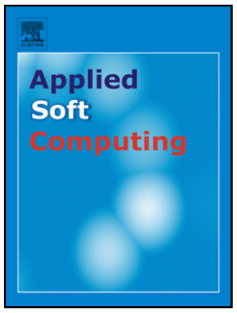


Title: A Meta Optimisation Analysis of Particle Swarm Optimisation Velocity Update Equations for Watershed Management Learning

Author: Karl Mason Jim Duggan Enda Howley

PII:	S1568-4946(17)30623-3
DOI:	https://doi.org/doi:10.1016/j.asoc.2017.10.018
Reference:	ASOC 4513
To appear in:	*Applied Soft Computing*
Received date:	6-11-2016
Revised date:	5-9-2017
Accepted date:	9-10-2017

A Meta Optimisation Analysis of Particle Swarm Optimisation Velocity Update Equations for Watershed Management Learning

Karl Mason[*], Jim Duggan, Enda Howley

Discipline of Information Technology,
National University of Ireland Galway

Abstract

Particle Swarm Optimisation (PSO) is a general purpose optimisation algorithm used to address hard optimisation problems. The algorithm operates as a result of a number of particles converging on what is hoped to be the best solution. How the particles move through the problem space is therefore critical to the success of the algorithm. This study utilizes meta optimisation to compare a number of velocity update equations to determine which features of each are of benefit to the algorithm. A number of hybrid velocity update equations are proposed based on other high performing velocity update equations. This research also presents a novel application of PSO to train a neural network function approximator to address the Watershed Management problem. It is found that the standard PSO with a linearly changing inertia, the proposed hybrid Attractive Repulsive PSO with Avoidance of Worst Locations (AR PSOAWL) and Adaptive Velocity PSO (AV PSO) provide the best performance overall. The results presented in this paper also reveal that commonly used PSO parameters do not provide the best performance. Increasing and negative inertia values were found to perform better.

Keywords: Particle Swarm Optimisation, PSO, Velocity Update Equation, Constriction, Inertia, Meta Optimisation, Watershed Management, Function Approximation, Neural Networks, Learning

1. Introduction

Particle Swarm Optimisation (PSO) is an optimisation algorithm that consists of a number of particles exploring a problem space and ultimately converging on a solution [21]. The particles evaluate potential solutions and share information with one another. This information is used to direct their movement so that they move towards the best known solutions. The PSO algorithm has been applied to numerous real world problem domains since its first proposal. These include design, scheduling and routing problems across several disciplines and industries ranging from imaging to energy production [1]. The main advantages of meta heuristic optimisation algorithms such as PSO, Differential Evolution and Scatter Search is their robustness, versatility and applicability to a wide range of problems. Traditional optimisation methods such as gradient decent struggle with certain problems, e.g. problems with noise and problems that are otherwise non differentiable. Such problems do not pose a problem to PSO. Another class of problems that PSO is well suited to are very large problems. Problems that are classified as NP-complete or NP-Hard (non-deterministic polynomial-time). These problems increase in size at an exponential rate as the number of parameters increase and are therefore much too time consuming for deterministic optimisation algorithms to solve. Algorithms such as PSO can provide good approximations to these problems however. The drawback to PSO is that it is heavily reliant on correct parameter selection and cannot guarantee that it will converge on the optimum solution. This research will aim to address the former of these two issues. There are more comprehensive studies that outline in detail the advantages and disadvantages of heuristic optimisation algorithms [46].

[*]I am corresponding author

Email addresses: k.mason2@nuigalway.ie (Karl Mason), jim.duggan@nuigalway.ie (Jim Duggan), ehowley@nuigalway.ie (Enda Howley)

1.1. Velocity Update Equations

This research will focus on the area of particle movement, in particular how the particles update their velocity. There has been a wealth of research published in this area, exploring all aspects of how best to update the particles' velocity. When the algorithm was first proposed in 1995, a particle's velocity was updated based on its velocity at the previous time step, the particle's best previous position and the groups best previous position [21]. An inertia term was added at a later date to better enable the particles to converge [38], this was followed shortly by a linearly decreasing inertia term [39]. It was later shown that implementing a constriction term rather than the inertia term is mathematically guaranteed to provide stable convergence [10]. An example of a more recent development on the inertia term is the adaptive inertia weight which changes between the maximum and minimum value based on the success of particles on improving their fitness between iterations [33].

There are many velocity update equations in the literature that implement novel operators within the velocity update equation in an effort to improve performance. The Attractive Repulsive PSO is an example of this [36]. This algorithm uses a diversity measure to determine when the swarm has converged. If the swarm is deemed to have converged, the particles then move away from the best locations found. A similar idea was proposed in the form of the PSO CV where a velocity control parameter was implemented as a means giving the particles' extra velocity when they slowed down closer to the optimum [17]. The Dissipative PSO is another PSO variant which gives a particle a random velocity and location if a random number generated is below a crucial value [43].

Incorporating the worst locations of the problem space into the motion of the particles was first proposed in 2005 [44]. This velocity update equation only took the worst locations into account when updating the particles' velocity and as such only avoided the worst locations rather than converging on the best locations. This issue was addressed in later research that also enabled the particles to converge [37, 18, 27, 25, 28].

In an opposite direction of much of the above research there has also been efforts made to reduce the complexity of the velocity update equation, most notably the Many Optimising Liaisons (MOL) [35] and the Adaptive Velocity PSO (AV PSO) [3]. MOL is a PSO variant that simplifies the velocity update equation by removing the influence of a particle's personal best position and focuses solely on the group's best position. The AV PSO aimed to reduce the complexity of the algorithm by removing most of the parameters that have to be tuned for the algorithm to run optimally. Instead this variant utilizes the Euclidean distance of a particle from the best location to update its velocity.

1.2. Meta Optimisation & Parameter Selection

When implementing the PSO algorithm, it is important to ensure that adequate parameters are selected. Parameter selection itself can be thought of as an optimisation problem. There are generally two approaches to addressing the issue of parameter selection: 1) A parameter sweep consisting of a brute force search. 2) Meta Optimisation. A parameter sweep is sufficient when only one or two parameters need to be selected, however it becomes very computationally expensive as the number of parameters increases. For this reason meta optimisation will be utilised in this paper to ensure each velocity update equation is tuned to give the best performance. Meta optimisation refers to applying an optimisation algorithm to the parameters of another optimisation algorithm. The field of meta optimisation dates back to 1978 when it was first applied to tune a genetic algorithm [32]. In the years since, meta optimisation has been applied to ant colony optimisation [6], differential evolution [34], COMPLEX-RF [22], particle swarm optimisation [31, 35] and genetic algorithms [14, 4, 20].

There are limitations to the previous studies conducted in applying meta optimisation to PSO. The first study in this area, the Optimised PSO (OPSO), optimises the PSO parameters separately per problem [31]. It is considered desirable within PSO research to find PSO parameters that give the best performance over a range of problems rather than just one. Although the authors of the OPSO presented their algorithm which incorporates meta optimisation as an optimisation algorithm itself rather than using meta optimisation to tune the PSO parameters, as done in later research [35] and in this research. The second limitation with each of the previous applications of meta optimisation to PSO is the size of the experiments conducted. With regards to the OPSO, meta optimisation was only applied to 1 PSO variant for 1 problem at a time [31]. In the later Many Optimising Liaisons (MOL) [35], meta optimisation was applied to 2 PSO variants over 5 problems. The research presented in this paper is much more comprehensive in terms of scale, implementing 20 PSO variants based on their velocity update equations evaluated over 8 problems.

1.3. Watershed Management

In order to gauge the performance of each velocity update equation, they will be applied to 8 optimisation problems. Of these problems, 7 are standard test optimisation functions. The final problem is a novel application of PSO to a neural network function approximator for the Watershed Management problem. This problem is essentially a resource management problem [45]. The resource in question is water. There are a number of interested parties that wish to withdraw water from the system. These include water to sustain a city, water for farm irrigation, water for hydroelectric power generation and water for the surrounding ecosystem. The problem consists of many constraints and multiple flow scenarios. Previous approaches to addressing this problem include Multi Agent Systems (MAS) [45, 2], Multi Agent Reinforcement Learning (MARL) [30], MAS combined with Genetic Algorithm [5], Multi Population Evolutionary Algorithm [12] and PSO [30]. The single neural network function approximator approach implemented in this research differs from many of these previous methods as it does not implement multiple individual controllers to each problem variable. There are many examples in the literature of training a neural network using PSO, however this is the first application of such approach to the watershed management problem. In the previous application of PSO to the Watershed Management problem, PSO was applied directly to the problem variables rather to find the optimum solution. This application of PSO to the watershed management problem is far more novel and interesting as it utilizes PSO in an offline learning manner to train a neural network to be able to find the optimum solution to the problem.

1.4. Contribution & Structure

The contributions of this paper are as follows:

1. Provides a comprehensive comparison of a wide range of velocity update equations using meta optimisation.
2. Establishes which set of parameters leads to the optimum PSO performance.
3. Addresses the questions: Does a constricted PSO perform better than a PSO with Inertia? Does a stable PSO convergence result in better performance? What is the best performing velocity update equation?
4. Investigates if hybridizing various velocity update equations will result in better performance.
5. Provides a novel application of a neural network function approximator to the Watershed Management problem.

The rest of the paper is structured as follows: Section 2 will provide a detailed outline of the Particle Swarm Optimisation algorithm and the velocity update equations that will be examined. Section 3 will explain the concept of Meta Optimisation along with how it will be applied in this research. In Section 4, concepts such as Learning, Artificial Neural Networks and the application of PSO will be described. Section 5 will present the Watershed Management problem. The results of each of the experiments will be presented in Section 6 and finally, the conclusions that can be drawn from these results will be made in Section 7. Here, future work will also be outlined.

2. Particle Swarm Optimisation

The PSO algorithm consists of a number of particles whose purpose is to evaluate candidate solutions and eventually move towards the best solution [21]. Initially these particles are distributed throughout the problem space with a random position and random velocity. The position and velocity of a particle at a time t are referred to as $\vec{x}_t$ and $\vec{v}_t$ respectively. At each time step each particle evaluates its position within the problem space defined by an objective function. This objective function measures the fitness of the particles current position which represents a candidate solution. Every particle remembers its previous best position. If a new position has a better fitness than the previous best position for that particle, the particle will remember this new position as its personal best position $\vec{pb}$. Each particle also has access to the best position within its neighbourhood of particles $\vec{gb}$. The other particles within a particle's neighbourhood are dictated by the topology used. Each particle updates its velocity, and as a result its position, using its own best position and that of its neighbours. Balancing this cognitive and social behaviour is critical to the success of the PSO. The motion of the particles throughout the problem space is defined by their equations of motion below:

$$\vec{v}_{t+1} = \chi(\vec{v}_t + r_1 c_1(\vec{pb}_t - \vec{x}_t) + r_2 c_2(\vec{gb}_t - \vec{x}_t)) \tag{1a}$$

$$\vec{x}_{t+1} = \vec{x}_t + \vec{v}_t \tag{1b}$$

Where r_1 and r_2 are random numbers between 0 and 1. The terms c_1 and $c_2 = 2.05$ are acceleration coefficients. The χ term is the constriction factor and is defined as:

$$\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \tag{2a}$$

$$\varphi = c_1 + c_2 \tag{2b}$$

Where $\chi \approx 0.72984$, and $c_1 = c_2 = 2.05$ have been mathematically proven to provide stable convergence [10]. The above definition of the various PSO parameters has been proposed as the standard PSO [9]. For this reason, this variation of PSO will be implemented as the overlaying meta optimiser in this research. The pseudo-code in Algorithm 1 below describes the structure of the PSO algorithm.

```
Create N particles with random position and velocity
while Iteration t <Imax do
    for Particle = 1 to N do
        Update personal best position
        Update neighbourhood best position
        Evaluate particle's current position
        Update particle's velocity
        Update particle's position
    end
end
Return best solution
```

Algorithm 1: PSO Algorithm

2.1. Velocity Update Equation Variations

The velocity update equation outlined by Equation 1 will be implemented in the PSO meta optimiser. Table 1 outlines the first set of the velocity update equations that will be evaluated.

The velocity update equations in Table 1 have been divided into three categories to better allow their performance to be analysed. Velocity equations 1, 2 & 3 are all very similar in structure and are considered to be standard. Velocity equations 4, 5, 6 & 7 all incorporate the worst locations of the problem space and are therefore grouped together. The final category of velocity update equations are non-standard velocity update equations, Velocity equations 8 to 13. Additional velocity update equations will be added later to test hybridized velocity update equations. The overall performance of each algorithm will be measured based on convergence speed, average fitness and fitness standard deviation.

The first and second velocity update equations in Table 1, are the most commonly used velocity update equations in the PSO literature. This first velocity equation implements a constriction factor χ and has been shown to provide stable convergence with certain parameters outlined at the beginning of this section by Equation 2a. Although these parameter values provide stable convergence, applying meta optimisation to these values will also determine if these values provide the best performance. This will address the research question: Does a stable PSO convergence result in better performance? Velocity equation 2 utilizes the linearly decreasing inertia term ω. The third velocity update equation uses an adaptive inertia weight. This ω changes its value based on the success of particles on improving their fitness between iterations. Comparing this group of velocity equations will address research question 3, Does a constricted PSO perform better than a PSO with Inertia?

Velocity equation 4 is the first to incorporate worst locations. It utilizes the both the personal and group worst positions of the particle but not the best locations to dictate the particles movement. Velocity update equations 5 and 6 both utilize the best and worst locations. Velocity equation 6 utilizes the best and worst personal and group positions, while velocity equation 5 does not utilize the group worst position. A constriction factor is implemented in velocity equation 7 rather than an inertia term. The worst locations also have an inverse effect on the particles' velocity in this equation.

Table 1: Velocity Update Equations

Algorithm	Velocity Equation	Parameters
1) Constriction [10]	$\vec{v}_{t+1} = \chi(\vec{v}_t + r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t))$	c_1, c_2, χ
2) Inertia [39]	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t)$ Where ω_{start} is the initial inertia and ω_{end} is the final inertia	$c_1, c_2, \omega_{start}, \omega_{end}$
3) AIW PSO [33]	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t)$	$c_1, c_2, \omega_{start}, \omega_{end}$
	Where $\omega = (\omega_{start} - \omega_{end})P_s + \omega_{end}$, P_s is fraction of particles with improved fitness - $P_s = \frac{\sum^{n} S_i}{n}$, S_i is the success of particle i, $S_i = 1$ if particle improved since last iteration, otherwise $S_i = 0$	
4) PSOA [44]	$\vec{v}_{t+1} = \vec{v}_t + r_1c_1(\vec{x}_t - \vec{pw}_t) + r_2c_2(\vec{x}_t - \vec{gw}_t)$ Where $\vec{pw}_t$ and $\vec{gw}_t$ are a particle's personal and group worst positions	c_1, c_2
5) NPSO1 [37]	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{x}_t - \vec{pw}_t) + r_3c_3(\vec{gb}_t - \vec{x}_t)$	$c_1, c_2, c_3, \omega_{start}, \omega_{end}$
6) NPSO2 [18]	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t) + r_3c_3(\vec{x}_t - \vec{pw}_t) + r_4c_4(\vec{x}_t - \vec{gw}_t)$	$c_1, c_2, c_3, c_4, \omega_{start}, \omega_{end}$
7) PSO AWL [29]	$\vec{v}_{t+1} = \chi(\vec{v}_t + t_1 + t_2 + t_3 + t_4)$	c_1, c_2, c_3, c_4, χ
	Where $t_1 = r_1c_1(\vec{pb}_t - \vec{x}_t), t_2 = r_2c_2(\vec{gb}_t - \vec{x}_t), t_3 = r_3c_3(\frac{t_1}{1+\lvert\vec{x}_t - \vec{pw}_t\rvert}), t_4 = r_4c_4(\frac{t_2}{1+\lvert\vec{x}_t - \vec{gw}_t\rvert})$	
8) MOL [35]	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{gb}_t - \vec{x}_t)$	$c_1, \omega_{start}, \omega_{end}$
9) PSO CV [17]	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t) + \frac{c_3 r}{\vec{v}^2}$	$c_1, c_2, c_3, \omega_{start}, \omega_{end}$
10) AR PSO [36]	$\vec{v}_{t+1} = \omega\vec{v}_t + dir(r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t))$	$c_1, c_2, \omega_{start}, \omega_{end}$
	Where $dir = -1$ if $(dir > 0 \& div < d_{low}), dir = 1$ if $(dir < 0 \& div > d_{high})$, div is the swarm diversity, $div = \frac{1}{\lvert S\rvert\lvert L\rvert} \sum_{i=1}^{\lvert S\rvert} \sqrt{\sum_{j=1}^{N} (p_{ij} - \bar{p}_j)^2}$, $\lvert S\rvert$ is the swarm size, L is the longest diagonal, N is the number of dimensions, p is a particle's position, $\bar{p}$ is the average position, $d_{low} = 5 \times 10^{-6}$ and $d_{high} = 0.25$	
11) AR PSO2 [36]	$\vec{v}_{t+1} = \omega\vec{v}_t + dir(r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t))$	$c_1, c_2, \omega_{start}, \omega_{end}, d_{low}, d_{high}$
12) DPSO [43]	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t)$ if$(rand() < c_v), v_{id} = rand() \times V_{max,d}$, if $(rand() < c_l), x_{id} = rand() \times x\_range_d$	$c_1, c_2, \omega_{start}, \omega_{end}, c_v, c_l$
13) AV PSO [3]	$\vec{v}_{t+1} = \frac{p\vec{Dist}_t}{maxDist_t}$	n
	Where $pDist_{i,t} = \sqrt{\sum_{i=1}^{d}(gb_{i,t} - x_{i,t})^2}$, $maxDist_t = max(p\vec{Dist}_t)$, $\vec{x} = (1-\tau)\vec{x} + \vec{v}$, $\tau = n \times rand() \times (1 - rand())$	

The first velocity equation in the final category of non-standard velocity update equations is the many optimising liaisons equation. This velocity equation only utilises the group best position and not a particle's personal best position. A velocity control parameter is added in velocity equation 9 with the aim of giving particles extra velocity when they have converged. Velocity equations 10 and 11 are very similar, the only difference is that the high & low diversity thresholds will be parameters to be optimised in velocity equation 11. These velocity equations implement a repulsive force once the diversity of particles is low enough. The final two velocity equations are the only two PSO variations that alter the position update equation specified in 1. This is because their original proposals specified these position update equations. Velocity equation 12 adds randomness into the particles velocity and position. Velocity equation 13 takes the Euclidean distance between a particle's current position and the best position and uses this to update the particle's velocity. A chaotic mapping is used to update the particles position.

3. Meta Optimisation

Meta optimisation refers to the application of one optimisation algorithm to the parameters of another optimisation algorithm that is optimising a problem or set of problems. Since the performance of many optimisation algorithms is subject to their parameters being correctly selected, meta optimisation provides a way of selecting good parameters that let the sub optimisation algorithm operate effectively. The pseudo-code in Algorithm 2 shows how the meta optimisation algorithm works when implemented with PSO. The meta optimisation algorithm generates a number of particles with random positions, i.e. parameter sets for the sub optimisation algorithm. Each meta particle will evaluate and update its position (sub swarm parameter set) for a predefined number of iterations. Each position is evaluated by averaging its cumulative performance over a number of problems for a number of runs. Each problem is optimised by creating a new PSO process and running it for a predetermined number of iterations. Ultimately the meta PSO optimiser will converge on what is hoped to be the optimum parameter set for the sub PSO optimiser.

```
Create & randomly initialise M meta particles
while metaIteration mI <metaImax do
    for metaParticle = 1 to M do
        for run = 1 to Rmax do
            for problem = 1 to Pmax do
                Create & randomly initialise N sub particles
                while Iteration t <Imax do
                    for Particle = 1 to N do
                        Update personal best position
                        Update neighbourhood best position
                        Evaluate particle's current position
                        Update particle's velocity
                        Update particle's position
                    end
                end
                Update cumulativeFitness
            end
        end
        Average cumulativeFitness over Rmax runs Update personal best meta position
        Update neighbourhood meta best position
        Evaluate meta particle's current position
        Update meta particle's velocity
        Update meta particle's position
    end
end
Return best parameter set
```

Algorithm 2: Meta Optimisation Algorithm

3.1. Application of Meta Optimisation

In this research, the meta optimisation process is represented in Figure 1. Each of the velocity update equations in Table 1 will be inserted into the sub PSO optimiser. The argument exists that by implementing meta optimisation, one will then need to optimise the parameters of the meta optimisation algorithm using an additional optimisation algorithm which will then need to be optimised, and so on. To circumvent this issue the meta optimisation algorithm implemented in this research is the constricted PSO (Equation 1). The parameters for the constricted PSO variant are guaranteed to converge. This does not guarantee the best performance of the meta optimiser but it does mitigate the issue of selecting the meta optimiser's parameters. Regardless of what meta optimisation algorithm is used, the issue of parameter selection will occur. By selecting the constricted PSO, at least it is known that the meta optimiser will converge and provide good performance.

Equation 1 is shown to have stable convergence with values $\chi \approx 0.72984$ & $c_1 = c_2 = 2.05$ and will not require parameter tuning. PSO was also selected as a meta optimiser because the problem of parameter selection is not a convex optimisation problem, therefore traditional gradient decent methods are not suited to it. The experimental parameters that will define each simulation are as follows: The meta optimising PSO will consist of 5 particles over

100 iterations. This equates to the evaluation of 500 combinations of parameters which should be sufficient given that the most parameters that need to be optimised by any velocity update equation is 6. The sub optimisation PSO will consist of 50 particles evaluating each problem for 3000 iterations. A larger number of particles and iterations is needed here as the problem spaces are much larger, typically 30 dimensions. The cumulative fitness achieved over all 8 problems is averaged over 10 statistical runs to ensure that any set of PSO parameters didn't perform well/poorly by chance.

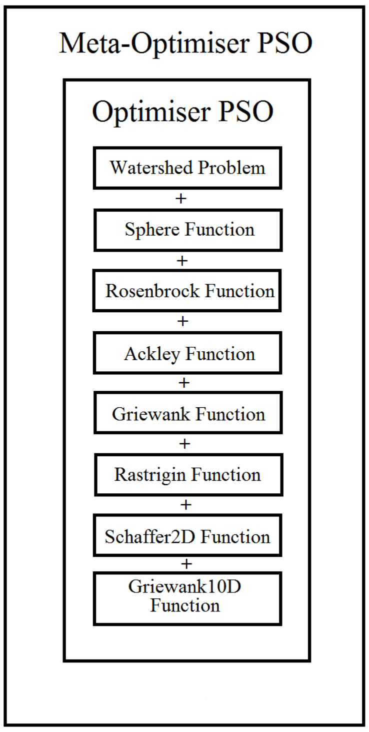


Figure 1: Meta Optimisation Process. The overlaying PSO meta optimiser is used to find the optimal parameters for the sub optimisation PSO. The parameters of each velocity update equation of the sub optimiser PSO will be optimised for 8 problems

3.2. Problem Definition

Table 2 outlines each of the functions that will be used to test the performance of each velocity update equation, aside from the watershed problem which will be outlined later on. The performance of each algorithm is judged by a linear combination of its performance on each of the problems. Since the Watershed Management problem is a maximisation problem rather than a minimisation problem, each of the functions in Table 2 have been converted to maximisation problems.

4. Machine Learning

In Machine Learning research, algorithms can be divided into either online or offline learning [23]. Online learning algorithms refer to algorithms that continuously learn in their environment. In offline learning, there is a clear distinction between the training phase and the testing phase. The training phase consists of a fixed set of historic data, known as training data. This data is used for the offline learning algorithm to approximate some function. Once the initial training period is complete, the algorithm is finished learning. The testing period consists of introducing the algorithm to previously unseen data, completely separate from the training data, and evaluating its performance. In online learning, there is no distinction between training and test periods/data. This research will focus on offline learning, namely an Artificial Neural Network (ANN) trained with PSO.

Table 2: Benchmark Maximisation Functions

Function Name	Equation
Sphere	$f(x) = -\sum_{i=1}^{d} x_i^2$
Rosenbrock	$f(x) = -\sum_{i=1}^{d-1}[100(x_{i_1} - x_i^2)^2 + (x_i - 1)^2]$
Ackley	$f(x) = 20exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) + exp\left(\frac{1}{d}\sum_{i=1}^{d} cos(cx_i)\right) - 20 - exp(1)$
Griewank	$f(x) = -\sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Rastrigin	$f(x) = -10d - \sum_{i=1}^{d}[x_i^2 - 10cos(2\pi x_i)]$
Schaffer	$f(x) = -0.5 - \frac{sin^2(x_1^2 - x_2^2) - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2}$

4.1. *Artificial Neural Networks*

Artificial Neural Networks (ANN), is a field of study within Machine Learning that is inspired by the brain [7, 15]. ANNs are have been applied to a range of areas such as classification, regression, control, online and offline learning, robotics and function approximation. ANN consist of an input layer of units (commonly referred to as neurons), one or more hidden layer of neurons and an output layer. The network receives information in the form of a signal into the input layer. This signal is carried through the connected layers of neurons via weighted connections. The network then outputs the signal through the output layer. The most commonly used method used to train the network weights is the backpropagation method [16]. This method requires a set of labelled data and is therefore not suitable for this application as the optimum network outputs for each flow scenario are not known. This provides the motivation for this research and justifies the fifth contribution outlined in the introduction. Unsupervised learning only consists of unlabelled input data where the targets are not known. This research consists of taking a set of unlabelled input data, in the form of flow scenarios for the Watershed Management problem, and use PSO to train the ANN in an unsupervised offline manner.

Information is fed into the network through the input layer of neurons (or units). This signal is then passed through the neurons in the subsequent hidden layers of the network until it is outputted from the final output layer. This process is commonly referred to as a forward pass. Figure 2 shows the structure of a multi-layer ANN. By conducting parameter sweeps, it was found that a network configuration of 1 hidden layer with 4 neurons provided the good performance while still keeping the computation time relatively low. This configuration consists of 28 network weights which must be optimised. It is possible to implement PSO in a manner to also evolve the configuration of the neural network. This is however out of the scope of this research. The primary concern when applying meta optimisation to the PSO in this paper is to find the optimum parameters of each velocity update equation. By also applying the meta optimiser to the neural network configuration, one would run the risk of failing to identify good velocity equation parameters due to poor network configuration. Therefore it is thought to be advantageous in this instance to keep the network configuration constant.

As the signal is propagated through the network, its strength is weighted as it passed between each layer of neurons. A neuron in any given layer will have as input, the sum of the weighted outputs from the previous layer of neurons.

$$v_j = \sum_{i=1}^{N} w_{i,j} a_i \tag{3}$$

Where v_j is the input to a neuron in the j^{th} layer, layer i is the preceding layer to j that contains N neurons, each neuron in layer i has output a_i and each of these output signals are weighted by the value $w_{i,j}$ as they are passed to each neuron in layer j.

Each neuron a_i outputs a value between 0 and 1. This output value is determined by the activation function of the neuron. The most commonly used action function is the sigmoid (or logistic) function:

$$a_j = \frac{1}{1 + \exp -v_j} \tag{4}$$

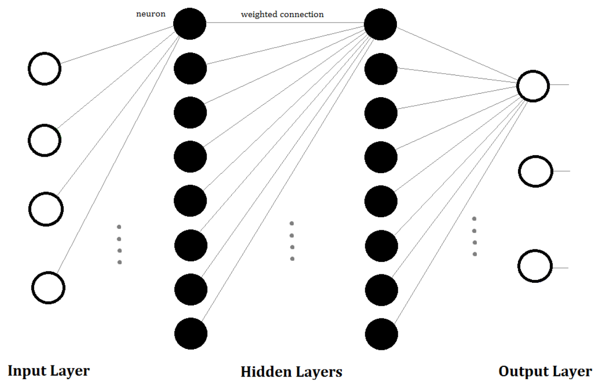


Figure 2: Multi-Layer Artificial Neural Network

4.2. Related Work

Training neural networks can also be viewed as optimising a set of weights to minimise a cost function. There are many methods used to do this. As previously mentioned, the gradient descent and back propagation algorithms are the most commonly used methods [16]. Reinforcement Learning (RL) algorithms commonly use neural networks to handle continuous problem spaces [8], e.g. Temporal Difference learning [42]. Evolutionary computing has also been used to train neural networks. This is known as Neuroevolution [13]. Neuroevolution methods incorporate some form of evolutionary algorithm, e.g. the NEAT algorithm which evolves neural networks using genetic algorithms [41]. Recent work has also applied Differential Evolution to the task of neural network weight optimisation [26, 11, 24] Particle Swarm Optimisation has also been applied to train neural networks for classification tasks [35] and control problems [19]. This research builds on these previous methods by applying them to the Watershed Management problem.

4.3. Application to Watershed Management

In the case of training a neural network function approximator for the Watershed Management problem, a particles position will correspond to the network weights. The 3 network inputs will correspond to the current state of the river $[q_1, q_2, s]$. The 4 outputs of the network correspond to the 4 directly controlled variables of the Watershed problem, i.e. the 4 interested parties seeking to withdraw water. The fitness of the current network weights configuration (or particle position) is calculated as the sum of the individual fitness evaluations for each training state over all training states $[Q_1, Q_2, S]$. The fitness of each state is calculated using Equation 11. Each PSO particle will then use the fitness over all states as a means to determine how good the current set of network weights is.

5. Watershed Problem

As mentioned in the introduction section, the Watershed problem is a resource management problem which consists of multiple interested parties. Each of these parties are withdrawing water from a common and finite supply for their own purposes. This problem has multiple objectives and constraints, and consists of continuous variables. However this problem will be treated as a single objective problem as only the overall system performance is of interest. There are 6 variables which must be optimised, 4 of which are controlled directly, the remaining 2 are reactive variables which are indirectly optimised. The 4 direct variables are the water withdrawn from the river for municipal and industrial use in the city (x_1), the water withdrawn for the irrigation of farms (x_4 and x_6) and finally the water released from a dam for hydro power generation (x_2). The 2 reactive variables are the water available for the ecosystems (x_3 and x_5). Each of these variables must be selected to maximise a series of objective functions representing the benefits obtained from the water by the various interested parties.

The benefit of the water withdrawn for each interested party is represented by the following function:

$$f_i(x_i) = a_i x_i^2 + b_i x_i + c_i \tag{5}$$

Where a_i, b_i and c_i are dimensionless constants corresponding to each party [45]. Their values are highlighted in the following table, along with the values for α_i which represents the minimum values for x_i in L^3, where i represents each objective.

Table 3: Watershed Constants

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value (L^3)
a_1	−0.20	b_1	6	c_1	−5	α_1	12
a_2	−0.06	b_2	2.5	c_2	0	α_2	10
a_3	−0.29	b_3	6.28	c_3	−3	α_3	8
a_4	−0.13	b_4	6	c_4	−6	α_4	6
a_5	−0.056	b_5	3.74	c_5	−23	α_5	15
a_6	−0.15	b_6	7.6	c_6	−15	α_6	10

Since the indirect variables, x_3 and x_5, are not directly controlled, they must be calculated using the following equations:

$$x_3 = Q_2 - x_4 \tag{6a}$$
$$x_5 = x_2 + x_3 - x_6 \tag{6b}$$

Where Q_2 is the monthly tributary inflow in L^3. The values for Q_2 will be outlined later. The problem variables x_i are is subject to the following constraints which restrict their potential values.

$$\alpha_1 - x_1 \leq 0 \tag{7a}$$
$$\alpha_2 - Q_1 + x_1 \leq 0 \tag{7b}$$
$$x_2 - S - Q_1 + x_1 \leq 0 \tag{7c}$$
$$\alpha_4 - x_3 \leq 0 \tag{7d}$$
$$\alpha_3 - x_4 \leq 0 \tag{7e}$$
$$\alpha_4 - Q_2 + x_4 \leq 0 \tag{7f}$$
$$\alpha_6 - x_5 \leq 0 \tag{7g}$$
$$\alpha_5 - x_6 \leq 0 \tag{7h}$$
$$\alpha_6 - x_2 - x_3 + x_6 \leq 0 \tag{7i}$$

Where S represents the storage capacity of the dam in L^3 while both Q_1 and Q_2 represents the monthly inflows of water into the mainstream and tributary in L^3. The Watershed problem evaluated in this paper will consist of a total of 150 flow states for the river. This is divided up into $\frac{2}{3}$ training data (100 states) and $\frac{1}{3}$ testing data (50 states). The values for Q_1, Q_2 and S can be found in the appendix.

The Watershed problem can be formulated as optimising the following equation subject to the constraints highlighted in Equation 7.

$$F(x) = max \sum_{i=1}^{6} f_i(x_i) \tag{8}$$

The Watershed problem is graphically illustrated in Figure 3 from the research of Yang et al. [45].

5.1. Boundary Definition

In order to address the Wateshed Management problem, the maximum and minimum possible values for each direct variable must first be explicitly defined. The ranges for each of the directly optimised variables are defined as follows based on Equation 7:

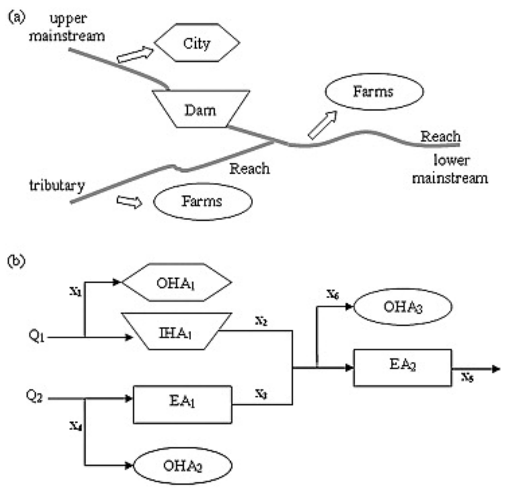


Figure 3: Watershed Illustration [45]

$$\alpha_1 \leq x_1 \leq Q_1 - \alpha_2 \tag{9a}$$

$$\alpha_3 \leq x_4 \leq Q_2 - \alpha_4 \tag{9b}$$

$$0 \leq x_2 \leq S + Q_1 - \alpha_1 \tag{9c}$$

$$\alpha_5 \leq x_6 \leq S + Q_1 + Q_2 - \alpha_1 - \alpha_3 - \alpha_6 \tag{9d}$$

5.2. *Constraint Handling*

Any violations of the constraints highlighted in Equation 7 will be handled using the static penalty method [40]. This penalty function will be incorporated into the fitness of any given solution. The penalty function is defined below:

$$f_p = \sum_{i=1}^{N} C(|h_i + 1|\delta_i) \tag{10}$$

Where $N = 9$ is the total number of constraints handled using this method per flow scenario, $C = 10E2$ is the violation constant, h_i is the violation amount of each constraint and $\delta = 0$ if there is no violation for a particular constraint and $\delta = 1$ if a constraint is violated. The violation constant $C = 10E2$ was selected so that any solution which violates a constraint will have a significantly lower fitness than acceptable solutions. Lower C values were evaluated but caused each algorithm to converge on infeasible solutions.

5.3. *Fitness Function*

In order to gauge the suitability of a given solution, a fitness function must first be defined. As previously mentioned, the fitness function will be based on the overall performance of the system based on the objective function in Equation 8 and the penalty function from Equation 10. These two equations are combined to give the following fitness function which is to be maximised:

$$F = \sum_{i=1}^{6} f_i(x_i) - \sum_{j=1}^{N} C(|h_j + 1|\delta_j) \tag{11}$$

6. Experimental Results

In this section, the results of each of the experiments will be presented and discussed. The performance of each velocity update equation will be gauged based on its convergence speed, best fitness and consistency. These will be judged for both the cumulative fitness over all 8 problems and for each problem individually. Any performance comparisons between two algorithms will be done using the two tailed t-test with a significance level of 5%. This will ensure that any differences in performance are statistically significant.

6.1. Constriction vs Inertia

Table 5 and Figure 4 shows the cumulative fitness convergence for the constricted, linear decreasing inertia and adaptive inertia velocity update equations. These results clearly show that the linear decreasing inertia provides better performance than both constriction and adaptive inertia weight, constriction being the worst of the three. This gives a clear answer to the third research question: stable convergence does not guarantee optimum performance and a linearly decreasing inertia outperforms a constricted PSO.

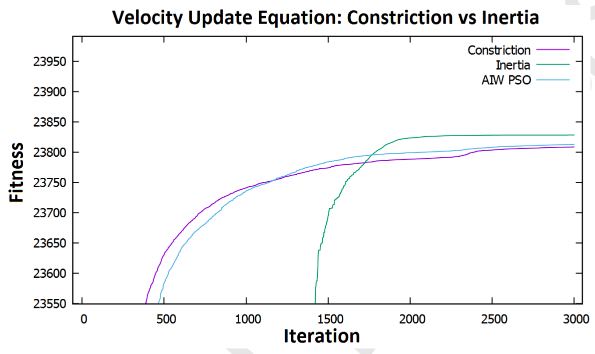


Figure 4: Convergence of Velocity Update Equations using Constriction & Inertia This graph illustrates the average cumulative fitness convergence of the standard constricted PSO, PSO with linear inertia and adaptive inertia PSO over all 8 problems.

6.2. Avoidance Strategies

Since the results from the previous section show that a linearly decreasing inertia offers superior performance, two additional modified versions of the PSOA and PSOAWL were also tested to include a linearly decreasing inertia. These velocity equations can be seen in Table 4, numbered velocity equations 14 & 15. The parameter sets to be optimised for each of the PSOA2 and PSOAWL2 are $[c_1, c_2, \omega_{start}, \omega_{end}]$ and $[c_1, c_2, c_3, \omega_{start}, \omega_{end}]$ respectively. These two velocity update equations were included to ensure that a fair comparison was made amongst each of the velocity update equations.

The performance results of each of the avoidance velocity update equations can be seen in Table 5 and Figure 5. The two best performing velocity equations here are the PSOAWL2 and the NPSO2. Each of these variants converge to almost the same solution fitness. The PSOAWL2 performs statistically better than the NPSO2. The third best performing algorithm is the PSOAWL with constriction followed by the NPSO1. Both of these algorithms converge much faster than the PSOAWL2 and NPSO1, however they converge on a significantly worst solution. The PSOAWL2 implemented with a linear decreasing inertia performs much better than the PSOAWL implemented with a constriction. This reaffirms the results from the previous section where the linear decreasing inertia outperformed the constriction value. Both variants of the PSOA perform worst. This was expected as each of these variants only avoid the worst locations and do not converge on the best locations. Each of these velocity update equations produce solutions far worse than those previously mentioned and therefore do not appear in Figure 5. These velocity update equations were included for comparative purposes.

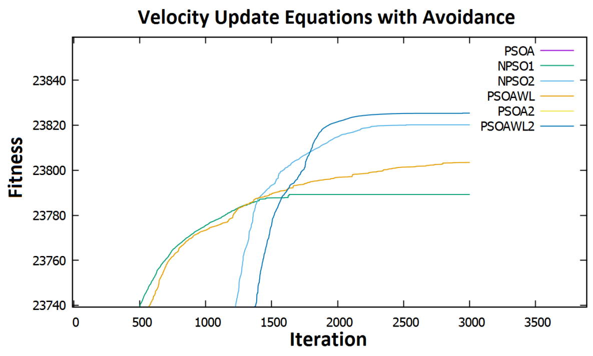


Figure 5: Convergence of Avoidance Velocity Update Equations This graph illustrates the average cumulative fitness convergence of the 6 PSO variants that utilize avoidance strategies over all 8 problems.

6.3. *Non-Standard Velocity Update Equations*

As illustrated in Figure 6 and highlighted in Table 5, ARPSO2 and ARPSO1 are the 2 best performing velocity update equations. These are followed closely by the DPSO. The difference between these two variants of the ARPSO is the parameters being optimised. The ARPSO1 uses the default values for the critical upper and lower diversity levels, $d_{low} = 5 \times 10^{-6}$ and $d_{high} = 0.25$ [36]. The meta optimiser for the ARPSO2 found the optimum values to be $d_{low} = 0.0570$ and $d_{high} = 0.0123$ (rounded to 4 decimal places). Interestingly the meta optimiser actually found the optimum d_{high} value to be lower than the optimum d_{low} value. This would lead to the situation where the parameter dir is alternating between -1 and 1 at each iteration. This will be discussed later in more detail in Section 6.6. The ARPSO1 offers faster convergence than the ARPSO2, however the ARPSO2 converges to a better solution and has a smaller standard deviation meaning that it is more consistent. MOL is the next best performing algorithm which converges quickly but converges to a sub optimal solution.

The second worst performing velocity update equation is the AVPSO. This is an interesting velocity update equation however as this variant has only one parameter that needs to be optimised. The AVPSO is also very inconsistent with a high standard deviation. The worst performing of the non-standard velocity update equations is the PSOCV. This variant does not provide fast convergence as seen in Figure 6. The velocity control in this equation is similar in philosophy to the ARPSO but performs statistically worse. The advantage of implementing the PSOCV over the ARPSO however is that the PSOCV does not require calculating the swarm diversity at each iteration. This variant converges very slowly to a poor solution.

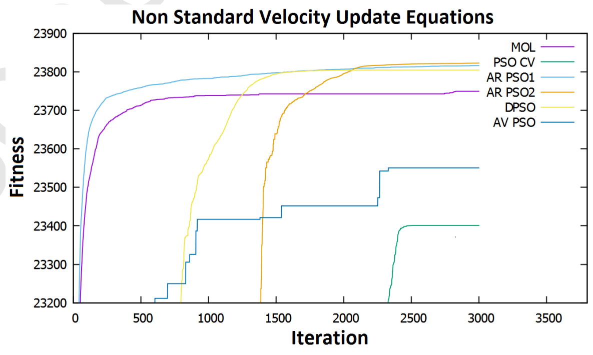


Figure 6: Convergence of Non-Standard Velocity Update Equations This graph illustrates the average cumulative fitness convergence of the 6 non standard PSO velocity equations over all 8 problems.

6.4. *Hybrid Velocity Update Equations*

In this section, a number of hybrid velocity update equations will be proposed based on the performance of the algorithms in the previous sections. Since essentially all of the algorithms are variations of the standard velocity update equation, the hybrid velocity update equations proposed here are combinations of velocity equations from the categories of avoidance and non-standard velocity equations. A linearly decreasing inertia was implemented in each of these hybrid velocity update equations as it is the best performing of the standard velocity update equations.

As a result of this, hybrid velocity update equations were created based on the PSOAWL2, NPSO2, ARPSO2 and DPSO as these are the top performing non-standard velocity equations. Table 4 displays the 5 hybrid velocity update equations which were evaluated, numbered 16 - 20.

Table 4: Modified Velocity Update Equations

Algorithm	Velocity Equation	Parameters
14) PSOA2	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{x}_t - \vec{pw}_t) + r_2c_2(\vec{x}_t - \vec{gw}_t)$	$c_1, c_2, \omega_{start}, \omega_{end}$
15) PSOAWL2	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{pb} - \vec{x}) + r_2c_2(\vec{gb} - \vec{x}) + r_3c_3(\frac{r_1c_1(\vec{pb}-\vec{x})}{1+\lvert\vec{x}-\vec{pw}\rvert})$	$c_1, c_2, c_3, \omega_{start}, \omega_{end}$
16) AR-PSOAWL	$\vec{v}_{t+1} = \omega\vec{v}_t + dir(r_1c_1(\vec{pb} - \vec{x}) + r_2c_2(\vec{gb} - \vec{x}) + r_3c_3(\frac{r_1c_1(\vec{pb}-\vec{x})}{1+\lvert\vec{x}-\vec{pw}\rvert}))$	$c_1, c_2, c_3, \omega_{start}, \omega_{end}, d_{low}, d_{high}$
17) D-PSOAWL	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{pb} - \vec{x}) + r_2c_2(\vec{gb} - \vec{x}) + r_3c_3(\frac{r_1c_1(\vec{pb}-\vec{x})}{1+\lvert\vec{x}-\vec{pw}\rvert})$	$c_1, c_2, c_3, \omega_{start}, \omega_{end}, c_v, c_l$
18) AR-NPSO	$\vec{v}_{t+1} = \omega\vec{v}_t + dir(r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t) + r_3c_3(\vec{x}_t - \vec{pw}_t) + r_4c_4(\vec{x}_t - \vec{gw}_t))$	$c_1, c_2, c_3, c_4, \omega_{start}, \omega_{end}, d_{low}, d_{high}$
19) D-NPSO	$\vec{v}_{t+1} = \omega\vec{v}_t + r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t) + r_3c_3(\vec{x}_t - \vec{pw}_t) + r_4c_4(\vec{x}_t - \vec{gw}_t)$	$c_1, c_2, c_3, c_4, \omega_{start}, \omega_{end}, c_v, c_l$
20) AR-DPSO	$\vec{v}_{t+1} = \omega\vec{v}_t + dir(r_1c_1(\vec{pb}_t - \vec{x}_t) + r_2c_2(\vec{gb}_t - \vec{x}_t)))$	$c_1, c_2, \omega_{start}, \omega_{end}, d_{low}, d_{high}, c_v, c_l$

The results of these hybrid velocity equations are displayed in Table 5 and in Figure 7. The AR PSOAWL is the best performing hybrid velocity update equation, followed by the D PSOAWL. The AR PSOAWL performs statistically better than the D PSOAWL and also has a very small standard deviation meaning that its performance was consistent across the 10 statistical runs. This is a highly desirable feature of an optimisation algorithm. This velocity update also converges very quickly, converging to a nearly optimum solution after 1000 iterations. The reason for this fast convergence to a near optimum solution is thought to be due to the enhanced exploration provided by the attractive/repulsive part of the velocity update equation and the enhanced convergence provided by avoiding the worst locations. The best 2 hybrid velocity update equations were variants of the PSO AWL while the worst and 3rd worst were variants of the NPSO 2.

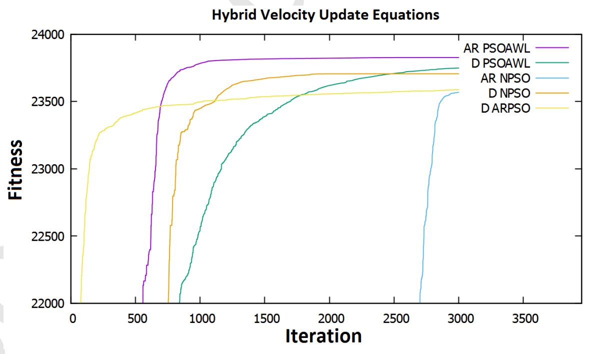


Figure 7: Convergence of Hybrid Velocity Update Equations This graph illustrates the average cumulative fitness convergence of the 5 hybrid PSO velocity update equations over all 8 problems.

6.5. *Overall Comparison*

The convergence graphs in Figure 8 displays the convergence of the best 8 performing velocity update equations. The best two performing algorithms are linearly changing inertia and the AR PSOAWL. When compared the two tailed t-test these two algorithms produce solutions that are statistically equal.

There are advantages and disadvantages with each velocity update equation. The linearly changing inertia PSO is computationally cheaper and simpler to implement. This would make the linearly decreasing inertia PSO ideal for some problems where limited computational resources are available. Conversely the AR PSOAWL converges much faster. This makes the algorithm much more suitable for problems where a solution is needed rapidly.

It is also worth noting the performance of the Linear Decreasing Inertia PSO with MOL. MOL was originally proposed as a simplified version of the standard PSO and it was found to perform better than the standard PSO with a static inertia and therefor was claimed to be better than the standard PSO [35]. The research conducted in this paper demonstrates that this is not necessarily the case. MOL is the 12th best performing algorithm and is statistically outperformed by more complex velocity update equations. However it is among the least computationally expensive velocity update equations of those evaluated.

Each of the velocity update equations evaluated here were reported to be superior to the standard linearly decreasing inertia. This research demonstrates that this is not strictly true. This research has confirms that negative inertia values can provide increased performance as previously established [35]. This research has also established that a linear increasing inertia can provide superior performance. However when each of these velocity update equations were compared to the standard linear changing inertia, negative or increasing inertia values were never considered. This is because without the use of meta optimisation, a practitioner implementing PSO would never consider to implement such a velocity update equation. It was found that the PSOAWL2 and AR PSOAWL perform statistically equal to the linear changing inertia. Since the linear changing inertia is much simpler to implement and requires less computational time, this basic velocity update equation would still be the preferred choice. However the AR PSOAWL does converge faster, which would make this algorithm the obvious choice if a solution is needed in fewer iterations.

There is also the possibility that the meta optimiser has not fully converged on the optimum parameters for the more complex velocity update equations. The linear changing inertia has 4 parameters that must be optimised while the AR NPSO has 8 parameters. The meta optimiser was implemented with same number of iterations (50) and particles (10) for each velocity update equation regardless of the number of parameters. It was not possible to run the meta optimiser for more iterations due to the large computational cost required to do so. In the future, it will be possible to conduct larger meta optimisation analysis as computers become faster. Running larger experiments now would require more computational resources than is available. The results presented in this paper required 12 days to produce when implemented in parallel using a separate thread for each velocity update equation. 4 threads were running at any one time. This was simply due to the scale of the experiments conducted in the research rather than the efficiency of the PSO algorithm. To evaluate 20 variants of any optimisation algorithm over 8 problem spaces using meta optimisation will require a considerable amount of time.

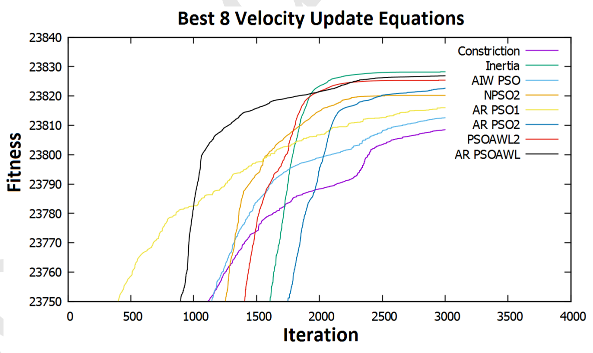


Figure 8: Convergence of Best 8 Velocity Update Equations This graph illustrates the average cumulative fitness convergence of the 8 best performing PSO velocity update equations over all 8 problems.

6.6. Optimum PSO Parameters

The parameter boundaries for the meta optimiser are as follows:

$\chi \in [-2, 2]$, $\omega_{start,end} \in [-2, 2]$, $c_{1,2,3,4} \in [-4, 4]$, $d_{high,low} \in [0, 1]$, $c_{v,l} \in [0, 1]$, $n \in [-10, 10]$,

These reflect the maximum and minimum values that the parameters outlined in Tables 1 and 4 can obtain. It is possible to apply meta optimisation to other PSO parameters outside of the velocity update equation, such as the swarm size and number of iterations. This was not implemented in this research however for the same reasons previously stated for not applying meta optimisation to the neural network configuration. The goal is to keep everything else consistent other than the velocity update equation. The optimum parameters found by the meta optimiser for each velocity update equation can be seen in Table 5. All of the values in this table are rounded to four decimal places. It should be noted that velocity clamping was implemented for each simulation. This was to ensure that the velocity of the particle did not exceed the length of the problem in each dimension.

Table 5: Optimum Parameter Settings & Fitness Results

Velocity Equation	Average	StdDev	c_1	c_2	c_3	c_4	χ	w_{start}	w_{end}	d_{low}	d_{high}	c_v	c_l	n
Constriction	23808.5182	6.7336	0.4790	3.9911	-	-	0.6952	-	-	-	-	-	-	-
Linear Inertia	23828.2486	5.6245	1.3518	2.3349	-	-	-	-1.6980	1.0446	-	-	-	-	-
AIWPSO	23812.5791	8.4896	0.1419	3.5428	-	-	-	-0.2897	0.5107	-	-	-	-	-
PSOA	16938.1398	1631.0666	0.0015	-0.0850	-	-	-	-	-	-	-	-	-	-
NPSO1	23789.2629	9.7344	2.8440	0.0016	0.7030	-	-	0.5862	-1.0542	-	-	-	-	-
NPSO2	23820.1917	4.0547	1.4219	0.9868	-0.0001	0.7510	-	-1.2431	1.2678	-	-	-	-	-
PSOAWL	23803.4837	9.1318	1.4909	2.7956	-0.3557	2.8016	0.4629	-	-	-	-	-	-	-
PSOA2	19239.9814	1938.4064	-0.6689	-3.1511	-	-	-	1.1945	-1.8163	-	-	-	-	-
PSOAWL2	23825.3919	3.6048	1.6220	2.4593	-2.4723	-	-	-1.4442	0.8631	-	-	-	-	-
MOL	23749.1889	47.6048	3.1569		-	-	-	0.0260	-1.2086	-	-	-	-	-
PSOCV	23401.1286	40.9073	1.0876	3.1321	-0.0017	-	-	0.1860	-1.5133	-	-	-	-	-
ARPSO	23816.0095	10.3410	1.6283	1.8452	-	-	-	0.4836	-0.1581	-	-	-	-	-
ARPSO2	23822.6522	5.4883	0.8250	1.6549	-	-	-	-1.2192	0.0210	0.0570	0.0123	-	-	-
DPSO	23804.4577	6.8192	0.9839	2.1932	-	-	-	-1.2385	1.5159	-	-	0.0013	0.0052	-
AVPSO	23550.5377	177.4378	-	-	-	-	-	-	-	-	-	-	-	5.4255
AR PSOAWL	23826.8770	4.4922	0.6635	0.9234	-0.2334	-	-	-1.1235	0.5260	0.1610	0.0087	-	-	-
D PSOAWL	23748.5109	13.8354	1.4513	1.6637	-1.4938	-	-	1.0247	-0.0040	-	-	0.0092	0.0124	-
AR NPSO	23567.5277	21.1172	0.6697	3.3012	-0.0376	-1.4002	-	0.2529	-1.2638	0.2390	0.8096	-	-	-
D NPSO	23705.5609	29.7061	1.6686	0.7249	0.0061	0.2196	-	-1.5168	1.6145	-	-	0.0033	0.0031	-
AR DPSO	23587.1871	20.0939	1.0394	0.6239	-	-	-	0.1151	0.0386	0.7689	0.4601	0.0056	0.0266	-

As seen in the results of other studies [35], the optimum parameter settings found by the meta optimiser are drastically different from those commonly used in the PSO literature. The optimum parameters found for the constricted PSO are very far from those proven to provide stable convergence. The meta optimiser found a $\chi = 0.6952$ with $c_1 = 0.4790, c_2 = 3.9911$ gave the best performance. It was mathematically proven that $\chi = 0.72984$ with $c_1 = c_2 = 2.05$ provide stable convergence [10]. The meta optimiser did not find these values to give the best performance.

There are two interesting observations with regards to the optimum inertia values found. The first observation is that in many cases, the meta optimiser found that a negative inertia value leads to better performance for many of the velocity update equations evaluated. It is very interesting that the meta optimiser found a negative ω to be optimal. The vast majority of research into PSO does not consider implementing a negative ω. This would have the effect of alternating the direction of the particles' velocity between iteration, possibly leading to greater exploration. For every velocity update equation implemented with an inertia term, the meta optimiser found that an $\omega < 0$ at some stage of the optimisation process gives the best performance. This would be a potential avenue of future research to further investigate why this is the case.

The second observation is that in many cases, a linearly increasing inertia value was found to be optimal. It would therefore be more accurate to refer to the inertia term as a linearly changing inertia rather than a linearly decreasing inertia. In the velocity update equations where an increasing inertia was found to be optimal, typically the optimum found initial inertia value is $\omega < -1$. This would cause the particles to initially move with a lot of velocity changing direction at each iteration, causing the particles to explore the problem space. These inertia terms then always increase to final values where $\omega > 0$, in some cases where $\omega > 1$. In situations where $1 > \omega_{end} > 0$, the particles will converge on the best known location as is normal with PSO. This is the case with the AIWPSO, PSOAWL2, ARPSO2 and AR PSOAWL. For other velocity update equations where $\omega_{end} > 1$ this is not the case. It is likely that the particles will have converged to some solution while ω changes between 0 and 1. However when ω increases above 1 again, the particles would receive more velocity and begin exploring again. It is thought that this would aid the performance of the PSO algorithm by enabling the particles to perform a local search after they have converged. This is the case for the standard linear changing inertia PSO, NPSO2, DPSO and DNPSO.

In many cases it is still found that a linearly decreasing inertia is still optimal. This is the case for the NPSO1, PSOA2, MOL, PSOCV, ARPSO, D PSOAWL and AR NPSO. In many of these velocity update equations $\omega_{end} < -1$. This would serve the purpose of performing a local search after the particles have converged.

It is also worth noting that each of these parameter sets highlighted in Table 5 have small standard deviations relative to their average fitness. This indicates that they give consistent performance. This is not the case for the PSOA or PSOA2 as this velocity update equation was included for comparative purposes and does not perform well.

The meta optimiser found interesting upper and lower diversity thresholds for the ARPSO. In its first proposal, the ARPSO found that $d_{low} = 5 \times 10^{-6}$ and $d_{high} = 0.25$ gave the best performance [36]. Table 5 however shows that the optimal diversity thresholds are very different from these values. Most notably that the $d_{low} > d_{high}$. The optimal values found for the ARPSO2 are $d_{low} = 0.0570$ and $d_{high} = 0.0123$. As the particles converge, when the diversity gets to a point between these two values, the value for dir would alternate between -1 and 1 until the current diversity moves out of this range. This could happen if the inertia was a small enough value. These values for the diversity thresholds are not what the authors had in mind in the original proposal of the ARPSO. The meta optimiser however found that these values produce a better solution and are more consistent. However these two variants perform statistically the same due to the relatively large standard deviation of the ARPSO.

The optimum c_v and c_l terms for the DPSO were found to be very small. This is unsurprising as giving the particles a random velocity and position is intended to increase their exploration. This is only supposed to happen a small portion of the time because the algorithm still needs to converge. Table 5 shows that for each variant of the DPSO, the assignment of a random velocity or position occurs less than 1% of the time.

The optimum values for each of the acceleration coefficients (c) vary for each velocity update equation. For each of the standard velocity update equations, a stronger influence from the global best position provides the best performance. Typical $c \approx 2$ is used in the literature. In many cases these c values were far larger or smaller than 2. In all cases $c > 0$ for the global and personal best position portions of the velocity update equations. In some velocity update equations where worst locations influence the motion of the particles, the c values relating to the velocity received from the worst locations are negative. It is thought that the reason for this is to aid exploration. With regards to the PSO CV, the c_3 term which relates to the additional velocity control term is almost 0. This indicates that the additional control velocity term should not have a strong influence on the particles' motion. The results presented in Table 5 clearly show that this small influence from the velocity control term has a very adverse effect on the convergence of the particles when compared to the standard linearly changing inertia PSO. The velocity control term ($\frac{c_3 r}{v^2}$) within the PSO CV velocity update equation could be an avenue for future research. When v^2 is large and the particles are exploring, the algorithm would operate as expected as the velocity control term would be very small and insignificant. However as v^2 approaches 0, the $c_3 r$ term becomes much more significant. The problem with this is that the value of c_3 then needs to be tailored for each problem because the velocity control term does not possess any terms that can scale the additional velocity to the current problem being optimised. This lack of generalisation is a fundamental problem with this velocity update equation in its current form and would be worth investigating.

In summary, it is well known that the performance of the PSO algorithm is heavily reliant on the parameters selected. This is particularly true for the more complex hybrid velocity update equations. This results section has presented some very important results: 1) Negative inertia terms should be considered when implementing a PSO algorithm. 2) An increasing inertia should be considered as well as a decreasing inertia term. 3) The diversity thresholds of the ARPSO should be the subject of further research. This research demonstrates that very unorthodox diversity thresholds can perform well. 4) With regards to the constricted PSO, stable convergence does not equate to optimum performance.

6.7. Individual Problem Performance

The performance results of each velocity update equation on each individual problem are presented in Table 6. The most interesting observation from Table 6 is the performance of the AV PSO. When evaluating the performance of the AV PSO in Table 5, it performs 4th worst of all velocity update equations. The results presented in Table 5 are a linear combination of the fitness of each individual problem performances. This is also the fitness that the meta optimiser uses to select the optimum parameter set for each velocity update equation. The results presented in Table 6 however reveal that the AV PSO performs best on 6 out of the 8 problems evaluated. The AV PSO performs best on all of the benchmark problems except for the Rosenbrock function. The AR PSO performs best on the Rosenbrock function while the AR PSOAWL performs best for the Watershed ANN training. The linear changing inertia does not perform best on any one of the functions evaluated. The size of the Watershed training problem (28 weights) is very similar to many of the benchmark functions, most of which have 30 dimensions except for the Schaffer 2D and Griewank 10D functions.

The fact that the AV PSO performs best on 6 benchmark functions but performs 3rd worst on the Watershed training problem is indicative that the Watershed training problem requires an optimisation algorithm with different characteristics than the benchmark problems do. It is thought that the Watershed training problem requires much more exploration than the benchmark problems. The AV PSO performs best on the Sphere, Griewank 10D and Schaffer 2D. All of these problems are much simpler than the Watershed training problem. It would therefore be advantageous of an optimisation algorithm to converge quickly for these problems. The AV PSO is also unique when compared to the other velocity update equations tested as it only possess one parameter that must be optimised in its velocity update equation. This is clearly of benefit for the AV PSO when optimising simpler problems. This design simplicity is also an issue for more complex problems such as the Watershed training problem and the Rosenbrock function because the simple velocity update equation lacks the capability for its parameters to be selected for better exploration.

The impressive performance of the AV PSO on the benchmark functions demonstrates that a linear combination of each problem fitness should not be the only measure of performance used to compare algorithms. It should also be highlighted with regards to the Watershed neural network training, that a good fitness score for the training data does not equate to good performance on test data. This will be covered in more detail in the next section.

In terms of the computational cost of each velocity update equation, there was little variation in their computational requirements. In general the simpler velocity update equations were computationally cheaper, e.g. MOL, constricted PSO and PSOA. These variations in computational costs were minimal however and when compared to the significant differences in the average fitness results.

6.8. Watershed Test Data

This final results section will look at the performance of each neural network function approximator on independent test data. This will be done for each of the 20 velocity update equations. As previously stated, these neural networks were trained using the set of training data consisting of 100 training cases outlined in the appendix. After the initial training period, the neural networks were then tested on a completely separate set of test data consisting of 50 test cases, also outlined in the appendix. Table 7 displays the test data results of each neural network trained with a different velocity update equation. The fitness result is a cumulative fitness over all 50 test cases, referring to the cumulative utility of the water for all interested parties for 50 different river flow scenarios. The fitness values have been rounded to two decimal places.

The results in Table 7 illustrate a key issue in Machine Learning research, the problem of over training. When evaluated using the test data, the best 5 performing neural networks are those trained with:

1. PSOAWL2
2. Linear Changing Inertia PSO
3. AR PSO1
4. DPSO
5. AR PSOAWL

However Table 6 in Section 6.7 shows that the best 5 performing neural networks on the training data are:

1. AR PSOAWL
2. Linear Changing Inertia PSO
3. AR PSO2
4. PSOAWL2
5. NPSO2

The linearly changing inertia consistently performs 2nd best on both the training and test data. From an optimisation perspective the AR PSOAWL is the best performing algorithm for the Watershed problem as it performs best on the training data. From a Machine Learning perspective, the PSOAWL2 is the best performing algorithm as it performs best on the test data.

Table 6: Individual Problem Average Fitness & Standard Deviation.This table presents the average fitness and fitness standard deviation of each velocity update equation for all 8 problem spaces.

Velocity	Watershed		Sphere		Rosenbrock		Ackley		Griewank		Rastrigin		Schaffer2D		Griewank10D	
Equation	Fitness	Std Dev	Fitness	Std Dev	Fitness	Std Dev	Fitness	Std Dev	Fitness	Std Dev	Fitness	Std Dev	Fitness	Std Dev	Fitness	Std Dev
constriction	2.384E+04	4.908E+00	-3.194E-09	2.542E-09	-2.404E+01	3.380E+00	-3.853E-04	1.798E-04	-1.428E-02	1.609E-02	-1.070E+01	3.617E+00	-6.898E-04	4.516E-04	-7.996E-02	2.759E-02
linear inertia	2.385E+04	5.450E+00	-1.361E-16	1.629E-16	-2.363E+01	2.487E-01	-7.523E-08	9.282E-08	-5.173E-03	6.558E-03	-5.712E-01	7.573E-01	-6.907E-04	4.501E-04	-3.568E-02	2.531E-02
AIW PSO	2.384E+04	7.922E+00	-2.613E-06	2.592E-06	-2.283E+01	6.431E+00	-6.732E-03	3.280E-03	-3.562E-02	3.058E-02	-5.267E+00	2.806E+00	-9.854E-04	0.000E+00	-9.076E-02	3.589E-02
PSOA	1.947E+04	1.481E+03	-6.573E+01	8.611E+00	-1.860E+03	3.220E+02	-1.847E+01	3.659E-01	-2.437E+02	3.145E+01	-3.181E+02	1.296E+01	-1.006E-03	1.687E-04	-2.968E+01	5.897E+00
NPSO1	2.384E+04	8.425E+00	-4.790E-04	5.426E-05	-2.491E+01	1.941E+00	-1.379E-01	1.855E-02	-3.201E-01	6.402E-02	-2.148E+01	4.044E+00	-3.942E-04	4.827E-04	-1.364E-01	3.959E-02
NPSO2	2.385E+04	4.385E+00	-1.430E-05	2.214E-06	-2.678E+01	1.972E-01	-1.920E-01	5.279E-01	-1.892E-02	7.259E-03	-5.045E-01	6.696E-01	-2.309E-04	3.815E-04	-6.726E-02	7.683E-02
PSOAWL	2.384E+04	7.648E+00	-3.657E-11	2.809E-11	-2.464E+01	1.631E+00	-3.402E-05	2.732E-05	-7.738E-02	8.969E-02	-1.350E+01	6.270E+00	-8.869E-04	2.956E-04	-3.379E-01	2.043E-01
PSOA2	1.974E+04	1.927E+03	-7.369E+00	1.526E+00	-2.670E+02	4.613E+01	-1.044E+01	8.174E-01	-2.474E+01	4.123E+00	-1.864E+02	1.592E+01	-5.609E-04	4.219E-04	-5.893E+00	1.243E+00
PSOAWL2	2.385E+04	4.430E+00	-4.482E-15	1.327E-14	-2.257E+01	4.992E-01	-7.791E-06	5.243E-06	-2.583E-02	2.402E-02	-1.094E+00	1.805E+00	-8.869E-04	2.956E-04	-5.435E-02	3.088E-02
MOL	2.384E+04	6.044E+00	-2.343E-03	7.008E-03	-3.853E+01	2.577E+01	-9.630E+00	7.496E+00	-6.552E-02	4.114E-02	-4.200E+01	2.744E+01	-1.573E-03	1.176E-03	-1.611E-01	6.528E-02
PSO CV	2.361E+04	5.480E+01	-7.088E-01	1.774E-01	-3.833E+01	1.610E+00	-4.966E+00	1.883E-01	-1.121E+01	2.649E+00	-1.549E+02	2.654E+01	-8.370E-04	2.975E-04	-4.841E-01	9.524E-02
AR PSO	2.385E+04	5.246E+00	-2.365E-11	1.818E-11	**-2.124E+01**	1.616E+00	-2.288E-06	1.050E-06	-3.184E-02	2.682E-02	-8.358E+00	6.006E+00	-7.883E-04	3.942E-04	-7.428E-02	4.171E-02
AR PSO2	2.385E+04	5.654E+00	-1.156E-05	6.421E-06	-2.663E+01	2.493E-01	-2.473E-03	1.241E-03	-9.878E-04	2.959E-03	-2.024E-01	5.966E-01	-3.942E-04	4.827E-04	-4.894E-02	3.596E-02
DPSO	2.385E+04	5.440E+00	-5.590E-04	2.697E-04	-2.700E+01	2.987E-01	-1.430E-01	6.798E-02	-2.665E-01	1.561E-01	-1.425E+01	5.162E+00	-7.073E-04	4.272E-04	-6.272E-02	2.728E-02
AV PSO	2.358E+04	1.774E+02	**0.000E+00**	0.000E+00	-2.875E+01	7.473E-02	**-4.441E-16**	0.000E+00	**0.000E+00**	0.000E+00	**0.000E+00**	0.000E+00	**0.000E+00**	0.000E+00	**0.000E+00**	0.000E+00
AR PSOAWL	**2.385E+04**	4.345E+00	-2.370E-04	3.510E-04	-2.722E+01	1.865E-01	-1.700E-02	9.296E-03	-1.427E-02	1.150E-02	-2.545E-01	3.898E-01	-4.927E-04	4.927E-04	-7.652E-02	6.886E-02
D PSOAWL	2.384E+04	5.563E+00	-6.002E-02	1.845E-02	-2.773E+01	1.969E+00	-2.798E+00	3.415E-01	-1.246E+00	6.142E-02	-5.701E+01	1.384E+01	-8.869E-04	2.956E-04	-1.222E-01	8.224E-02
AR NPSO	2.376E+04	1.272E+01	-1.294E+00	6.649E-01	-5.080E+01	8.290E+00	-5.485E+00	3.485E-01	-3.901E+00	3.540E-01	-1.289E+02	1.207E+01	-9.857E-04	7.384E-07	-4.239E+00	5.568E+00
D NPSO	2.384E+04	4.510E+00	-3.601E-02	6.338E-03	-2.838E+01	6.074E-01	-2.061E+00	1.309E-01	-1.119E+00	2.124E-02	-9.923E+01	2.759E+01	-2.945E-04	2.940E-04	-5.611E-01	1.309E-01
D ARPSO	2.378E+04	5.998E+00	-1.231E+00	2.524E-01	-1.018E+02	8.868E+00	-1.246E+00	2.359E-01	-1.040E+00	1.012E-02	-8.281E+01	1.304E+01	-3.217E-04	4.412E-04	-8.167E-02	2.656E-02

Table 7: Test Data Neural Network Results

Velocity Equation	Fitness
Constriction	11766.53
linear inertia	11771.45
AIW PSO	11758.68
PSOA	10651.81
NPSO1	11763.56
NPSO2	11767.38
PSOAWL	11766.57
PSOA2	10790.92
PSOAWL2	11771.45
MOL	11758.68
PSO CV	11672.73
ARPSO	11771.43
ARPSO2	11766.63
DPSO	11771.26
AVPSO	11694.33
AR PSOAWL	11771.16
D PSOAWL	11762.08
AR NPSO	11722.51
D NPSO	11763.66
AR DPSO	11716.07

7. Conclusion

The aim of this paper was to compare a wide range of PSO velocity update equations and to apply them to the Watershed Management problem in an offline learning manner. Meta optimisation was implemented to ensure that the optimum parameters for each velocity update equation were selected. Overall the linear changing inertia PSO performs extremely well. It is thought that the reason for this is due to the unusual parameters found by the meta optimiser which leads to its increased performance. The proposed hybrid velocity update equation AR PSOAWL performs statistically the same as the linear changing PSO. The proposed AR PSOAWL provides faster convergence however it requires more computational time to execute. When comparing each velocity update equation on each of the 8 problems individually, it was found that the AV PSO performs best on the 6 out of 8 problems. The AV PSO performs best on simpler problems but lacks the ability to perform well on more complex problems that require more exploration. A neural network trained with the proposed PSOAWL2 performs best on unseen test data for the Watershed Management problem. However this variant only performs 4th best on the training data. This highlights the problem of over training for Watershed Management control.

The main contributions of this paper are:

1. Overall the best velocity update equations are the linearly changing inertia PSO, AR PSOAWL and AV PSO. The linearly changing inertia PSO performs statistically equal to the AR PSOAWL. The AV PSO performs best on the highest number of problems but performs badly on complicated problems.
2. A linear changing inertia provides superior performance than the constriction value. This demonstrates that stable convergence does not equate to superior performance.
3. Better performance can be obtained by utilizing both a negative inertia and an increasing inertia value.
4. For the AR PSO, meta optimisation reveals that a maximum diversity threshold that is lower than the minimum diversity threshold can provide better exploration.

5. Commonly used parameters such as those recommended as standard [9] do not provide the best performance. Meta optimisation reveals that very strange parameters that are not commonly implemented actually provide better performance.
6. The proposed hybrid velocity update equations, AR PSOAWL, performs extremely well providing faster convergence than other velocity update equations.
7. This research also provided a novel application of a neural network controller trained using PSO for the Watershed Management problem.

7.1. Future Work

There are many avenues of future work that have arisen as a result of this research. The inertia values and AR PSO diversity thresholds that were found by the meta optimiser lead to superior performance. A more in depth analysis into these parameters is needed to determine why this is. The AV PSO performs well on simple optimisation problems but not on more complex problems. Modifying this algorithm to also perform well on more complex problems would therefore be a worthy route for future research.

References

[1] Shafiq Alam, Gillian Dobbie, Yun Sing Koh, Patricia Riddle, and Saeed Ur Rehman. Research on particle swarm optimization based clustering: a systematic review of literature and techniques. *Swarm and Evolutionary Computation*, 17:1–13, 2014.

[2] Francesco Amigoni, Andrea Castelletti, and Matteo Giuliani. Modeling the management of water resources systems using multi-objective dcops. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 821–829. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[3] MA Arasomwan and Aderemi Oluyinka Adewumi. An adaptive velocity particle swarm optimization for high-dimensional function optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 2352–2359. IEEE, 2013.

[4] Thomas Bäck. Parallel optimization of evolutionary algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 418–427. Springer, 1994.

[5] Nikos Barbalios and Panagiotis Tzionas. A robust approach for multi-agent natural resource allocation based on stochastic optimization algorithms. *Applied Soft Computing*, 18:12–24, 2014.

[6] Mauro Birattari and Marco Dorigo. The problem of tuning metaheuristics as seen from a machine learning perspective. 2004.

[7] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[8] Justin Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pages 369–376, 1995.

[9] Daniel Bratton and James Kennedy. Defining a standard for particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 120–127. IEEE, 2007.

[10] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, Feb 2002.

[11] Carlos A Duchanoy, Marco A Moreno-Armendáriz, Leopoldo Urbina, Carlos A Cruz-Villar, Hiram Calvo, and J de J Rubio. A novel recurrent neural network soft sensor via a differential evolution training algorithm for the tire contact patch. *Neurocomputing*, 2017.

[12] Tohid Erfani and Rasool Erfani. Fair resource allocation using multi-population evolutionary algorithm. In *European Conference on the Applications of Evolutionary Computation*, pages 214–224. Springer, 2015.

[13] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[14] John J Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics*, 16(1):122–128, 1986.

[15] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[16] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989.

[17] Yukinobu Hoshino and Hiroshi Takimoto. Pso training of the neural network application for a controller of the line tracing car. In *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*, pages 1–8. IEEE, 2012.

[18] T Jayabarathi, Ravi Tej Kolipakula, M Vamsi Krishna, and Afshin Yazdani. Application and comparison of pso, its variants and hde techniques to emission/economic dispatch. *Arabian Journal for Science and Engineering*, 39(2):967–976, 2014.

[19] Chia-Feng Juang. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):997–1006, 2004.

[20] Andy J Keane. Genetic algorithm optimization of multi-peak problems: studies in convergence and robustness. *Artificial Intelligence in Engineering*, 9(2):75–83, 1995.

[21] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995.

[22] Petter Krus and Johan Andersson. Optimizing optimization for design optimization. In *ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 951–960. American Society of Mechanical Engineers, 2003.

[23] Pat Langley. *Elements of machine learning*. Morgan Kaufmann, 1996.

[24] Karl Mason, Jim Duggan, and Enda Howley. Evolving multi-objective neural network using differential evolution for dynamic economic emission dispatch. In *Proceedings of the 2017 on Genetic and Evolutionary Computation Conference Companion*. ACM, 2017.

[25] Karl Mason, Jim Duggan, and Enda Howley. Multi-objective dynamic economic emission dispatch using particle swarm optimisation variants. *Neurocomputing (In Press)*, 2017.

[26] Karl Mason, Jim Duggan, and Enda Howley. Neural network topology and weight optimization through neuro differential evolution. In *Proceedings of the 2017 on Genetic and Evolutionary Computation Conference Companion*. ACM, 2017.

[27] Karl Mason and Enda Howley. Avoidance strategies in particle swarm optimisation. In *Mendel 2015*, pages 3–15. Springer, 2015.

[28] Karl Mason and Enda Howley. Avoidance techniques & neighbourhood topologies in particle swarm optimisation. *Master's thesis, National University of Ireland Galway*, 2015.

[29] Karl Mason and Enda Howley. Exploring avoidance strategies and neighbourhood topologies in particle swarm optimisation. *International Journal of Swarm Intelligence*, 2(2-4):188–207, 2016.

[30] Karl Mason, Patrick Mannion, Jim Duggan, and Enda Howley. Applying multi-agent reinforcement learning to watershed management. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS 2016)*, 2016.

[31] Michael Meissner, Michael Schmuker, and Gisbert Schneider. Optimized particle swarm optimization (opso) and its application to artificial neural network training. *BMC bioinformatics*, 7(1):125, 2006.

[32] Robert E Mercer and JR Sampson. Adaptive search using a reproductive meta-plan. *Kybernetes*, 7(3):215–228, 1978.

[33] Ahmad Nickabadi, Mohammad Mehdi Ebadzadeh, and Reza Safabakhsh. A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied Soft Computing*, 11(4):3658–3670, 2011.

[34] Magnus Erik Hvass Pedersen. *Tuning & simplifying heuristical optimization*. PhD thesis, University of Southampton, 2010.

[35] Magnus Erik Hvass Pedersen and Andrew J Chipperfield. Simplifying particle swarm optimization. *Applied Soft Computing*, 10(2):618–628, 2010.

[36] Jacques Riget and Jakob S Vesterstrøm. A diversity-guided particle swarm optimizer-the arpso. *Dept. Comput. Sci., Univ. of Aarhus, Aarhus, Denmark, Tech. Rep*, 2:2002, 2002.

[37] A Immanuel Selvakumar and K Thanushkodi. A new particle swarm optimization solution to nonconvex economic dispatch problems. *Power Systems, IEEE Transactions on*, 22(1):42–51, 2007.

[38] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 1998.

[39] Yuhui Shi and Russell C Eberhart. Empirical study of particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999.

[40] Alice E Smith, David W Coit, Thomas Baeck, David Fogel, and Zbigniew Michalewicz. Penalty functions. *Evolutionary computation*, 2:41–48, 2000.

[41] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[42] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[43] Xiao-Feng Xie, Wen-Jun Zhang, and Zhi-Lian Yang. Dissipative particle swarm optimization. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1456–1461. IEEE, 2002.

[44] Chunming Yang and Dan Simon. A new particle swarm optimization technique. In *Systems Engineering, 2005. ICSEng 2005. 18th International Conference on*, pages 164–169. IEEE, 2005.

[45] Yi-Chen E Yang, Ximing Cai, and Dušan M Stipanović. A decentralized optimization algorithm for multiagent system–based watershed management. *Water Resources Research*, 45(8), 2009.

[46] Stelios H Zanakis and James R Evans. Heuristic optimization: Why, when, and how to use it. *Interfaces*, 11(5):84–91, 1981.

Appendix A. Section in Appendix

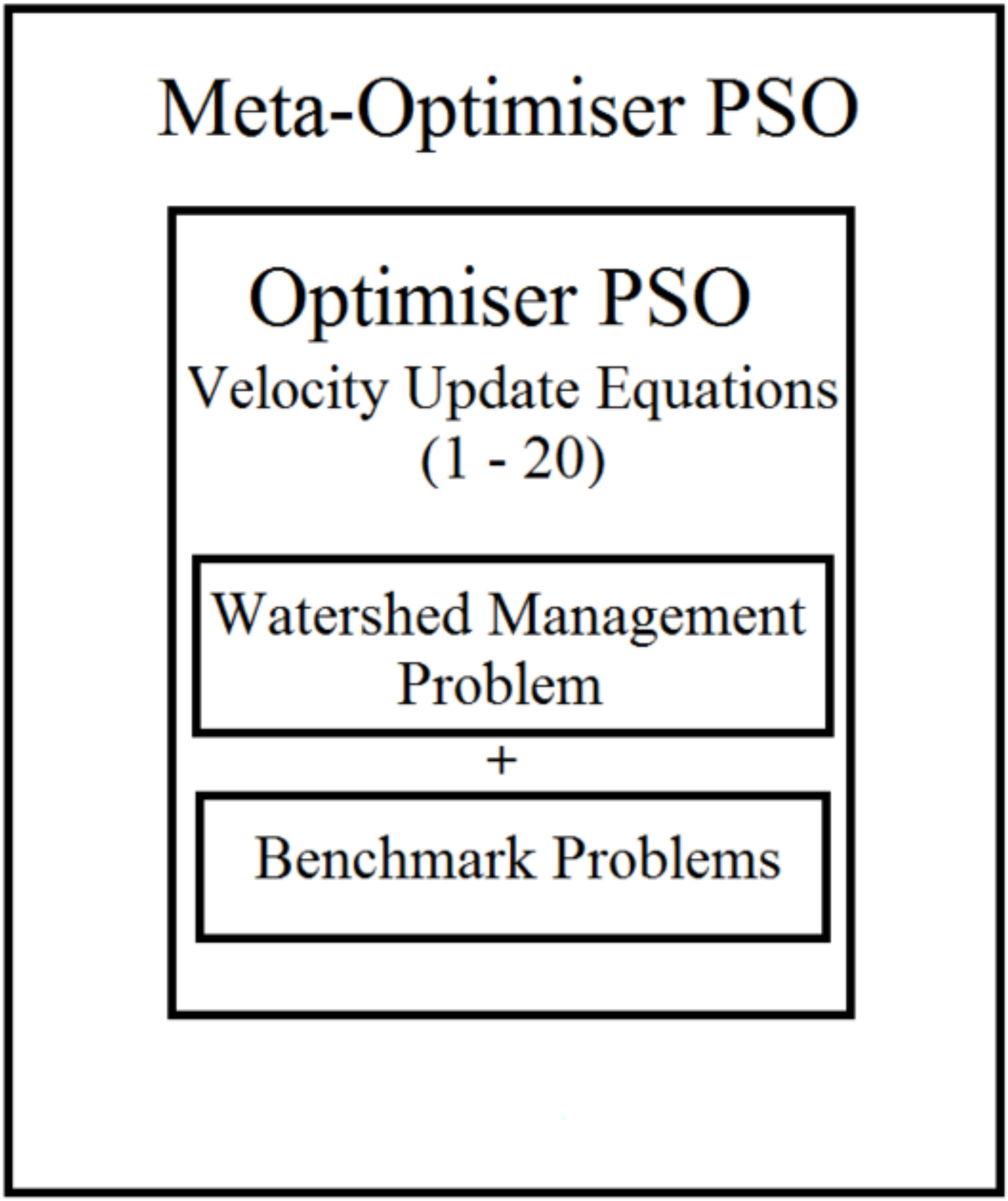
Table A.8: Training States

State	$Q_1(L^3)$	$Q_2(L^3)$	$S(L^3)$	State	$Q_1(L^3)$	$Q_2(L^3)$	$S(L^3)$
1	92	39	11	51	81	35	10
2	129	53	13	52	84	37	10
3	129	53	13	53	154	63	15
4	143	59	14	54	99	42	11
5	84	36	10	55	129	53	13
6	116	48	12	56	113	47	12
7	86	37	10	57	125	52	13
8	141	58	14	58	105	44	12
9	117	49	12	59	135	56	13
10	137	56	14	60	134	55	13
11	114	48	12	61	86	37	10
12	85	37	10	62	146	60	14
13	141	58	14	63	114	48	12
14	139	57	14	64	159	65	15
15	94	40	11	65	102	43	11
16	113	47	12	66	124	51	13
17	154	63	15	67	97	41	11
18	152	62	14	68	106	45	12
19	87	37	10	69	90	39	11
20	88	38	10	70	107	45	12
21	107	45	12	71	136	56	14
22	127	53	13	72	83	36	10
23	111	47	12	73	85	37	10
24	140	57	14	74	125	52	13
25	140	58	14	75	125	52	13
26	82	36	10	76	110	46	12
27	122	51	13	77	153	62	15
28	133	55	13	78	138	57	14
29	102	43	11	79	103	44	11
30	134	55	13	80	138	57	14
31	132	54	13	81	145	59	14
32	110	46	12	82	144	59	14
33	122	51	13	83	144	59	14
34	115	48	12	84	100	43	11
35	140	57	14	85	86	37	10
36	129	54	13	86	101	43	11
37	97	41	11	87	154	63	15
38	116	49	12	88	88	38	10
39	152	62	15	89	156	64	15
40	159	65	15	90	132	54	13
41	116	48	12	91	137	56	14
42	115	48	12	92	91	39	11
43	126	52	13	93	110	46	12
44	86	37	10	94	108	45	12
45	87	38	10	95	150	61	14
46	159	64	15	96	118	49	12
47	95	41	11	97	151	62	14
48	124	51	13	98	123	51	13
49	85	37	10	99	101	43	11
50	107	45	12	100	115	48	12

Table A.9: Test States

State	$Q_1(L^3)$	$Q_2(L^3)$	$S(L^3)$	State	$Q_1(L^3)$	$Q_2(L^3)$	$S(L^3)$
1	125	52	13	26	146	60	14
2	151	62	14	27	104	44	11
3	116	49	12	28	126	52	13
4	160	65	15	29	148	61	14
5	148	60	14	30	83	36	10
6	153	62	15	31	82	36	10
7	148	60	14	32	121	50	13
8	149	61	14	33	145	59	14
9	122	51	13	34	145	60	14
10	86	37	10	35	89	37	10
11	84	36	10	36	98	42	11
12	112	47	12	37	133	55	13
13	106	45	12	38	95	40	11
14	90	39	11	39	158	64	15
15	134	55	13	40	102	43	11
16	145	59	14	41	100	43	11
17	87	37	10	42	131	54	13
18	155	63	15	43	111	48	13
19	149	61	14	44	109	46	12
20	126	52	13	45	157	64	15
21	105	44	12	46	115	48	12
22	117	49	12	47	80	35	10
23	83	36	10	48	104	44	11
24	102	43	11	49	136	56	13
25	121	50	13	50	126	52	13

- The best velocity update equations are: standard PSO, proposed AR PSOAWL and AV PSO.
- A linear changing inertia provides superior performance than the constriction value.
- Better performance is obtained by utilizing a negative and an increasing inertia.
- Unusual parameters provide better performance than those commonly used.
- PSO is applied to the Watershed Management problem using Neural Networks.


Meta-Optimiser PSO
Optimiser PSO
Velocity Update Equations
(1 - 20)
Watershed Management Problem
+
Benchmark Problems