Neural-Guided Particle Swarm Optimization

Amani M. Benhalem and Michael A. Lones
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, UK
ab3@hw.ac.uk, m.lones@hw.ac.uk

***Abstract*—In this paper, we present a new form of particle swarm optimisation (PSO) in which each particle uses an artificial neural network (ANN) to guide its movements. Information about each of the particle's informants is passed as input to the ANN and the ANN's outputs are then used to select which informant to follow at the next iteration. Using a distributed evolutionary process, each particle's ANN is able to learn about the solution landscape over the course of an optimisation run, potentially allowing the particle to avoid unfavourable regions. An initial evaluation of this approach using a suite of 5 continuous optimisation functions suggests that it improves performance, managing to get consistently closer to the global optima than conventional PSO on all of these problems. An analysis of the trajectories indicates that the behaviour of the algorithm is quite different to conventional PSO, with a much higher degree of exploration than the baseline PSO algorithm.**

***Index Terms*—Artificial Neural Networks, Particle Swarm Optimization**

I. Introduction

Particle swarm optimisation (PSO) [1] is a well-known and successful optimiser that has received a lot of attention since its initial development in the mid-1990s [2]. The PSO algorithm is broadly modelled on the idea of swarm intelligence, particularly the concept that complex behaviour can emerge from simple interactions between relatively simple agents. In the case of PSO, the simple agents are local search processes known as particles, and the emergent behaviour is global optimisation, which the particles achieve by following one another according to certain rules. Recently, there has been considerable interest in developing new swarm optimisers based around models of animal foraging behaviours; however, this approach has been broadly criticised for producing algorithms that often differ in trivial ways from PSO and other optimisers [3], [4]. A more promising recent development is the use of neural and evolutionary algorithms to develop new optimisers from scratch [5], [6], though the resulting algorithms are often quite different to those the community are used to working with, and do not leverage our current understanding of how to optimise.

In this work, we take a different approach: rather than using human intuition to improve existing algorithms, or building a new algorithm from scratch, we attempt to use artificial neural networks (ANN) to learn how to improve a standard PSO algorithm. In this paper, we explore an initial approach based around the idea of embodied evolution [7]: each particle is assigned its own ANN, which it uses to guide its decision about which informant to follow at each iteration, and a distributed evolutionary process is used to optimise the weights of the ANN. The idea is that this will allow the ANNs to collectively learn about the search space over the course of an optimisation run, potentially identifying the areas which are best to explore or avoid. The baseline behaviour of PSO remains the same, i.e. the velocities of particles are updated at each iteration according to the standard equation; the key difference is that the choice of informant to follow is determined by the ANN rather than always following the one with the fittest personal best.

This paper reports on an initial evaluation of this approach using a suite of continuous function benchmarks, looks at how the size of the ANN affects performance, and aims to give an idea of how its behaviour differs from standard PSO. The paper is organised as follows: Section II discusses related work, Section III describes the methodology, Section IV presents results and analysis, Section V concludes.

II. Related Work

In essence, this work adds an ANN to each particle within a PSO swarm, and the particle then uses the ANN to select between its informants. In this respect, the approach has commonalities with previous attempts to increase the cognitive abilities of PSO particles. For instance, Broderick and Howley [8] reported that they were able to improve the performance of standard PSO by using an internal memory to store additional best positions for each particle, providing more information on which to base their velocity updates. Ciuprina et al. [9] went further in their Intelligent-Particle Swarm Optimization (IPSO), by allowing particles to record group experiences, unpleasant memories (which are comparable to tabu search), local landscape models based on virtual neighbours, and also carry out memetic replication of successful behaviour parameters. They report better convergence speed and better avoidance of local optima. Another related area of PSO research is the use of dynamic topologies, which attempt to change a particle's informants over the course of a run [10].

However, this work arguably has more in common with swarm robotics than previous attempts to improve PSO within an optimisation context. Particularly notable is the similarity with embodied evolution [7], where each robot within a swarm moves around the landscape under the direction of a controller, which is often an ANN. Over time, the robot mutates its controller, and when in the proximity of another robot, it can

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

Algorithm 1 Pseudocode for PSO algorithm

$best \leftarrow <>$ ▷ Best location
for each particle i **do** ▷ Initialise particles
  for each dimension d **do**
    Initialize position x_{id} randomly within range
    Initialize velocity v_{id} randomly within range
  end for
end for
for each particle i **do** ▷ Allocate informants
  Randomly select nf particles as informants
end for
do
  for each particle i **do** ▷ Update bests
    $pbest_i \leftarrow$ best previous location of i
    if fitness(x_i) $<$ fitness($best$) **then**
      $best \leftarrow x_i$
    end if
    $gbest_i \leftarrow pbest$ of first informant
    for each subsequent informant j **do**
      if fitness($pbest_j$) $<$ fitness($gbest_i$) **then**
        $gbest_i \leftarrow pbest_j$
      end if
    end for
  end for
  for each particle i **do** ▷ Update vs and xs
    for each dimension d **do**
      $v_{id} \leftarrow wv_{id} + c_1 r_1 (pbest_{id} - x_{id})$
        $+ c_2 r_2 (gbest_{id} - x_{id})$
      $x_{id} \leftarrow x_{id} + v_{id}$
    end for
  end for
while maximum iteration or minimum error not attained

Algorithm 2 Pseudocode for ANN-PSO algorithm

$best \leftarrow <>$ ▷ Best location
for each particle i **do** ▷ Initialise particles
  for each dimension d **do**
    Initialize position x_{id} randomly within range
    Initialize velocity v_{id} randomly within range
  end for
  $\boldsymbol{ann_i} \leftarrow$ random neural network
  fitness($\boldsymbol{ann_i}$) $\leftarrow 0$
end for
for each particle i **do** ▷ Allocate informants
  Randomly select nf particles as informants
end for
do
  for each particle i **do** ▷ Update bests
    $pbest_i \leftarrow$ best previous location of i
    if fitness(x_i) $<$ fitness($best$) **then**
      $best \leftarrow x_i$
    end if
    $gbest_i \leftarrow pbest$ of first informant
    for each subsequent informant j **do**
      if $\boldsymbol{ann_i}(pbest_j) > \boldsymbol{ann_i}(gbest_i)$ **then**
        $gbest_i \leftarrow pbest_j$
      end if
    end for
  end for
  for each particle i **do** ▷ Update vs and xs
    for each dimension d **do**
      $v_{id} \leftarrow wv_{id} + c_1 r_1 (pbest_{id} - x_{id})$
        $+ c_2 r_2 (gbest_{id} - x_{id})$
      $x_{id} \leftarrow x_{id} + v_{id}$
    end for
  end for
  for each particle i **do** ▷ Evaluate ANNs
    $change \leftarrow$ improvement in fitness(x_i)
    $\boldsymbol{ann_i}.success \leftarrow \boldsymbol{ann_i}.success + change$
  end for
  for each particle i **do** ▷ Evolve ANNs
    $ann_{best} \leftarrow$ most successful ann among informants
    if $ann_{best}.success > \boldsymbol{ann_i}.success$ **then**
      $\boldsymbol{ann_i} \leftarrow$mutate($ann_{best}$)
    end if
  end for
while maximum iteration or minimum error not attained

swap its controller or carry out recombination with the other robot's controller, thus allowing information to spread within the population. A similar mechanism for distributed evolution of ANNs is used in this work. A key difference, however, is that our approach combines this distributed evolution of ANNs with the standard rules of PSO, using the ANN to guide the choice of informant to follow rather than controlling movement directly. There has also been work on adding other (i.e. not neural) learning mechanisms to swarm robotics; for instance, in [11], PSO was hybridised with a modified Q-learning approach, which the authors found to improve learning time.

This work also relates to broader research on using evolutionary and neural approaches to develop new optimisation algorithms, both swarm-based and otherwise. For instance, in [12], an evolutionary algorithm was used to learn new low-level rules for swarm optimisers, and there is a growing body of work on using deep ANNs to learn and generate optimisation behaviours [5], [13]. Again, a key difference in this work is that optimisation behaviours are not being learnt or generated from scratch, but rather within the context of the dynamics of the existing PSO algorithm. In this respect, it represents a mid-point between efforts to develop new swarm algorithms by hand [4], and using machine learning to develop entirely new optimisers [6].

In our approach, the ANN is being used to decide which particle from a group to follow, so in a sense it is evaluating the particles in the group. In this regard, it also has some commonality with the use of surrogate fitness models in evolutionary algorithms, where ANNs are often used to approximate the fitness of search points [14]. However, the role of the ANN is much less clear cut in our approach, and there is no selective

pressure towards it being used solely (or at all) as a fitness approximator. It is also worth noting that there has been a lot of previous work on using PSO to optimise ANNs (e.g. [15]); this is unrelated to the work described in this paper, where we are essentially going in the opposite direction, using ANNs to optimise PSO.

III. Methodology

A. PSO

Algorithm 1 outlines the baseline implementation of PSO used in this work. PSO is an iterative algorithm that uses a population (swarm) of search processes (particles) to explore a solution space. In addition to having a position (search point) within the solution space, each particle also records a personal best, which is the best search point it has seen during its search trajectory. Each particle has a velocity within the search space, and samples a new search point at each iteration by adding its velocity to its current point. For each dimension d of each particle i, the velocity is updated at each iteration using the equation:

$$v_{id}^{t+1} = \omega v_{id}^{t} + c_1 r_1 (pbest_{id}^{t} - x_{id}^{t}) + c_2 r_2 (gbest_{id}^{t} - x_{id}^{t}) \quad (1)$$

where x is the particle's current position, $pbest$ is the particle's personal best, r_1 and r_2 are uniformly distributed random variables, and ω, c_1 and c_2 are acceleration coefficients that weight the new velocity towards the previous velocity and the search points $pbest$ and $gbest$. In early forms of PSO, the group best, $gbest$, was the best personal best amongst all the particles. In more recent implementations, however, it is common for each particle to be assigned a group of other particles (known as informants), and $gbest$ is the best personal best amongst this group of particles. Informants can be assigned in various ways; in this paper, we take the common approach of doing this randomly at the start of a run, meaning that particles tend to have overlapping groups of informants. Typical values from the literature of 0.9, 2.0 and 2.0 are used for the weights ω, c_1 and c_2 [16], [17].

B. ANN-PSO

Our implementation of neural-guided PSO (ANN-PSO) builds upon the standard version of PSO outlined in Algorithm 1. ANN-PSO is outlined in Algorithm 2, with highlighting showing the parts of the algorithm that differ from Algorithm 1. A key difference is that each particle is assigned an ANN, which, in this paper, is a simple multi-layer Perceptron with one hidden layer. The number of input neurons is the same as the dimensionality of the problem, and it has one output neuron. At each iteration, each informant's $pbest$ is used as an input to the ANN, generating a response value from the output neuron (lines 19-23 in Algorithm 2). The $pbest$ with the highest response value is then selected as the $gbest$ used in Equation 1.

At the start of a run, the weights of each particle's ANN are randomly initialised in the range $[-1, 1]$. They are then trained using a distributed evolutionary process that is similar to the embodied evolution approaches used in swarm robotics. Within this evolutionary process, the fitness of an ANN is measured by its cumulative success in guiding particles to better locations, in a manner akin to reinforcement learning. This is represented by an ANN's *success* variable in Algorithm 2. Initially, the success of each particle's ANN is zero. Then, at each iteration, the improvement in each particle's fitness is measured and this value is added to its ANN's success. Hence, if a particle moves to a better search point as a result of following guidance from its ANN, the success value of its ANN increases; conversely, if it moves to a worse search point, the ANN's success value decreases. At the end of each iteration, each particle looks at the success values of its informants' ANNs. If one of these has a higher value than its own ANN, then it replaces its own ANN with a mutated copy of the informant's ANN (if more than one have higher values, then the best one is used). Mutation involves randomly changing the value of one of the ANN's weights, whilst keeping its accumulated success value.

C. Benchmark Functions

At this stage, our aim is to obtain an impression of how ANN-PSO compares behaviourally with standard PSO. To this end, we chose 5 continuous-valued benchmark functions from the optimisation literature, selected to capture a range of different landscapes and functional properties. These are listed in Table I. The sphere function is the simplest: unimodal, convex and separable. Dixon-Price and Zakharov are also unimodal, but are non-separable and non-convex, which makes them more challenging. Levy and Rastrigin are both multimodal, the former with just a few local optima, the latter with many.

To give an idea of how the approach scales, quantitative results are collected for 5, 10 and 30-dimensional versions of the benchmark functions. 2-dimensional versions of the functions are used to visualise particle trajectories. The search space bounds are $[-5, 5]$ for all dimensions. Particle velocities are initialised in the range $\pm[0.5, 2.5]$ for each dimension.

IV. Results

Tables II–VI show the results. For each problem and dimensionality, the mean distance of the best solution from the global optimum is calculated over the course of 50 independent runs. Each run uses 40 iterations and a swarm size of 25 particles, giving a function evaluation budget of 1×10^3. Additionally, for ANN-PSO, three different neural network sizes are compared, with 5, 10 or 20 neurons in the hidden layer. In all cases, non-parametric Mann Whitney U tests show that the means of the ANN-PSO and PSO fitness distributions are significantly different at the 95% confidence level.

From these tables, it is evident that PSO-ANN performs better than PSO. The best mean result (shown underlined) is produced by ANN-PSO for every function in this test suite, and this advantage is retained as the dimensionality of the problem increases. This suggests (for these functions at least), that using an ANN to guide PSO is beneficial. For most of the problems, this benefit appears to be quite sizeable. However, it

TABLE I: Benchmark functions

Function	Definition
Sphere	$f(x) = \sum_{i=1}^{d} x_i^2$
Dixon-Price	$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{d} i(2x_i^2 - x_{i-1})^2$
Zakharov	$f(x) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^4$
Levy	$f(x) = \sin^2(\pi w_1) + \sum_{i=1}^{d-1}(w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] + (w_d - 1)^2[1 + \sin^2(2\pi w_d)]$, where $w_1 = 1 + \frac{x_i - 1}{4}$, for all $i - 1, ..., d$
Rastrigin	$f(x) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i)]$

TABLE II: Sphere

Dimension	Method	Neurons	Mean	Std. dev.
5	ANN-PSO	5	1.298	0.612
		10	1.129	0.528
		20	0.915	0.467
	PSO		2.006	1.320
10	ANN-PSO	5	5.858	2.166
		10	5.679	1.840
		20	5.263	1.838
	PSO		13.146	4.603
20	ANN-PSO	5	19.924	4.787
		10	19.735	4.601
		20	19.081	4.292
	PSO		47.216	9.156

TABLE III: Dixon-Price

Dimension	Method	Neurons	Mean	Std. dev.
5	ANN-PSO	5	6.706	4.739
		10	4.879	3.492
		20	4.517	3.183
	PSO		12.828	13.012
10	ANN-PSO	5	127.568	108.117
		10	124.941	78.703
		20	108.886	68.954
	PSO		622.307	483.036
20	ANN-PSO	5	1844.718	930.227
		10	1815.730	905.882
		20	1810.545	838.109
	PSO		9888.205	3487.883

TABLE IV: Zakharov

Dimension	Method	Neurons	Mean	Std. dev.
5	ANN-PSO	5	2.287	1.470
		10	1.964	1.152
		20	1.721	0.846
	PSO		4.880	2.846
10	ANN-PSO	5	13.296	4.615
		10	12.147	4.499
		20	11.882	4.240
	PSO		28.717	9.506
20	ANN-PSO	5	44.624	12.649
		10	42.186	12.300
		20	41.802	10.136
	PSO		79.161	15.777

TABLE V: Levy

Dimension	Method	Neurons	Mean	Std. dev.
5	ANN-PSO	5	0.252	0.092
		10	0.243	0.092
		20	0.241	0.091
	PSO		0.529	0.304
10	ANN-PSO	5	1.301	0.406
		10	1.253	0.401
		20	1.218	0.382
	PSO		3.478	1.009
20	ANN-PSO	5	4.522	1.104
		10	4.487	1.166
		20	4.415	1.069
	PSO		12.369	2.722

is perhaps significant that the smallest margin of improvement is found for Rastrigin, the function with the most complex landscape, and in future work it would be interesting to further explore the relationship between landscape complexity and the performance of PSO-ANN.

It can also be seen that, for each problem and dimensionality, increasing the size of the neural network improves the performance of ANN-PSO, although the benefit of doing this is small relative to the benefit of introducing the ANN in the first place. This improvement could be explained by the greater computational capacity of larger ANNs. However, increasing the size of the ANN also increases the search space, so there is likely to be a trade-off between these two factors at some point (from the results, this is presumably beyond 20 neurons).

A. Search Trajectories

These quantitative results seem to show a benefit to using ANNs to guide PSO particles. To get more insight into how this is achieved, we carried out further experiments using 2-dimensional versions of the problems, which can be readily visualised. Figs. 1-5 show plots of search trajectories generated by ANN-PSO and PSO, with one example for each of the benchmark functions. Each of the particles in the swarms is shown using a different colour. A red cross shows the location of the global optimum, and a black cross shows the best solution found in a run.

From these plots, it is obvious that the behaviours of the two algorithms are quite different. In standard PSO, the particles tend to explore different areas of the search space, and each

TABLE VI: Rastrigin

Dimension	Method	Neurons	Mean	Std. dev.
5	ANN-PSO	5	17.857	4.398
		10	17.129	3.598
		20	16.675	3.418
	PSO		20.129	5.015
10	ANN-PSO	5	59.512	9.089
		10	58.547	7.845
		20	58.076	7.290
	PSO		70.662	10.596
20	ANN-PSO	5	159.128	13.300
		10	158.150	12.734
		20	156.094	11.010
	PSO		192.600	14.322

of them tends to converge to a particular point. In ANN-PSO, the trajectories are much more interspersed, and they seem to maintain movement throughout the run. This suggests that the improvement due to ANN-PSO may, at least in part, be due to increased exploration of the search space. The benefit of this can be seen, for instance, in Fig. 5, where ANN-PSO is able to explore more of the local optima within the highly multimodal landscape of the Rastrigin function, and consequently finds the basin containing the global optimum.

Our initial hypothesis when developing this method was that the ANN would learn to explore or avoid certain areas of the search space as a result of learning from past experiences. However, this behaviour is not apparent from the trajectory plots, which might suggest that the ANN is carrying out a different role to the expected one. Having said that, it is difficult to infer the ANN's role from these trajectory plots alone. Further analysis of the evolved ANNs may help to understand their role. However, it may also be the case that their role depends on the dimensionality of the problem. For low-dimensional spaces, there may be little pressure to do anything beyond increasing exploration. For higher dimensional spaces, where the search volume is much greater, this is unlikely to be the case, so there may be value to looking more closely at the trajectories generated when solving higher-dimensional problems.

V. Conclusions

This paper explores the idea of using ANNs to increase the cognitive capacity of particles within PSO. The approach assigns a separate ANN to each particle, and this is used to choose which informant to follow at each iteration. The population of ANNs are then evolved in a distributed manner based on their ability to successfully guide particles towards higher fitness regions. Results on five benchmark problems suggest that this approach significantly improves the ability of PSO to find the global optimum in a range of different solution landscapes.

In future work, we aim to gain more insight into *how* the ANNs improve search. In this paper, an initial analysis of trajectories on 2-dimensional versions of the problems suggests that this may be due to a higher degree of exploration, but it remains unclear whether this is also true in the higher-dimensional cases. An investigation of higher-dimensional trajectories, although more challenging, may give some insight into this. Analysis of the behaviour of the evolved ANNs may also be informative.

More work is also required to understand *when* this approach works, since it is unlikely that it will lead to an improvement in all continuous optimisation landscapes. Understanding of this may emerge from a better understanding of what roles the ANNs are playing, e.g. are they explicitly remembering features of the landscape (in which case more complex landscapes could be problematic) or are they doing something more diverse. The use of standard benchmark sets (e.g. CEC 2015 [18]) may also contribute towards this.

The approach described in this paper is just one way of using ANNs to guide PSO, and there are many other possibilities to investigate. More complexity could be added by passing additional information to the ANN, for instance the velocity or current position of an informant in addition to its personal best. Simpler variants could use a single ANN for the entire population, and train this using back-propagation (though it may be difficult to find a suitable error function for this). However, one of the motivations for using distributed evolution to train the ANNs is that the algorithm (like PSO itself) can be implemented within a distributed system; an example of where this might be beneficial is within a swarm robotics context. It could also be possible for learning to be carried out over multiple runs, rather than reinitialising the ANNs each time; this might be useful as a form of transfer learning, or for specialising PSO to particular classes of landscape.

Acknowledgment

We are grateful to the Libyan Embassy in the UK for their support of the first author, especially during the current difficult times.

References

[1] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the 1999 Congress on Evolutionary Computation (CEC '99)*, vol. 3. IEEE, 1999, pp. 1945–1950.

[2] M. R. Bonyadi and Z. Michalewicz, "Particle swarm optimization for single objective continuous space problems: a review," *Evolutionary Computation*, vol. 25, no. 1, pp. 1–54, 2017.

[3] K. Sörensen, "Metaheuristics—the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.

[4] M. A. Lones, "Mitigating metaphors: A comprehensible guide to recent nature-inspired algorithms," *SN Computer Science*, vol. 1, no. 1, p. 49, 2020.

[5] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 2017, pp. 3751–3760.

[6] M. Lones, "Optimising optimisers with Push GP," in *European Conference of Genetic Programming (EuroGP 2020)*, 2020.

[7] R. A. Watson, S. G. Ficici, and J. B. Pollack, "Embodied evolution: Distributing an evolutionary algorithm in a population of robots," *Robotics and autonomous systems*, vol. 39, no. 1, pp. 1–18, 2002.

[8] I. Broderick and E. Howley, "Particle swarm optimisation with enhanced memory particles," in *International Conference on Swarm Intelligence*. Springer, 2014, pp. 254–261.

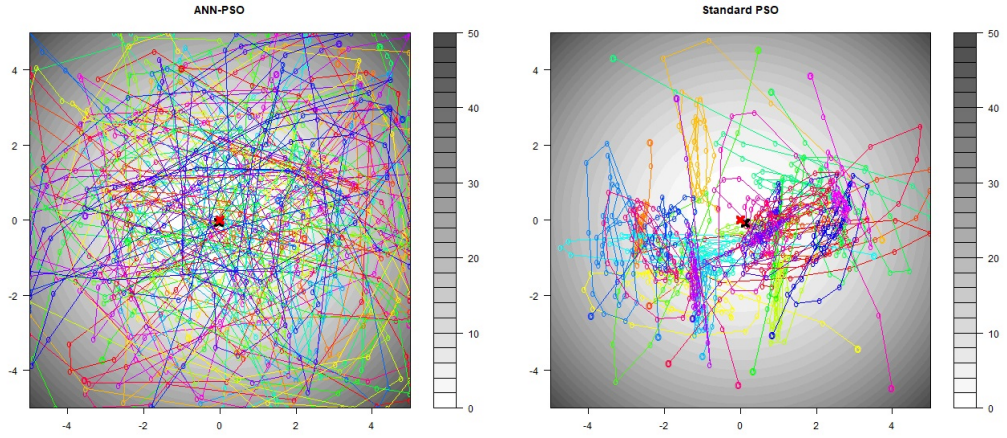

Fig. 1: Comparison of particle trajectories in standard PSO and ANN-PSO on SPHERE function.

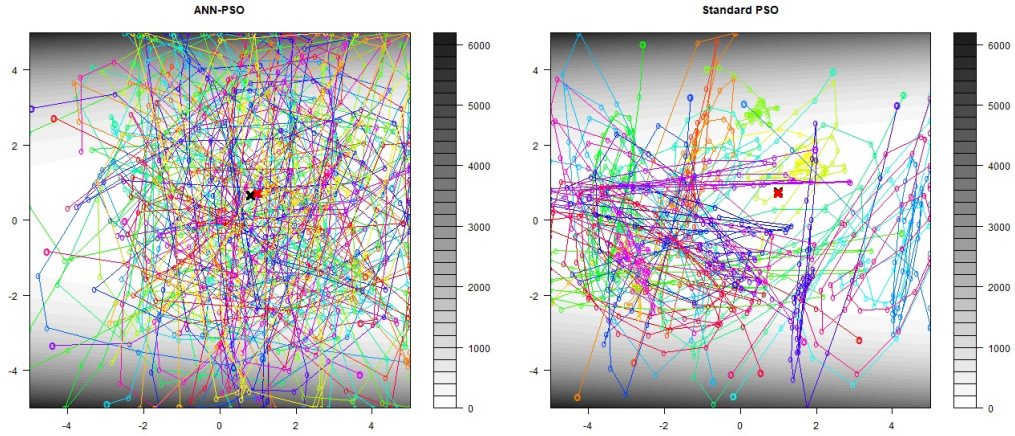

Fig. 2: Comparison of particle trajectories in standard PSO and ANN-PSO on DIXON-PRICE function.

[9] G. Ciuprina, D. Ioan, and I. Munteanu, "Use of intelligent-particle swarm optimization in electromagnetics," *IEEE Transactions on Magnetics*, vol. 38, no. 2, pp. 1037–1040, 2002.

[10] L. Wang, B. Yang, and J. Orchard, "Particle swarm optimization using dynamic tournament topology," *Applied Soft Computing*, vol. 48, pp. 584–596, 2016.

[11] O. Watchanupaporn and P. Pudtuan, "Multi-robot target reaching using modified Q-learning and PSO," in *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2016, pp. 66–69.

[12] A. Bogdanova, J. P. Junior, and C. Aranha, "Franken-swarm: grammatical evolution for the automatic generation of swarm-like metaheuristics," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '19)*. ACM, 2019, pp. 411–412.

[13] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[14] M. Holeňa, D. Linke, U. Rodemerck, and L. Bajer, "Neural networks as surrogate models for measurements in optimization algorithms," in *International Conference on Analytical and Stochastic Modeling Techniques and Applications*. Springer, 2010, pp. 351–366.

[15] C. Zhang, H. Shao, and Y. Li, "Particle swarm optimisation for evolving artificial neural network," in *IEEE international conference on systems, man and cybernetics (SMC 2000)*, vol. 4. IEEE, 2000, pp. 2487–2490.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization (pso)," in *Proc. IEEE International Conference on Neural Networks, Perth, Australia*, 1995, pp. 1942–1948.

[17] B. Liu, L. Wang, Y.-H. Jin, F. Tang, and D.-X. Huang, "Improved particle swarm optimization combined with chaos," *Chaos, Solitons & Fractals*, vol. 25, no. 5, pp. 1261–1271, 2005.

[18] J. Liang, B. Qu, P. Suganthan, and Q. Chen, "Problem definitions and evaluation criteria for the CEC 2015 competition on learning-based real-parameter single objective optimization," *Technical Report 201411A, Computational Intelligence Laboratory, Zhengzhou University and Technical Report, Nanyang Technological University*, vol. 29, pp. 625–640, 2014.

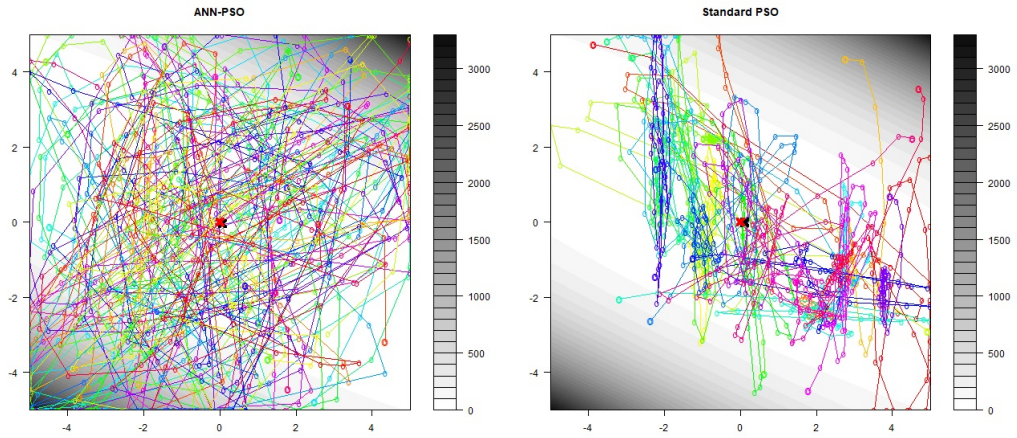

Fig. 3: Comparison of particle trajectories in standard PSO and ANN-PSO on ZAKHAROV function.

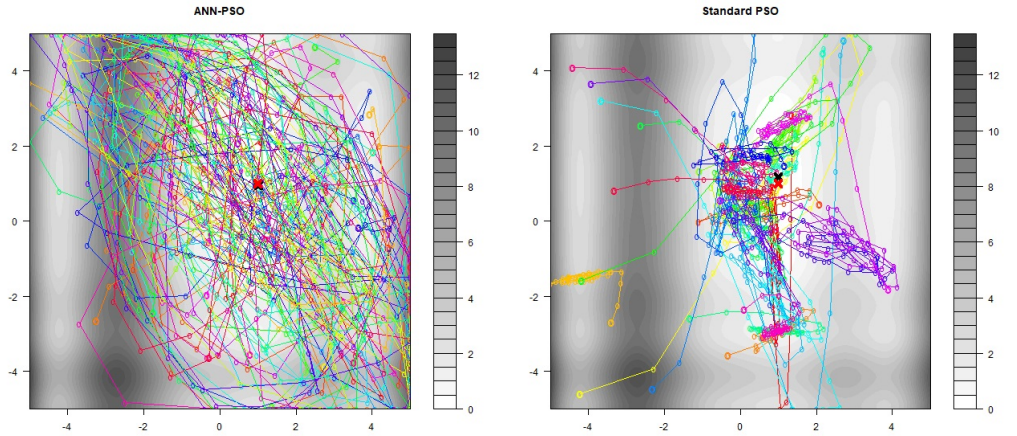

Fig. 4: Comparison of particle trajectories in standard PSO and ANN-PSO on LEVY function.

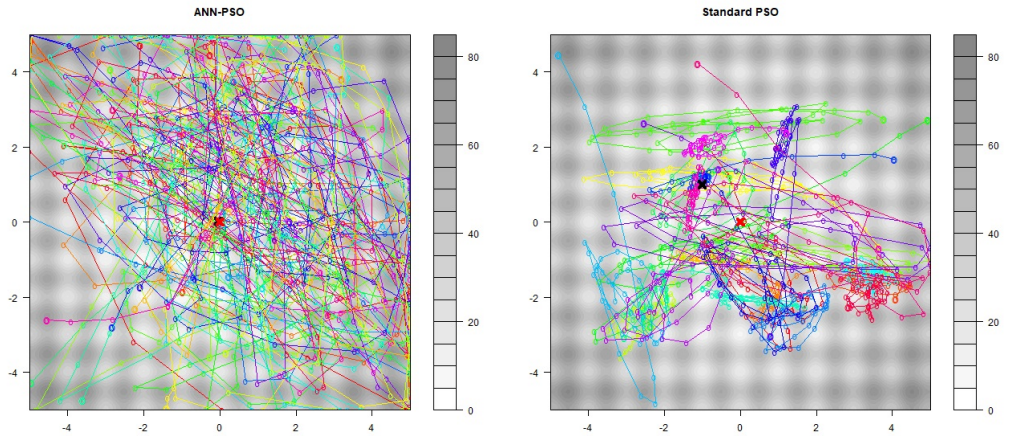

Fig. 5: Comparison of particle trajectories in standard PSO and ANN-PSO on RASTRIGIN function.