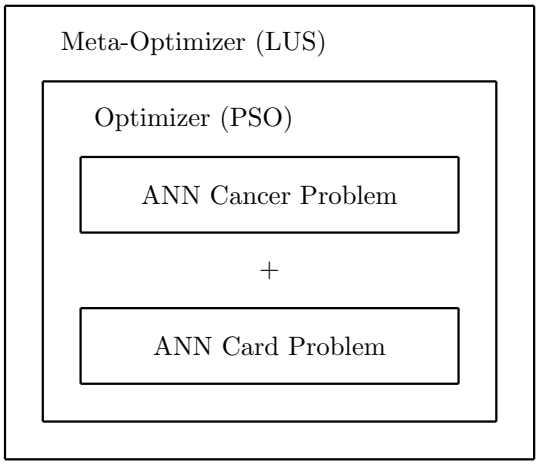


Figure 3: The concept of meta-optimization. The LUS optimization method is used as an overlaid meta-optimizer for finding good behavioural parameters of the PSO method, when it is in turn used to optimize the ANN weights for both the Cancer and Card problems. The same concept is used to find the behavioural parameters of the MOL method.

tuned to perform well on both the ANN Cancer and Card problems. This concept also works for tuning the behavioural parameters of the MOL method, and can also be used with any number of ANN problems.

## 5.1 Meta Fitness Measure

The crux of automatically finding good behavioural parameters for an optimization method, is to define an appropriate fitness measure that can be made the subject of meta-optimization. The fitness measure must reflect how the optimization method is ultimately to be used, but at the same time allow for efficient meta-optimization, and also be simple to describe and implement.

A typical way of performing optimization with the PSO method is to let it run for some predetermined number of iterations. This means the performance of the PSO method and a given choice of behavioural parameters, can be rated in terms of the ANN fitness that can be obtained within this number of iterations. Since the PSO method is stochastic by nature, it will be likely that it gives a different result for each optimization run, and a simple way of lessening this stochastic noise is to perform a number of optimization runs in succession, and use the average of the fitnesses obtained to guide the meta-optimization.

## 5.2 Tuning For Multiple Problems

Another important factor of using the PSO method in practice, is that its performance must generalize well to other optimization problems. Some optimization

methods can have their behavioural parameters tuned to perform well on a single optimization problem, and the parameters seem to automatically work well for other problems [21]. However, from the experiments below it will be clear that this is not the case with the PSO method, as its behavioural parameters appear to easily get overtuned and only perform well on the one problem they were tuned for. Some means of tuning the PSO parameters to perform well on multiple problems is therefore needed, in an attempt to make the performance generalize to other problems as well. This means that meta-optimizing the PSO parameters now becomes a multi-objective optimization problem, which introduces new challenges.

The difficulty with optimizing multiple objectives simultaneously is that the objectives may be conflicting or contradictory. So when the meta-optimizer improves the PSO performance on one problem, it may worsen its performance on another. A number of optimization methods have been introduced to deal with multi-objective optimization problems in general, see e.g. [39] [40] [41] [42]. But these techniques are not suited as the overlaid meta-optimizer because they are more complicated than what is desired.

Instead, one of the simplest approaches to multi-objective optimization is to combine the individual objectives into a single fitness measure, by weighting their sum. It is not clear who is the original author of this technique, but it is mentioned in [43] as having been popular for many years. The weighted-sum approach also works well for meta-optimization, as it is merely the overall performance that matters. So instead of tuning the PSO parameters to work well on just a single ANN problem, each meta-fitness evaluation now consists of evaluating the performance of the PSO method with a given choice of behavioural parameters on two or more ANN problems, and adding the results to create the overall fitness measure used to guide the meta-optimizer. This also has the advantage of being supported directly by the time-saving technique described next.

### 5.3 Pre-Emptive Fitness Evaluation

The algorithm for computing the fitness of the PSO method with a given choice of behavioural parameters, is now to perform optimization runs with PSO on multiple problems, and each of these is repeated a number of times to decrease the influence of stochastic noise. This means each iteration of the meta-optimizer is computationally expensive.

A simple way of saving computational time in meta-optimization is to pre-emptively abort the meta-fitness evaluation once the meta-fitness becomes worse than that needed for the overlaying meta-optimizer to accept the new parameters as an improvement, and the meta-fitness is known not to get better for the rest of the evaluation. This technique is termed Pre-Emptive Fitness Evaluation and has been used by researchers for decades in other contexts, although its original author is difficult to establish as the technique is seldom mentioned in the literature.

Greedy optimization methods are generally compatible with pre-emptive fitness evaluation because they only move their optimizing agents in the case of strict improvement to the fitness. Take for example the LUS method from above, which works by choosing a random point in the vicinity of the current position, and moving to that position only in the case of improvement to the fitness. Therefore the fitness evaluation of the new position can be aborted as soon as it becomes known that it is actually worse than that of the current position; and provided it will not improve later during computation of the fitness.

Pre-emptive fitness evaluation is also directly applicable to fitness functions that are iteratively computed, and where the overhead of monitoring the progress and consequently aborting the fitness evaluation, does not cancel out the gain in computational time arising from only evaluating a part of the fitness function.

The evaluation order of the individual ANN problems can be sorted to further increase time-savings. The ANN problems are sorted according to their last contributions to the overall meta-fitness. This means the ANN problems which appear most likely to cause the meta-fitness evaluation to be aborted will be optimized first.

## 5.4 Meta Fitness Algorithm

To employ pre-emptive fitness evaluation when meta-optimizing behavioural parameters of the PSO method, the fitness measure must be ensured to be non-decreasing and hence only able to grow worse. Since the global minimum for the underlying ANN fitness measure is zero, the best performance of the PSO method when optimizing ANN weights is also a meta-fitness value of zero. The PSO method is used on a number of different ANN problems, and allowed a number of repeated optimization runs on each of these to lessen the effect of stochastic noise, but since the fitness result of each of these runs is non-negative, then the overall fitness sum is non-decreasing and hence suitable for meta-optimization using pre-emptive fitness evaluation.

To use pre-emptive fitness evaluation with other optimization problems than the ANN problems here, it is again important that the fitness is non-negative so the summation is ensured to be non-decreasing. For all practical purposes one should be able to define a fitness boundary for all relevant optimization problems. In the rare case where this is not possible because the fitness boundary is virtually unknown, one can still employ pre-emptive fitness evaluation. This is done by first guessing the boundary, and then adjusting both it and the fitness stored in the overlaid meta-optimizer, whenever it becomes known that the actual boundary is in fact lower. However, since this is not needed here, it has been omitted in the algorithm descriptions to give a simpler presentation.

The overall algorithm for using the LUS method as the overlaid meta-optimizer to find the behavioural parameters of the PSO method is shown in figure 4, consisting of initializing LUS with a random choice of PSO parameters and compute the associated fitness using the algorithm in figure 5. Then a number of LUS iterations are performed trying to find the optimal choice of

PSO parameters. Again, the fitness of a given choice of PSO parameters is computed by the algorithm in figure 5, where the pre-emptive fitness limit $F$ is the limit beyond which the fitness evaluation can be aborted. This fitness limit is passed as an argument by the overlaying LUS meta-optimizer, so that $F$ is the fitness of the currently best known choice of parameters for the PSO method, corresponding to $f(\vec{x})$ in the LUS algorithm from figure 2. For the initial fitness computation using this algorithm, the pre-emptive fitness limit $F$ is merely set to inifinity.

---

- Initialize the LUS meta-optimizer with a random choice of PSO behavioural parameters.
- Perform a number of iterations of the LUS meta-optimizer, where each iteration concists of the following:
  - Determine a new choice of PSO parameters from the previously best known parameters, according to the optimization methodology of the LUS meta-optimizer.
  - Compute a performance measure for how good these new potential PSO parameters fare when PSO is used to optimize a number of ANN problems. The algorithm for computing this meta-fitness measure is shown in figure 5.
  - Keep the new PSO parameters if the meta-fitness is an improvement, otherwise discard the parameters.

---

Figure 4: Overall algorithm for performing meta-optimization of PSO behavioural parameters.

Depending on the experimental settings, the time-saving resulting from the use of pre-emptive fitness evaluation in meta-optimization, ranges from approximately 50% to 85%, and is therefore a significant contribution to the success of this approach to meta-optimization.

# 6 Experimental Results

This section presents the experimental results of optimizing ANN weights using the PSO and MOL methods with meta-optimized behavioural parameters. The results show that both methods are susceptible to overtuning of their behavioural parameters, so the performance increase that is achieved on the ANN problems for which the behavioural parameters are tuned, does not generalize that well to the remainder of the ANN problems. The experiments do show however, that the simpler MOL method is a slight improvement over the orig-

- Initialize the problem-counter: $i \leftarrow 1$, and the fitness-sum: $s \leftarrow 0$
- While $(i \leq M)$ and $(s < F)$, do:
  - Initialize the run-counter: $j \leftarrow 1$
  - While $(j \leq L)$ and $(s < F)$, do:
    - Perform an optimization run on the $i$'th ANN problem using the PSO method with the given choice of behavioural parameters.
    - Add the best fitness obtained in the run (call it $\bar{f}$) to the fitness-sum: $s \leftarrow s + \bar{f}$
    - Increment the run-counter: $j \leftarrow j + 1$
  - Increment the problem-counter: $i \leftarrow i + 1$
- Sort the ANN problems descendingly according to their contributions to the overall fitness sum $s$. This will allow earlier pre-emptive abortion of the fitness evaluation next time.
- Return $s$ to the overlaying meta-optimizer as the fitness value of the PSO method with the given choice of behavioural parameters.

Figure 5: Algorithm for performing a single fitness evaluation in meta-optimization, for rating the performance of the PSO optimization method with a given choice of behavioural parameters.

inal PSO method from which it was derived, at least on the ANN problems tested here.

## 6.1 Datasets

Five ANN datasets are used in this study, and they are all taken from the Proben1 library [33]. These particular datasets are chosen because they give acceptable MSE results within a reasonable number of optimization iterations. All five datasets are classification problems and their specifications are detailed in table 1. The Cancer dataset diagnoses breast tumour as being either benign or malignant from various microscopic measurements. The Card dataset determines whether a credit-card can be granted an applicant by a financial institute according to various data of that applicant. The Gene dataset detects intron/exon boundaries in genetic sequences from a window of 60 DNA sequence elements. The Soybean dataset recognizes diseases in soybeans according to various measurements on the soybean and the plant itself. The Thyroid dataset diagnoses performance of the thyroid from patient query data.

| Problem | Inputs | Classes | Weights | Data Pairs |
|---|---|---|---|---|
| Cancer | 9 | 2 | 50 | 699 |
| Card | 51 | 2 | 218 | 690 |
| Gene | 120 | 3 | 499 | 3175 |
| Soybean | 35 | 19 | 427 | 683 |
| Thyroid | 21 | 3 | 103 | 7200 |

Table 1: ANN dataset specifications, giving for each dataset the number of inputs, the number of possible classifications, the total number of ANN weights and bias-values when using an ANN with a single hidden layer having 4 nodes, and the number of input-output pairs in the dataset.

## 6.2 ANN Initialization & Boundaries

All ANN weights and bias-values are initially picked randomly and uniformly from the range $(-0.05, 0.05)$, which is common for classic BP and works well for PSO also:

$$\forall\, i,\, j,\, k:\ w_{ijk} \sim U(-0.05, 0.05),\ b_{ij} \sim U(-0.05, 0.05)$$

Boundaries for the weights and bias-values are used to limit the search for optima. These boundaries were chosen from experiments with several different optimization methods, suggesting they are applicable in general. Although classic BP does not require such boundaries, the values chosen do not impair the performance for the class of problems considered here. The boundaries are as follows:

$$\forall\, i,\, j,\, k:\ w_{ijk} \in [-7, 7],\ b_{ij} \in [-7, 7]$$

When an optimizing agent steps outside the boundaries of the search-space, it will be moved back to the boundary value.

## 6.3 Optimization Runs And Iterations

To lessen the effect of stochastic variation a number of optimization runs must be performed and their results averaged, so as to give a more truthful measure of the performance that can be expected from a single optimization run. But as some of the ANN problems are very large and time-consuming 10 optimization runs of each ANN problem will have to suffice. For each ANN optimization run a number of iterations are executed, equal to 20 times the total number of weights and bias-values for the problem in question. So for instance, as the ANN Cancer problem can be seen from table 1 to have 50 weights, it means 1000 iterations are performed. An iteration is here taken to mean a single fitness evaluation of the MSE measure.

## 6.4 Meta-Optimization Settings

The experimental settings for meta-optimization with ANN problems are as follows. Six meta-optimization runs are conducted with the LUS method as the overlaying meta-optimizer, trying to find the best performing behavioural parameters of the PSO method. Each meta-optimization run has a number of iterations which equals 20 times the number of parameters to be optimized. So for the PSO method which has four behavioural parameters, 80 iterations will be performed for each meta-optimization run of the overlaying LUS method. The PSO method is in turn made to perform ten optimization runs on each of a number of ANN problems. First these experiments are conducted with the ANN Cancer problem, and then with both the ANN Cancer and Card problems. These problems are used because they are the fastest to compute. These meta-optimization settings are also used for finding the behavioural parameters of the MOL method.

It takes roughly the same amount of time to perform these meta-optimization experiments on the PSO and MOL methods. In particular, when tuning for just the ANN Cancer problem it takes less than 6 hours[1] of computation time to tune the PSO or MOL parameters. When tuning for both the ANN Cancer and Card problems it takes around 136 hours[2] to tune either the PSO or MOL parameters. These measures come from running the experiments on an Intel Pentium M 1.5 GHz laptop computer using an implementation in the ANSI C programming language. It may seem strange that the time usage is similar for tuning both the PSO and MOL methods, when less iterations are performed in meta-optimizing the MOL method, due to it having one less behavioural parameter than the PSO method. The reason is the use of pre-emptive fitness evaluation which tries to limit the number of optimization runs repeated by the PSO and MOL methods. When the time usage is similar it therefore indicates the overlaid meta-optimizer had more success tuning the behavioural parameters

[1] This was because of a slow implementation of the ANN in the NeuralOps source-code library. Using a faster implementation on the same computer brought down the computation time to less than 15 minutes.

[2] Around 3 hours using the new implementation.

of the MOL method than it had with the PSO method, because the LUS meta-optimizer was unable to abort the meta-fitness evaluations as early for the MOL method as it was for the PSO method. This suggests the MOL may even perform better than the PSO method, which is indeed confirmed below.

### 6.5 Meta-Optimized PSO Parameters

The boundaries for the PSO parameters are chosen to be broader than usual, see e.g., the extensive survey and manually conducted experiments in [7]. The PSO parameter boundaries in these meta-optimization experiments are as follows:

$$S \in \{1, \cdots, 200\},\ \omega \in [-2, 2],\ \phi_p \in [-4, 4],\ \phi_g \in [-4, 4]$$

Where $S$ is the swarm-size, and $\omega$, $\phi_p$, and $\phi_g$ are the parameters of the velocity update in Eq.(3). Using these parameter boundaries and the above settings for meta-optimization, the best performing parameters found for the PSO method, when used to optimize just the ANN Cancer problem, are as follows:

$$S = 138,\ \omega = 0.055290,\ \phi_p = 0.112175,\ \phi_g = 1.749212 \tag{7}$$

Another choice of PSO parameters that was discovered during meta-optimization and performed almost as well, is the following:

$$S = 189,\ \omega = 0.027790,\ \phi_p = -1.630695,\ \phi_g = 1.505282$$

Tuning the PSO parameters to perform well on both the ANN Cancer and Card problems instead, resulted in the following PSO parameters:

$$S = 148,\ \omega = -0.046644,\ \phi_p = 2.882152,\ \phi_g = 1.857463 \tag{8}$$

And another choice of PSO parameters resulting from this tuning on two ANN problems and which performed almost as well, are the following:

$$S = 63,\ \omega = 0.549836,\ \phi_p = 1.957720,\ \phi_g = 1.156836$$

Comparing these four sets of behavioural parameters for the PSO method, to the advice given after the extensive and manually conducted experiments by Carlisle and Dozier [7], it is clear that meta-optimization discovers some unusual parameter choices. First is the swarm-size which is commonly recommended to be around 30 particles [7] [6], where the meta-optimized parameters are frequently closer to 150. Then is the inertia weight $\omega$ which is close to zero for all but one of the parameter choices listed above. This would suggest the recurrence of a particle's velocity could be eliminated altogether, and is suggested as a topic for future research. There does not seem to be a consensus in what constitutes a good choice of the weight $\phi_p$, which is indeed also the part of the PSO formula that is sought to be eliminated in the MOL method. But the weight $\phi_g$ appears to yield best performance when taken from the range $\phi_g \in [1, 2]$. It is however suggested in [7] that $\phi_p = 2.8$ and $\phi_g = 1.3$, or at least that their sum

is $\phi_p + \phi_g = 4.1$. But neither of these conditions are satisfied in the PSO parameters discovered through meta-optimization, perhaps with exception of Eq.(8) which approximates this choice of weights; but then has an inertia weight of nearly zero. A different recommendation for PSO parameters is found in [44], where the swarm-size is suggested to be 50, $\phi_p = \phi_g = 1.49445$, and the inertia weight $\omega = 0.729$, which are also far from the parameters discovered through meta-optimization.

Furthermore, the above parameters that were found using meta-optimization, generally do not satisfy the conditions derived through mathematical analysis [3] [4], which was supposed to determine the parameters that would yield good contractive behaviour of the swarm's particles, and hence believed to yield good optimization performance. This confirms that the mathematical analyses of PSO parameters were oversimplified and based on the seemingly ill-founded notion that swarm contraction and optimization performance are tightly connected.

Since there is no consensus view amongst researchers who have experimented with manual selection of PSO parameters, and since the mathematical analysis is lacking, the behavioural parameters that have been found by way of meta-optimization are considered more accurate and reliable in giving good performance for the PSO method, on the problems for which the parameters were tuned.

### 6.6 Meta-Optimized MOL Parameters

Although it is possible for meta-optimization to uncover the MOL simplification by finding a set of PSO parameters having $\phi_p = 0$, it appears from the above that the influence of $\phi_p$ on the PSO performance is somewhat irregular, so it is perhaps unlikely that $\phi_p = 0$ should be discovered as an optimal choice for the PSO parameters, even though it may perfectly well be so. It therefore still makes sense to manually eliminate the $\phi_p$ parameter from the PSO method as done in the MOL method, and see if this simplified variant can be tuned to perform well.

The question then arises of which behavioural parameters to use for the MOL method. But since the method has not received much attention in the literature, there does not exist any guidance for selecting its behavioural parameters. The study on the "social only" PSO by Kennedy [37] appears to merely use the same behavioural parameters as for the PSO. The most sensible course of action is therefore to use meta-optimization to discover good MOL parameters.

The boundaries for the behavioural parameters of the MOL method are chosen similarly to those for the PSO method, because the intrinsic meaning of the parameters are probably related somewhat to their meaning in the PSO method. The boundaries for the MOL method are thus:

$$S \in \{1, \cdots, 200\},\ \omega \in [-2, 2],\ \phi_g \in [-4, 4]$$

Using these boundaries to meta-optimize the behavioural parameters of the MOL method, when it is in turn used to optimize the ANN Cancer problem,

yields the following MOL parameters:

$$S = 74,\ \omega = -0.265360,\ \phi_g = 1.612467 \tag{9}$$

Another choice of MOL parameters performing almost as well, is the following:

$$S = 55,\ \omega = -0.250734,\ \phi_g = 2.119994$$

When instead tuning the MOL parameters to perform well on both the ANN Cancer and Card problems, the best found parameters are as follows:

$$S = 153,\ \omega = -0.289623,\ \phi_g = 1.494742 \tag{10}$$

And another choice of MOL parameters with almost equal performance, is the following:

$$S = 144,\ \omega = -0.172554,\ \phi_g = 2.497806$$

The meta-optimized MOL parameters show clear tendencies of what causes good performance for that method. Namely an inertia weight around $\omega \simeq -0.25$, and the attraction weight to the swarm's best known position must be in the range $\phi_g \in [1.5, 2.5]$, with the optimal MOL performance occurring around $\phi_g \simeq 1.5$. The number of particles $S$ that yields a good performance on the ANN Cancer problem is well below 100, yet when the MOL method must perform well on both the ANN Cancer and Card problems, it appears the swarm-size must be around 150. These parameter choices are far from those commonly suggested for the original PSO method [7] [44], and without the simple and efficient technique of meta-optimization presented here, it would have been a laborious task to have to manually find the MOL parameters giving such good performance.

## 6.7 Results of ANN Weight Optimization

Table 2 shows the results of optimizing ANN weights using the PSO and MOL methods with the behavioural parameters from Eqs.(7) and (9) respectively, which were tuned to perform well on the ANN Cancer problem. Both methods approach the MSE performance of the classic BP method on this problem, but not on the remainder of the ANN problems for which the PSO and MOL parameters were not specifically tuned. This suggests the behavioural parameters are overtuned to the ANN Cancer problem, and demonstrates the necessity of tuning the parameters with regard to multiple ANN problems, so as to make the performance generalize better.

The behavioural parameters from Eqs.(8) and (10) were tuned for both the ANN Cancer and Card problems, and using these when optimizing the weights of all the ANN problems gives the results in table 3. The performance has worsened slightly on the ANN Cancer problem compared to table 2, where the parameters were tuned specifically for just that one problem. But the performance on the ANN Card problem has improved, especially for the MOL method which now approaches the performance of the classic BP method. While performance for

| | Problem | MSE | CLS |
|---|---|---|---|
| PSO | Cancer | 0.043 (0.007) | 0.041 (0.005) |
| | Card | 0.153 (0.021) | 0.185 (0.026) |
| | Gene | 0.188 (0.006) | 0.442 (0.021) |
| | Soybean | 0.048 (1.9e-4) | 0.858 (0.026) |
| | Thyroid | 0.047 (5.2e-4) | 0.074 (1.4e-17) |
| MOL | Cancer | 0.046 (0.006) | 0.042 (0.004) |
| | Card | 0.147 (0.015) | 0.173 (0.025) |
| | Gene | 0.187 (0.007) | 0.447 (0.023) |
| | Soybean | 0.048 (8.4e-5) | 0.850 (0.045) |
| | Thyroid | 0.047 (4.0e-4) | 0.074 (1.4e-17) |
| BP | Cancer | 0.035 (0.004) | 0.036 (0.001) |
| | Card | 0.108 (0.002) | 0.142 (0.003) |
| | Gene | 0.073 (0.004) | 0.136 (0.010) |
| | Soybean | 0.030 (2.3e-4) | 0.477 (0.022) |
| | Thyroid | 0.047 (3.8e-5) | 0.074 (1.4e-17) |

Table 2: PSO and MOL optimization of ANN weights. The behavioural parameters have been tuned for just the ANN Cancer problem. Table shows the average best MSE and corresponding CLS obtained over 10 runs. Numbers in parentheses are the standard deviations.

the PSO and MOL methods have also improved somewhat on the ANN Gene problem for which their parameters were not specifically tuned, the performance is far from that of the BP method. Performance with regard to the MSE fitness measure is almost unchanged on the ANN Soybean and Thyroid problems.

Regarding the CLS fitness measure that is computed by using the ANN weights that were optimized for the MSE measure, the same performance tendencies are evident, only on a greater scale. Again the CLS performance improves on the ANN problems for which the PSO and MOL parameters are tuned, but the CLS performance on the Gene and Soybean problems are far from the performance of the classic BP method, although they do improve somewhat when the PSO and MOL parameters are tuned for more than just one ANN problem.

It can be concluded from these experiments that both the PSO and MOL methods are susceptible to overtuning, as parameters that are tuned to perform well on one or even two ANN problems, do not necessarily generalize well to other ANN problems. Future research must seek a way to alleviate this, perhaps by making meta-optimization for multiple problems even more efficient. However, the MOL method does perform slightly better overall than the PSO method on these ANN problems, and the topic of this study is therefore found to be viable, namely that even the basic PSO method does offer room for simplification.

Figures 6, 7, 8, 9 and 10 show the averaged fitness traces for the optimization runs that led to the results in table 3. It can be seen from these fitness traces, that the classic BP method is generally more rapid than the PSO and MOL

| | Problem | MSE | CLS |
|---|---|---|---|
| PSO | Cancer | 0.051 (0.008) | 0.046 (0.008) |
| | Card | 0.143 (0.011) | 0.170 (0.019) |
| | Gene | 0.183 (0.004) | 0.436 (0.016) |
| | Soybean | 0.048 (1.4e-4) | 0.844 (0.037) |
| | Thyroid | 0.047 (2.5e-4) | 0.074 (1.4e-17) |
| MOL | Cancer | 0.059 (0.013) | 0.056 (0.015) |
| | Card | 0.117 (0.006) | 0.145 (0.010) |
| | Gene | 0.174 (0.005) | 0.405 (0.011) |
| | Soybean | 0.047 (4.9e-4) | 0.770 (0.047) |
| | Thyroid | 0.047 (2.1e-4) | 0.074 (1.4e-17) |
| BP | Cancer | 0.035 (0.004) | 0.036 (0.001) |
| | Card | 0.108 (0.002) | 0.142 (0.003) |
| | Gene | 0.073 (0.004) | 0.136 (0.010) |
| | Soybean | 0.030 (2.3e-4) | 0.477 (0.022) |
| | Thyroid | 0.047 (3.8e-5) | 0.074 (1.4e-17) |

Table 3: PSO and MOL optimization of ANN weights. The behavioural parameters have been tuned for both the ANN Cancer and Card problems. Table shows the average best MSE and corresponding CLS obtained over 10 runs. Numbers in parentheses are the standard deviations.

methods in approaching the optimum of these ANN problems. The PSO and MOL methods do sometimes have an advantage though, for example midways during optimization of the ANN Cancer problem where they outperform the BP method for a while. But the troubling tendencies of overtuning that were demonstrated above are also prevalent in these fitness traces, as the PSO and MOL methods have a performance comparable to that of the BP method on the ANN problems for which their behavioural parameters were tuned, but this does not generalize to the problems for which the parameters were not tuned. However, the simpler MOL method also appears to be a slight improvement on the PSO method from looking at these fitness traces.

## 6.8 Comparison To Differential Evolution

The advantage of using the PSO or MOL method over the classic BP method in general, is that the gradient is not required by the PSO and MOL methods. This makes it easier to devise new mathematical models requiring optimization, because the gradient is typically laborious to derive.

Another optimization method which also does not rely on the gradient is known as Differential Evolution (DE) and was originally introduced by Storn and Price [45]. The DE method had its behavioural parameters tuned to perform well on the ANN Cancer problem by us [21]. Since the experimental settings were exactly the same as those used in this study, it is possible to compare the performance of the DE method with that of the PSO and MOL methods.

The results of using the DE method to optimize ANN weights are reprinted from [21] in table 4, and should be compared to the results of the PSO, MOL, and

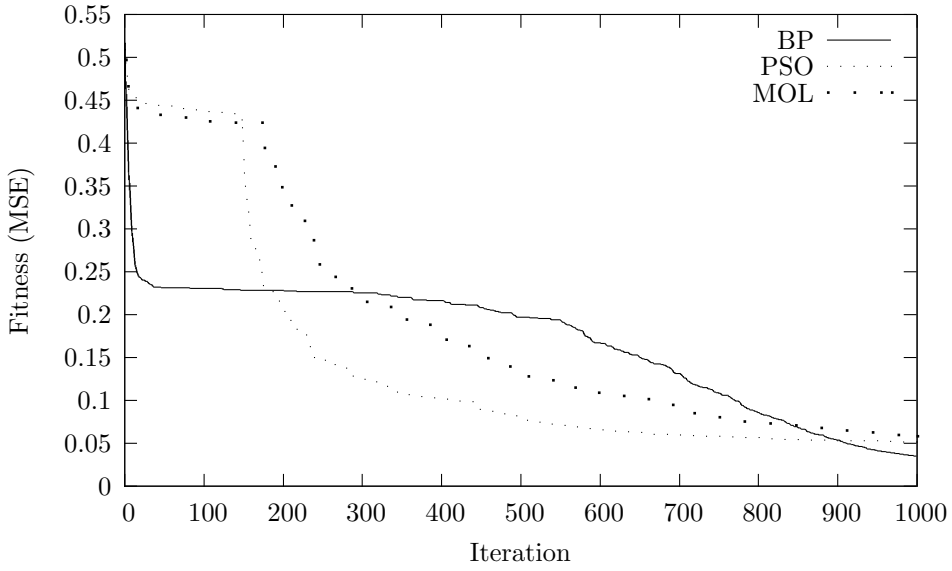


Figure 6: Cancer-problem fitness trace. Plot shows the average fitness obtained at each iteration of the 10 optimization runs. Both PSO and MOL parameters are tuned for the Cancer and Card problems. Note how MOL leads over PSO until around iteration 160, after which MOL is outperformed by PSO, which also outperforms the BP until all three methods have somewhat similar performance towards the end, with the BP winning slightly.

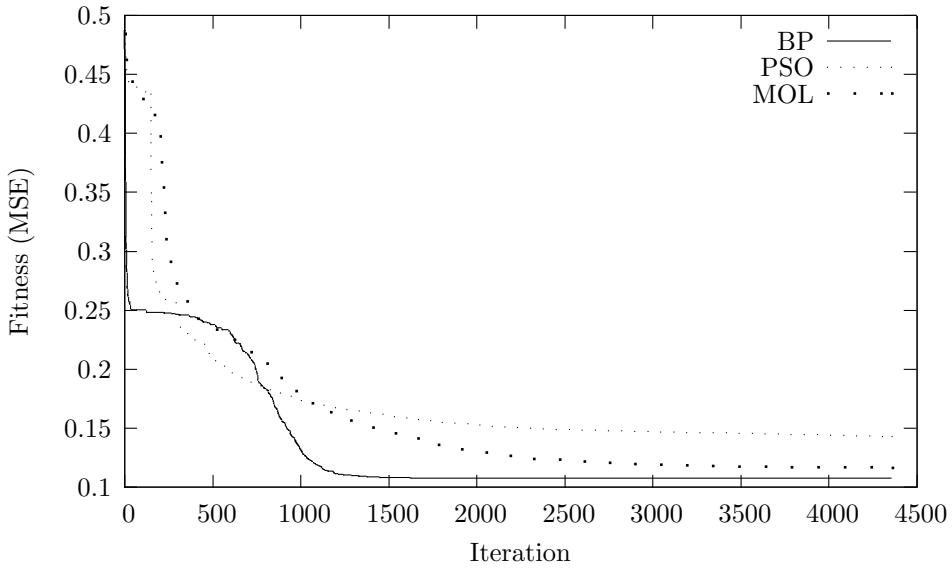


Figure 7: Card-problem fitness trace. Plot shows the average fitness obtained at each iteration of the 10 optimization runs. Both PSO and MOL parameters are tuned for the Cancer and Card problems. Note how MOL performs significantly better than PSO from around iteration 1100. Both methods are outperformed by BP from around iteration 600 though.

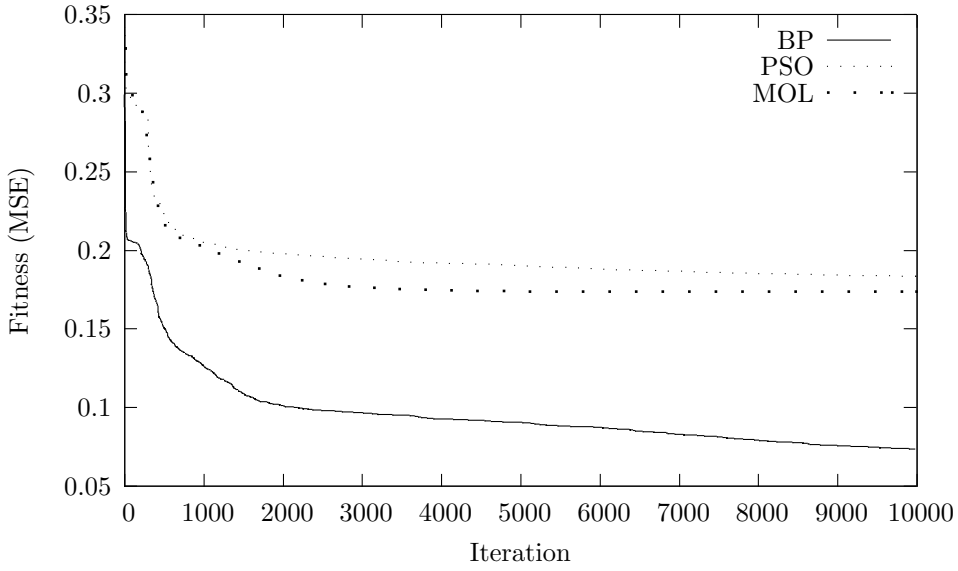


Figure 8: Gene-problem fitness trace. Plot shows the average fitness obtained at each iteration of the 10 optimization runs. Both PSO and MOL parameters are tuned for the Cancer and Card problems. Note how MOL already early outperforms PSO. Although BP performs vastly better on this Gene problem, perhaps because the PSO and MOL parameters were not specifically tuned for this problem.

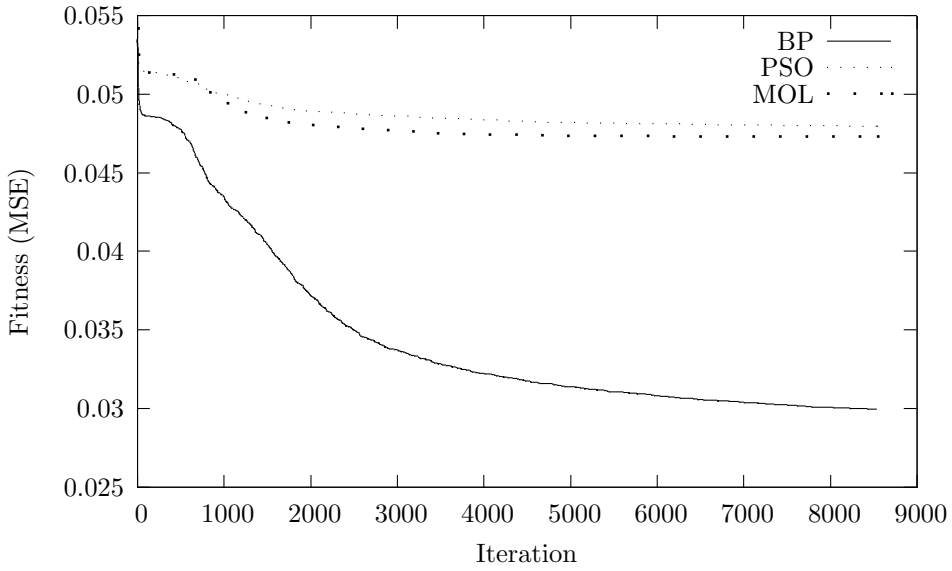


Figure 9: Soybean-problem fitness trace. Plot shows the average fitness obtained at each iteration of the 10 optimization runs. Both PSO and MOL parameters are tuned for the Cancer and Card problems. Note how PSO and MOL have somewhat similar performance, although MOL slightly outperforms PSO from around iteration 1000. But BP performs vastly better on this Soybean problem, perhaps because the PSO and MOL parameters were not specifically tuned for this problem.

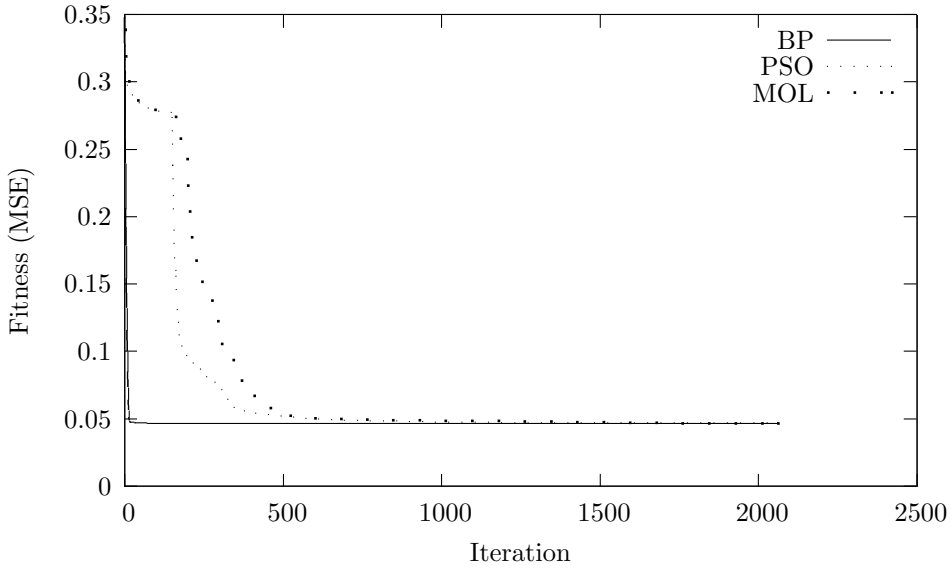


Figure 10: Thyroid-problem fitness trace. Plot shows the average fitness obtained at each iteration of the 10 optimization runs. Both PSO and MOL parameters are tuned for the Cancer and Card problems. Note how PSO, MOL, and BP have similar performance from around a third way into the optimization run. But BP is more rapid in approaching the optimum, while PSO and MOL are clearly slower than BP, but with PSO being slightly better than MOL.

| | Problem | MSE | CLS |
|---|---|---|---|
| DE | Cancer | 0.033 (0.004) | 0.033 (0.003) |
| | Card | 0.091 (0.005) | 0.118 (0.008) |
| | Gene | 0.123 (0.009) | 0.223 (0.039) |
| | Soybean | 0.038 (0.001) | 0.576 (0.058) |
| | Thyroid | 0.042 (0.001) | 0.074 (0.001) |

Table 4: DE optimization of ANN weights. The behavioural parameters have been tuned for just the ANN Cancer problem. Table shows the average best MSE and corresponding CLS obtained over 10 runs. Numbers in parentheses are the standard deviations. These results are reprinted from [21].

BP methods from table 3. The PSO and MOL methods had their behavioural parameters tuned to perform well on both the ANN Cancer and Card problems, while the DE method had its parameters tuned for just the Cancer problem. It is obvious from comparing these tables that the DE method performs much better on these ANN problems, and that it also generalizes well to ANN problems for which it was not specifically tuned. The DE method does not only approach the performance of the classic BP method, it often exceeds it. The DE method therefore appears to be preferable over the PSO and MOL methods in optimizing ANN problems in general.

# 7 Conclusion

This was a study on simplifying the PSO method without impairing its performance on a number of ANN weight optimization problems. To achieve this a technique for tuning the behavioural parameters of an optimizer was presented, in which an overlaid meta-optimizer was used. This meta-optimization technique was able to tune the behavioural parameters to perform well on several ANN problems simultaneously, in an attempt to make the performance increase generalize to ANN problems for which the behavioural parameters were not specifically tuned. This was necessary because the PSO method and its simplification, the MOL method, both appear highly susceptible to overtuning of their behavioural parameters, if just a single ANN problem was used in meta-optimization. Two ANN problems were therefore used during meta-optimization of PSO and MOL parameters, and while more ANN problems would have been desirable, it would have meant an impractical increase in computation time.

Even with these limitations of meta-optimization, it was still found that the simplified MOL method was a viable variant, as it had performance comparable to that of the PSO method from which it was derived; and provided the behavioural parameters were properly tuned through the use of meta-optimization. In fact, the MOL method sometimes even exceeded the original PSO method on some of these ANN problems. However, both the PSO and MOL methods could not have their behavioural parameters tuned so as to perform on par with the classic gradient-based BP method on all these ANN problems. This

might be possible as computational power increases in the future, and meta-optimization can be carried out on all five ANN problems simultaneously. The meta-optimization technique presented here readily supports this.

Additionally the experimental results of the PSO and MOL methods were compared to the results of Differential Evolution (DE), from which it can be concluded that the DE method is probably a better choice for optimizing ANN problems, not only in terms of performance but also in terms of how well the DE parameters generalize to ANN problems for which they were not specifically tuned.

## Source-Code

Source-code implemented in the ANSI C programming language and used in the experiments in this paper, can be found in the SwarmOps library on the internet address: `http://www.Hvass-Labs.org/`

## References


[1] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Perth, Australia, 1995.

[2] S. Haykin. *Neural Networks: a comprehensive foundation*. Prentice Hall, 2nd edition, 1999.

[3] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, Faculty of Natural and Agricultural Science, November 2001.

[4] I.C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85:317 – 325, 2003.

[5] Y. Shi and R.C. Eberhart. Parameter selection in particle swarm optimization. In *Proceedings of Evolutionary Programming VII (EP98)*, pages 591 – 600, 1998.

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

[7] A. Carlisle and G. Dozier. An off-the-shelf PSO. In *Proceedings of the Particle Swarm Optimization Workshop*, pages 1 – 6, 2001.

[8] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6:58 – 73, 2002.

[9] M. Løvbjerg and T. Krink. Extending particle swarm optimisers with self-organized criticality. In *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002)*, volume 2, pages 1588 – 1593, 2002.

[10] J. Riget and J.S. Vesterstrøm. A diversity-guided particle swarm optimizer – the ARPSO. *Technical Report 2002-02, Department of Computer Science, University of Aarhus*, 2002.

[11] M. Løvbjerg and T. Krink. The lifecycle model: combining particle swarm optimisation, genetic algorithms and hillclimbers. In *Proceedings of Parallel Problem Solving from Nature VII (PPSN)*, pages 621 – 630, 2002.

[12] R.C. Eberhart and Y. Shi. Evolving artificial neural networks. In *Proceedings of International Conference on Neural Networks and Brain*, pages PL5 – PL13, Beijing, China, 1998.

[13] H. Fan. A modification to particle swarm optimization algorithm. *Engineering Computations: International Journal for Computer-Aided Engineering*, 19:970–989, 2002.

[14] H-W. Ge, Y-C. Liang, and M. Marchese. A modified particle swarm optimization-based dynamic recurrent neural network for identifying and controlling nonlinear systems. *Computers and Structures*, 85(21-22):1611–1622, 2007.

[15] R.E. Mercer and J.R. Sampson. Adaptive search using a reproductive meta-plan. *Kybernetes (The International Journal of Systems and Cybernetics)*, 7:215–228, 1978.

[16] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[17] T. Bäck. Parallel optimization of evolutionary algorithms. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature (PPSN)*, pages 418–427, London, UK, 1994. Springer-Verlag.

[18] T. Bäck. *Evolutionary Algorithms in Theory and Practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford University Press, Oxford, UK, 1996.

[19] A.J. Keane. Genetic algorithm optimization in multi-peak problems: studies in convergence and robustness. *Artificial Intelligence in Engineering*, 9:75–83, 1995.

[20] M. Meissner, M. Schmuker, and G. Schneider. Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, 7(125), 2006.

[21] M.E.H. Pedersen and A.J. Chipperfield. Tuning differential evolution for artificial neural networks. Technical Report HL0803, Hvass Laboratories, 2008.

[22] K.E. Parsopoulos and M.N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1:235–306, 2002.

[23] C.M. Bishop. *Neural Networks For Pattern Recognition*. Oxford University Press, 1995.

[24] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[25] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[26] A.R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

[27] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[28] P.J. Werbos. *Beyond Regression: new tools for prediction and analysis in the behavioural sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

[29] P.J. Werbos. *The Roots of Backpropagation: from ordered derivatives to neural networks and political forecasting*. Wiley-Interscience, 1994.

[30] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: foundations*, pages 318–362. MIT Press, 1986.

[31] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[32] M.D. Richard and R.P. Lippmann. Neural network classifiers estimate bayesian a-posteriori probabilities. *Neural Computation*, 3(4):461–483, 1991.

[33] L. Prechelt. Proben1 – a set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Faculty of Informatics, University of Karlsruhe, Germany, 1994.

[34] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*, pages 69–73, Anchorage, AK, USA, 1998.

[35] R.C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC)*, pages 84–88, San Diego, CA, USA, 2000.

[36] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1951–1957, Washington D.C., USA, 1999.

[37] J. Kennedy. The particle swarm: social adaptation of knowledge. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 303–308, Indianapolis, USA, 1997.

[38] M.E.H. Pedersen and A.J. Chipperfield. Local unimodal sampling. Technical Report HL0801, Hvass Laboratories, 2008.

[39] D.E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.

[40] J. Horn, N. Nafpliotis, and D.E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE International Conference on Evolutionary Computation*, volume 1, pages 82–87, New Jersey, USA, 1994.

[41] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[42] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[43] C.M. Fonseca and P.J. Fleming. Multiobjective optimization. In T. Bäck, D.B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C4.5:1 – 9. IOP Publishing and Oxford University Press, 1997.

[44] R.C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 81 – 86, 2001.

[45] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341 – 359, 1997.