PSO Based on Surrogate Modelling as Meta-Search to Optimise Evolutionary Algorithms Parameters

Ahmed Kattan
Um Al Qura University
AI Real-World Applications Lab
Department of Computer Science
Kingdom of Saudi Arabia
ajkattan@uqu.edu.sa

Muhammad Arif
Um Al Qura University
AI Real-World Applications Lab
Department of Computer Science
Kingdom of Saudi Arabia
mahamid@uqu.edu.sa

ABSTRACT

The problem of setting suitable parameters for population-based Evolutionary Algorithms (EA) is not new. However, the process of tuning the EA parameters is still challenging, since their sensitivity to the given problem is highly non-linear. This paper proposes a framework that uses Particle Swarm Optimisation (PSO) based on Surrogate Modelling (SM) to optimise population-based EA parameters before they can be applied to solve problems. The proposed framework is comprised of two components; PSO that searches the parameters space and a Radial Basis Function Networks (RBFN) surrogate model to guide it. The main advantage of our model is that it optimises the EA parameters in a way that ensures that EA searches the problem within a limited number of evaluations. Experiments with three different benchmark problems demonstrate that our proposed framework managed to assist a Genetic Algorithm (GA) in order to optimise its parameters and achieves better solutions than the use of Standard PSO without surrogate assistance to optimise the GA parameters, Standard GA that is applied directly to the problem with fixed parameters settings, Standard 1+1 Evolutionary Strategy (ES) applied directly to the problem and simple Random Search.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]; I.2.8 [**Heuristic Methods**]

General Terms

Algorithms

Keywords

Parameters Tuning,Particle Swarm Optimisation, Evolutionary Algorithms, Surrogate, Radial Basis Function Networks.

1. INTRODUCTION

The problem of setting suitable parameters for population-based EAs has been the focus of an active research area in the past few decades [1]. However, tuning the EA parameters is still a challenging task since their sensitivity to the given problem is highly non-linear. A common parameter setting strategy involves running the algorithm multiple times with different settings, using a human-in-the-loop approach, in an attempt to fine-tune the EA approach to the given problem [1]. This can be a wearisome process; therefore, oftentimes, one is tempted to use default settings or settings in the literature that address similar problems.

The problem of parameter tuning can be seen as a classical optimisation problem. However, it requires a considerable amount of computational resources especially when the fitness function is computationally expensive (which is the case with most real-world problems). In such a circumstance, the process of tuning the parameters can become infeasible. Furthermore, in order to validate the goodness of a parameters set one may need to run the EA several times due to their stochastic nature. Thus, it is attractive to automate this process. One solution is to use “parameters sweep" strategy [2]. However, a proper parameters sweep can easily grow exponentially due to its combinatorial search space size. Another existing solution in the literature is to use an optimisation method as a top-level (or meta-search) [3]. This scenario raises an interesting question of how to tune the parameters of the top-level optimiser? This question does not seem to have any obvious answer [1]. Back to the computational intensity problem, one possible solution to reduce the computational cost of the top-level EA when optimising the low-level EA's parameters is to use approximation models, which are known as surrogate models, that can predict the fitness or objective function evaluation [4]. This can expedite the search by generating many candidate solutions in the search space at a smaller evaluation cost. The selection of these models is based on two main assumptions: *(a)* that their evaluation cost of the fitness function is much cheaper in comparison to the original evaluation (i.e., running the EA to validate the goodness of a parameters set), and *(b)* that they can successfully approximate the search space of the problem.

In this paper, we propose a framework that is based on PSO and RBFN surrogate to automatically optimise the EA parameters. The contributions of this paper are twofold:

1. We propose the use of PSO as a meta-search for other EA techniques. To the best of our knowledge, PSO-based on SM has never been used to search population-based EA's parameters space.

2. Unlike other parameters optimisation methods, the proposed framework ensures that EA searches the problem within a limited number of evaluations. Thus, our framework suggests the best settings under a pre-defined maximum number of evaluations.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO'12, July 7–11, 2012, Philadelphia, Pennsylvania, USA.
Copyright 2012 ACM 978-1-4503-1177-9/12/07 ...$10.00.

Experiments with three different benchmark problems demonstrate that our proposed framework managed to assist GA to optimise its parameters and achieves better solutions than other standard systems (details in Section 5).

The remaining of this paper is organised as follows: Section 2 briefly presents some of the previous work that is related to EA parameters optimisation via surrogate and PSO based surrogate systems. Section 3 details our proposed framework. Experiments' settings and results are presented Sections 4 and 5, respectively. Section 6 present some discussion on the results. Finally, Section 7 presents some conclusive remarks and future work.

2. RELATED WORK

2.1 Parameter Optimisation Techniques

In terms of performance or the convergence of the EA, the parameter tuning of the evolutionary algorithms showed promising results. Eiben et al. [5] described two kinds of strategies for parameter setting in the evolutionary algorithms. In parameter control, the parameters of an algorithm are constantly updated during a single run according to a certain control strategy. Conversely, in the parameter tuning approach, for a particular run of an algorithm, a single set of parameters is used and its performance is assessed according to some performance measure. A comparison of parameter tuning methods can be found in [6, 1]. Preuss et al [7] demonstrated the efficacy of surrogate models in tuning the parameters of the optimisation algorithm. Rudolph et al. [8] proposed a two-layered optimisation mechanism to optimise the meta models at two different levels; at level-1, surrogate model is used based on true objective function calculations and at level-2 another surrogate model is used as meta-model. In [1], the author reviewed parameters tuning mechanisms at different stages of EA and pointed out that there is no universal strategy with which to tune the internal parameters of an evolutionary algorithm for all types of applications.

None of the above work defines a limit of evaluations before optimising the EA parameters. The proposed framework suggests the best settings under a pre-defined maximum number of evaluations.

2.2 PSO-Based Surrogate

For computationally expensive optimisation, various meta-models for fitness approximation are proposed to accelerate the convergence of PSO. Bird et al [9] used least square regression to estimate the local fitness function and location of peak is predicted which is later used in PSO. They claimed that, which such an approach, the convergence speed is greatly improved. Gorshy et al [10] applied the quadratic surface method to approximate the fitness function and applied it in the ship design. Seirra and Coello [11] used various types of fitness approximation techniques in multi-objective particle swarm optimisation. Further, Robert et al. [12] used inexpensive surrogate models in multi-objective particle swarm optimisation to reduce the computational cost. He used kriging method for fitness approximation in engineering design problems. Parno et al. [13] has proposed a framework to incorporate the surrogate models into particle swarm optimisation. They showed that surrogate model can be built by using the objective function values that are already available during optimisation process and this model can guide the optimisation algorithm in the search space.

To the best of our knowledge, PSO based on SM has never been used before as meta-models to optimise the parameters for the EA. In this paper, we explore the potentials of PSO-Surrogate as meta search for population-based EAs' parameters.

3. PROPOSED META SEARCH

As mentioned previously, PSO has been used as a top-level search to explore the parameters space for population-based EA, and RBFN surrogate is used to guide the search. Note that the landscape that is defined by the EA's parameters have the same unknown properties as the problem given to the EA itself. An interesting future study is to investigate the relationship between the problem's difficulty that suggested the use of an EA and the sensitivity of the parameters settings.

Here, PSO automatically adjusts the size of population and number of generations in the EA system as well as the search operators' rates in such a way as to keep the whole process under control and facilitate the best possible results. For this task, the EA is associated with a numerical set of weights $W = \{u_0, u_1\}$, which is used to identify its settings. Hence, u_0 is used to set the size of population and number of generations, while u_1 is used to set the search operators' rates. During the optimisation process of the particles, the system applies a constraint to ensure the validity of all settings. Thus, to prevent the PSO from setting a large number for the population size and the number of generations (that potentially may be computationally infeasible), the total number of evaluations should not be allowed to exceed a predefined maximum number of evaluations. Hence,

$$\text{Population\_Size} \times Number\_of\_Generations \leq max\_explorations$$

The "max _explorations" is the upper limit of evaluations for the EA system. The reason why this upper limit of evaluations is set is to allow the user to define an upper boundary of the computational costs before using this framework to tune the parameters. Here, the framework will search for the optimal settings under this boundary.

Each $u_i \in W$ is a real number from the interval $(0, 1)$. EA parameters are calculated as follows:

- $Population {=} max\_explorations \times u_0$
- $Number\_of\_Generations = \frac{max\_explorations}{Population}$
- $Crossover\_rate = u_1$
- $Mutation\_rate = 1 - u_1$

The weights of the EA parameters form a two-dimensional vector $V = \{u_0, u_1\}$, which in turn corresponds to an individual/particle in a swarm population $P_{pso} = \{V_0, V_1, ..., V_n\}$, where n is the size of the swarm. Hence, swarm population (i.e., collection of weights) is used to optimise the parameters of the EA systems in such a way as to find the optimal solution under a limited number of evaluations.

3.1 PSO Parameters Optimisation

The proposed framework works as follows. First, the system randomly initialises a swarm population of size n, according to the constraint mentioned in the previous section. For each particle $V_i \in P_{pso}$ the system runs the EA by using the settings from the V_i. This raises the question of how to evaluate a set of EA's settings? In other words, what it the fitness measure of the PSO. In this work, we run the EA several times and report the best evolved solution across all of runs. The formal notation of the PSO fitness measure is defined as follows:

$$Fitness = \arg\max_{P_{pso}[V_i]} EA(P_{pso}[V_i]), V_i \in P_{pso} \qquad (1)$$

Here, $EA(P_{pos}[V_i])$ refers to the function that runs the EA using the parameters set V_i in the P_{pso} population and returns the fitness of the best solution.

Once the system evaluates all V_i particles in P_{pso}, it allocates the best particle (i.e., the one which provided the best adjustment to the EA in such a way as to return the best solution) as a centre and all other particles in the P_{pso} move toward the centre by using a velocity value $V_i[u_i]$. Velocities were adjusted according to their difference from the best known location, as in [14], for each u_i in V_i particle as follows:

$if(centre_{u_i} < V_i[u_i])$

$$V_i[u_i] = V_i[u_i] - vFactor \times rand_1 \times [|centre_{u_i} - V_i[u_i]|]$$

else if(centre$_{u_i} > V_i[u_i]$)

$$V_i[u_i] = V_i[u_i] + vFactor \times rand_1 \times [|centre_{u_i} - V_i[u_i]|] \quad (2)$$

Thus, each particle in P_{pso} remembers the globally best position (which is found by a member in the flock). Each particle moves into the two-dimensional weights space (or parameters space). The *"vFactor"* is a predefined constant and used to set the max step of any V_i particle in the P_{pso}. Preliminary experiments show that the best value for vFactor is 0.1. The particles optimisation process can be envisaged as birds flock flying in a 2D grid: as each particle moves into the 2D weights space, its weights u_0 and u_1 are adjusted in a simple manner: u_0 moves right or left by adding or subtracting a random amount weighted by the parameter *vFactor* and, similarly, u_1 moves up or down. At each move, the system checks whether the particles' values satisfy the constraint mentioned in Section 3. If the particles do not satisfy these constraints, then a new $rand_1$ value is placed in Equation 2 until the constraint is satisfied. The process of velocity adjustment iterates until the maximum number of moves.

Algorithm 1 broadly outlines the whole process. In **line 1**, the system generates P_{pso}, of size n particles, according to the constraint defined in Section 3. In **line 2**, the system defines the $max\_explore$ limit of evaluations. In **lines 3 - 14**, the system iterates over P_{pso}. For each V_i, it calculates the parameters set, namely, population size, number of generations, and search operators' rates (**lines 6 - 9**). Thereafter, the system runs the EA based on its new settings and stores fitness of the best solution found by the EA into $Best\_Sol\_Fitness$ (**line 11**). In **line 12**, the best V_i treated as a centre and all other individuals in P_{pso} update their velocity values (as explained in Equation 2) in order to move toward the centre (**line 13**). As previously stated, running the EA multiple times can be computationally expensive, therefore we replace (**line 11**) with RBFN surrogate to speed up the PSO search.

The size of the swarm population and number of moves is related to the problem difficulty (more details on the parameters' setting are provided in Section 4).

Algorithm 1: PSO tuning EA settings

```
1  Random-Generate-Swarm(V[$u_0, u_1$], $n$)
2  Integer $max\_explore$;
3  repeat
4      foreach $V_i$ do
5          Parameters_Set:
6          { Pop_Size = $u_0 \times max\_explore$;
7          Number_of_Gen = $max\_explore / Pop\_Size$;
8          Crossover = $u_1$;
9          Mutation = $1 - u_1$;
10         }
           // Here we replace the real
           evaluation with a cheap RBFN
           surrogate
11         Best_Sol_Fitness = Run_EA(Parameters_Set);
12     $centre_{best}[u_0, u_1] = Best\_Sol\_Fitness$
13     Update-All($V_0, \ldots, V_n$)
14 until max-moves ;
```

3.2 Radial Basis Function Networks Surrogate

There are a number of known approaches to learn a function that belongs to a certain class of functions from existing data-points [1] (i.e., finding a function in that class that interpolates and best fits the data-points according to some criteria). Some of these include Genetic Programming (GP), RBFN Interpolation, Artificial Neural Networks and Gaussian Process Regression (also known as Kriging) [4].

GP is a powerful method for approximating unknown functions. However, one major drawback of GP is its well-known expensive learning process. Gaussian Process Regression is a very powerful method with a solid theoretical foundation, which not only can make a rational extrapolation about the location of the global optimum, but also gives an interval of confidence about the prediction made. RBFN Interpolation is conceptually simpler than Gaussian Process Regression and can extrapolate the global optimum from the known data-points. In this paper, we focus on RBFNs as surrogate models.

[1]Data-point is the pair of solution and its real fitness value.

RBFNs can be seen as variants of an artificial neural network that uses radial basis functions (RBF) as activation functions [15]. Typically, RBFN consists of three layers: input, hidden, and output layer. The relationship between the input and the hidden layer is determined by the RBF activation function. The nodes in the output layer usually perform a simple summation for the linear weights of these activations. RBFNs have successfully been used in function approximation, time series prediction, and control [15].

A RBF is a real-valued function $\phi : \mathbf{R}^n \to \mathbf{R}$; its value depends only on the distance from a known point in the search space c, called *centre*, so that $\phi(V_q) = \phi(\|V_q - c\|)$. The point c is a parameter of the function and the point V_q is the query point in the PSO to be estimated. The norm is usually Euclidean, so $\|V_q - c\|$ is the Euclidean distance between c and V_q.

There are several types of RBF functions, including: Gaussian, Multiquadric, Inverse Quadratic and Inverse Multiquadric. In this paper we use the Gaussian function of the form:

$$\phi(V_q) = \exp(-\beta \|V_q - c\|^2)$$

where $\beta > 0$ is the width parameter. Radial basis functions are typically used to build function approximations of the form:

$$y(V_q) = w_0 + \sum_{i=1}^{N} w_i\, \phi(\|V_q - c_i\|) \quad (3)$$

Thus, $y(V_q)$ is used to approximate the real-objective function, (i.e., running the EA multiple times with particular parameters set in our case), when evaluating a PSO particle. The approximating function $y(V_q)$ is represented as a sum of N radial basis functions, each associated with a different centre c_i, a different width β_i, and weighted by an appropriate coefficient w_i, plus a bias term w_0. In principle, any continuous function can be approximated with arbitrary accuracy by a sum of this form, if a sufficiently large number N of radial basis functions is used [15]. The bias w_0 can be set to the mean of the values of the training set fitnesses that are used to train the surrogate model, or set to 0.

Training the RBFNs requires three parameters to be found: *(a)* the centres c_i, *(b)* the values of w_i in such a way that the predictions on the training set minimises the errors and, finally, *(c)* the RBF width parameter β.

The centres can be chosen to coincide with the training set and evaluate them with the real fitness evaluation. The β value can be either fixed for all N linear RBFs (global) or it can be customised for each RBF (local). In this work, we set the β value as $1/D^2$ where D is the maximum pairwise distance between the query point and all of the points in the training set. The value of β controls the radius of each RBF (spreading on the search space to cover all the other centres), so that each known function value at a centre can potentially contribute significantly to the prediction of the function value of any point in space.

Finally, the weights vector can be calculated by solving the system of N simultaneous linear equations in w_i by requiring that the unknown function exactly predicts the training. Formally, we have:

$$y(V_i) = b_i,\ i = 1 \dots N.$$

Setting $g_{ij} = \phi(||\mathbf{V}_j - \mathbf{V}_i||)$, the system can be written in a matrix form as $\mathbf{Gw} = \mathbf{b}$ where $\mathbf{b}$ is a vector of the true fitness values of the data-points that have been used to train the surrogate. The matrix $\mathbf{G}$ is non-singular, because we guarantee that the points V_i are distinct, so the weights w can be solved by simple linear algebra:

$$\mathbf{w} = \mathbf{G}^{-1}\mathbf{b}$$

The value of the bias term w_0 in Equation 3 is set to the mean value of the training set real fitnesses, i.e., the mean of vector $\mathbf{b}$. In this way, the predicted function value of a point which is out of reach of the influence of all centres is by default set to the average of their function values.

Once the RBF parameters are determined, the model is ready to estimate the fitness of any unseen point. Thus, the fitness $EA(P_{pos}[V_q])$ (defined in Equation 1) of a query point V_q in the search space is predicted by weighted linear combination of:

$$EA(P_{pos}[V_q]) \approx w_0 + \textstyle\sum_{i=1}^{N} [w_i * \phi(d(V_q, c_i))]$$

where, w_i is a vector of weights that has been calculated during the training phase and ϕ is the kernel function which is defined in Equation 3. Finally, $d(V_q, c_i)$ is the distance between the new point V_q and the training set points c_i.

3.3 Surrogate Implementation

Similar to traditional procedure of surrogate model based optimisation (SMBO) [4], the framework constructs an initial surrogate model by using the real fitness measure (defined in Equation 1) on a small set of solutions and replaces the real fitness measure in Algorithm 1 - **line 11**. The remaining expensive objective function evaluations out of a limited budget are applied to candidate solutions in which the surrogate model predicts to have promising performance. The process interleaves search of the surrogate model to obtain its optimum, evaluation of the optimum solution of the model by using the expensive objective function, and update of the surrogate model with the new point.

Here, we used PSO to optimise the surrogate model. We refer to this version of the PSO as *cheap PSO*, because it uses a computationally inexpensive surrogate model. The best solution that the cheap PSO has found is then compared with the best point of the training set (that are used to train the surrogate). If the newly suggested point is better than than the best existing point in the training set, then the system successfully managed to extrapolate from the training set. It updates the training set with the new point and re-trains the surrogate. Otherwise, the system adds a new random point to the training set, in a step to collect more information about the under-sampled search space. This procedure coincides with our main hypothesis that RBFN surrogate can successfully approximate the parameters search space.

Note that the role of surrogate is to infer the location of a promising solution, and it is not directly applied to the original problem with the expensive objective function. This is feasible because the computational cost of a complete run of the EA on the surrogate model is negligible (i.e., in the order of few seconds) with regard to the cost of evaluating a solution using the expensive objective function of the problem (in the order of minutes, hours, or even days, depending on the problem).

4. EXPERIMENT SETUP

Experiments have been conducted in order to validate our framework. The main aim of the experiments is to evaluate the performance of the framework and to assess its behaviour under a variety of circumstances. In principle, the proposed framework can optimise the parameters of any population-based EA. In this paper, we felt that GA is a good example with which to demonstrate the capabilities of our framework. The experiments included testing the framework on three different problems, namely, NK-Landscape [16] (unimodal problem), Hamming Centres [17], (multimodal problem), and three different continuous optimisation problems. For each problem we tested all systems under different chromosomes size m.

We compared the framework in terms of the best evolved solution with *(a)* Standard PSO without surrogate assistance to optimise the GA parameters, in order to verify whether surrogate can acutely find the best setting within a limited number of explorations in the parameters space, *(b)* Standard GA applied directly to the problem with fixed parameters settings to check whether parameters tuning has added any extra advantages to the search, *(c)* Standard 1+1 Evolutionary Strategy (ES) applied directly to the problem, to compare our framework performance against a standard search algorithm, and, finally, *(d)* simple Random parameters generator, to prove that surrogate sampling is better than random search in suggesting data-points in the search space.

Since the problem complexity is related to the the size of the GA chromosome (i.e., m value), here, we relate the value of $max\_explorations$ (the upper limit of evaluations as explained in Section 3) to problem complexity. Thus, $max\_explorations = m^2$. So, essentially our aim is to find the best solution to the problem the algorithm can produce in linear time out of an exponential number of candidate solutions in the problem space. Table 1 illustrates the settings used in our experiments for standard GA with fixed parameters and 1+1 ES.

Table 1: Settings used in the experiments

Operator	*Standard GA with fixed settings*	1+1 ES
Mutation	30%	100%
Crossover	70%	0%
Tournament size	2	N/A
Population Size	m	1
Generations	m	m^2

As for the PSO without surrogate assistance, the size of P_{pso} is equal to $\frac{m \times 2}{10}$ and the number of moves for each particle in the swarm is 5. Thus, each particle in the P_{pso} effectively evaluates 5 different parameters sets in a process of tuning the GA parameters. Note that the total number of evaluations performed by the GA in any settings of $V_i \in P_{pso}$ will not exceed the $max\_explorations$ value. Each particle runs the GA 10 times to validate the goodness of the V_i parameters set. Thus, each particle will effectively run the GA 10×5 times. Hence, to allow a fair comparison, when each individual in the P_{pso} runs the GA systems, we run both standard GA with fixed settings and 1+1 ES exactly 10×5. Obviously, this approach is not completely fair because the PSO, here, has the advantage of exploring different parameters sets while both standard GA and 1+1 ES searches the problem using a fixed parameters. However, it is also unfair to give them fewer number of problem evaluations than PSO. There is no clear method of comparing these algorithms.

Our proposed framework, PSO that uses a RBFN surrogate, received exactly the same number of parameters explorations $\frac{m \times 2}{10} \times 5$, and the same PSO fitness function to measure the parameters sets (i.e., running the GA 10 times and report the best fitness). The difference here is that PSO particles are evaluated using cheap RBFN surrogate rather than a complete real fitness measure (i.e., running the GA 10 times with each parameters set). Therefore, we were generous with the the cheap PSO settings. The number of particles in the cheap PSO is $10m$ and number of moves was set also to $10m$. Once the surrogate suggests a new promising point in the parameters space, it evaluates it using the real fitness measure and updates the surrogate model. Our aim is to utilise the RBFN surrogate to optimise the GA parameters using very small number of explorations in the parameters space (i.e., $5\frac{2m}{10} = m$)

5. RESULTS

5.1 Continuous Optimisation

In our experiments, we evaluated the framework using three different well-known benchmark problems. Namely, Rastrigin function, Dixson & Price function and, finally, Michalewics function [18]. The reason for choosing these three functions in particular is because they represent different landscapes with different levels of difficulty. The Rastrigin function has the overall structure of a hyper-parabola with many bumps, whereas the other two functions have a smoother, albeit deceptive, landscape (i.e., the best optima are on different sides of the search space). For each function, we investigate the performance of the system under $m = 20, 30, 40$, and 50. Each system was tested using 20 independent runs. Table 2 summarises the results of 1200 runs (20 independent runs for each system for each m value). It is clear that our framework comes in the first place in almost all cases with significant performance margins. The model failed to outperform the standard PSO only in two cases (when $m = 20$ and 30 in the Michalewics function). Note the the Std of our framework shows different levels of stability. This outcome is expected because PSO particles are searching for the best settings for the GA in the parameters space. Thus, depending on the PSO's initial population, each V_i particle explores different area in the search space, and thus, each V_i tries different settings which is not necessary to the best in a process of optimising the solution. However, in standard GA and 1+1 ES we used the same settings in all runs and thus, they tend to have more stability as shown by their Std.

5.2 NK-Landscape

NK-Landscape was established by Stuart Kauffman in [19]. The fitness of a gene could depend solely on its own state. At its most complex, the fitness of a gene could depend on the state of all others. Thus, there are N genes in which each gene's fitness depends of the state of K others.

We investigated the performance of our framework under different values of N. Namely, we used $N = 20$, 30, and 40. For each N value we tested three different K values $K = \frac{N}{5}$ *(easy problem)*, $\frac{N}{2}$ *(hard problem)*, and $\frac{N}{2} + 5$ *(very hard problem)*. [2] For each N, K combination we tested the system using 20 independent runs using the settings mentioned in Section 4.

Table 3 summarises the results of 900 independent runs (20 independent runs for each system for each N,K combination). As can be seen in the table, PSO-Surrogate managed to adjust the GA parameters in a way that makes the search achieves better results. It is clear that PSO-Surrogate is ranked first place both in terms of average (i.e., average of best solutions in the 20 runs) and best (i.e., best achieved solution across the entire 20 runs) in almost all test cases. The only time PSO-Surrogate did not manage to outperform PSO (that tunes the parameters directly without surrogate assistance) and Standard GA was when the problem was too easy ($N = 20$ and $K = 4$) so all algorithms can perform equally well.

5.3 Hamming Centres

Hamming centres is an NP-hard problem defined in [17] as follows. Lets a set S of k_i binary strings, where $i \in \{1, 2...I\}$, each of length m, and r is a positive integer. The objective is to find $m-$bits string y such that for every string k_i in S, the Hamming distance, $H(k_i, y) \leq r$.

We investigated the performance of our proposed framework under different values of $m-$bits ($m = 50$, 75, 100, and 125). For each m value we performed 20 independent runs for each system. The size of the set S was 20 in all experiments. The fitness value was measured as the number of cases that string y satisfied the condition of $H(k_i, y) \leq r$. Thus, the optimal solution is equal to the size of S (which is 20 in our case). The r value was set to $\sqrt{m}$. Table 4 illustrates the results of 400 runs (20 independent runs for each system for each m value). Clearly, PSO-Surrogate comes in first in all test cases in both average fitness (in 20 runs) and best fitness across all runs. Surprisingly, standard GA with fixed parameters settings outperformed PSO (applied directly to tune the GA parameters) in two test cases, namely, when $m = 100, 125$. This suggests that the allowed explorations in the parameters space is very low (i.e., m trials only where m is the GA chromosomes size) and it is not enough to allow PSO to converge to an optimum solution in this particular problem. This is also an indication that the use of SM has assisted the PSO in exploring potential areas in the parameters space.

6. DISCUSSION

It is a well-known fact in the field of optimisation that it is very difficult to determine an efficient optimisation algorithm for all types of functions of optimisation problems. The same is true for our proposed PSO-surrogate method. We have reported optimisation results for three different types of functions,including three continuous functions, Hamming centre and NK landscape. We have used RBFN as surrogate model which is well suited for approximation of continuous functions. Hence as expected our proposed algorithm has shown better results for all three continuous functions. In Table 5, improvement percentage is reported

[2]In [20] the authors provided an indication of NK-landscape hardness under different settings.

Table 2: Summary of 1200 **runs of Continuous Optimisation (20 independent runs for each system for each** m **size)**

	Mean	*Best*	*Std*	*Mean*	*Best*	*Std*	*Mean*	*Best*	*Std*	*Mean*	*Best*	*Std*
Dixon&Price Function												
	m=20			*m=30*			*m=40*			*m=50*		
PSO-Surrogate	**14063.7**	**15481.6**	795.8	**33328.1**	**35830.8**	2462.3	**63745.9**	**66033.8**	1709.9	**100863.8**	**105515.0**	3858.6
PSO	13428.1	14239.8	542.4	32487.1	35176.9	1999.5	60723.8	64666.9	2187.2	93936.8	103205.0	4462.5
Standard GA (Fixed settings)	12039.4	12919.7	579.1	27963.9	29972.9	1231.5	51728.3	55155.3	1232.8	82155.0	86391.2	1812.6
1+1 ES (Fixed settings)	10484.8	12215.1	775.0	22915.5	25598.9	903.3	39411.4	41936.5	1444.7	59656.4	64101.2	2416.6
Random Parameter search	9891.2	10851.6	496.8	21815.4	23319.3	886.7	38174.9	46156.5	2346.8	57476.1	60739.9	1596.5
Michalewics Function												
	m=20			*m=30*			*m=40*			*m=50*		
PSO-Surrogate	**8.7**	**10.5**	*0.9*	**13.8**	16.4	1.8	**22.3**	**25.2**	2.4	**29.2**	**33.7**	4.3
PSO	8.5	**10.5**	0.9	13.0	**17.0**	1.7	17.8	23.2	2.8	22.9	31.9	4.5
Standard GA (Fixed settings)	6.0	7.1	0.7	10.1	12.3	1.1	14.1	15.7	0.9	18.3	20.0	0.9
1+1 ES (Fixed settings)	4.4	5.1	0.4	6.2	7.3	0.6	7.1	9.0	0.7	8.6	11.0	0.9
Random Parameter search	4.1	5.5	0.6	5.4	7.3	0.7	6.5	7.3	0.5	7.9	9.9	0.9
Rastrigin Function												
	m=20			*m=30*			*m=40*			*m=50*		
PSO-Surrogate	**518.0**	**598.1**	*52.6*	**838.6**	**920.4**	60.9	**1142.9**	**1260.0**	114.1	**1466.8**	**1576.8**	139.3
PSO	478.0	552.3	36.0	748.5	875.4	92.5	931.3	1156.4	144.8	1247.7	1532.5	197.7
Standard GA (Fixed settings)	374.7	490.1	69.3	547.7	639.5	64.4	789.9	854.4	40.4	998.1	1142.0	71.2
1+1 ES (Fixed settings)	248.4	313.2	36.5	286.2	403.8	51.5	269.7	441.0	73.2	287.1	398.5	62.8
Random Parameter search	221.0	317.1	42.3	215.5	329.1	48.3	217.5	329.8	57.7	181.8	305.3	53.9

*__Bold__ numbers are the highest.

Table 3: Summary of 900 runs of NK-Landscape (20 independent runs)

Algorithm	*Mean*	*Best*	*Std*	*Mean*	*Best*	*Std*	*Mean*	*Best*	*Std*
	N20 K4			*N20 K10*			*N20 K15*		
PSO-Surrogate	**0.757**	0.775	0.017	**0.739**	**0.778**	0.017	**0.737**	**0.771**	0.013
PSO	0.743	0.775	0.020	0.737	0.770	0.015	0.729	0.756	0.019
Standard GA (Fixed settings)	0.725	0.775	0.025	0.714	0.755	0.023	0.687	0.750	0.018
1+1 ES (Fixed settings)	0.709	0.752	0.017	0.726	0.745	0.014	0.719	0.737	0.012
Random Parameters search	0.679	0.732	0.025	0.681	0.711	0.017	0.689	0.748	0.027
	N30 K6			*N30 K15*			*N30 K20*		
PSO-Surrogate	**0.766**	**0.798**	*0.019*	**0.724**	**0.760**	0.016	**0.710**	**0.757**	0.018
PSO	0.762	0.782	0.012	0.719	0.743	0.016	0.708	0.734	0.015
Standard GA (Fixed settings)	0.733	0.772	0.018	0.693	0.724	0.017	0.687	0.727	0.019
1+1 ES (Fixed settings)	0.697	0.721	0.011	0.701	0.728	0.014	0.698	0.731	0.011
Random Parameters search	0.670	0.700	0.014	0.673	0.708	0.017	0.663	0.697	0.015
	N40 K8			*K40 K20*			*N40 K25*		
PSO-Surrogate	**0.769**	**0.791**	*0.009*	**0.717**	**0.746**	0.014	**0.706**	**0.734**	0.013
PSO	0.767	0.782	0.008	0.716	0.731	0.010	0.702	0.724	0.012
Standard GA (Fixed settings)	0.721	0.757	0.014	0.689	0.713	0.016	0.683	0.714	0.015
1+1 ES (Fixed settings)	0.685	0.703	0.010	0.684	0.736	0.016	0.686	0.704	0.007
Random Parameters search	0.653	0.720	0.023	0.652	0.674	0.011	0.648	0.678	0.015

***Bold** numbers are the highest.

for all functions. Improvement in percentage is calculated between PSO-surrogate and second best result. For Michalewics and Rastrigin functions, optimisation results are improved by more than 20%. In Table 5, an increasing trend of improvement in the case of continuous functions is evident as the complexity of functions increase. This observation is very encouraging and makes our proposed algorithm a good candidate for very complex optimisation problems. Dixon & Price function is a unimodal function. Hence, all the algorithms have performed well and we have seen a maximum of 7% improvement by using PSO-surrogate method. But Michalewics and Rastrigin functions are multimodal functions and hence optimising these function is difficult. In these functions, our PSO-surrogate method outperformed all other algorithms by a wide margin.

In case of Hamming centre problem, our proposed algorithm has shown good improvement of about 7% for cases of $m = 75$ to $m = 125$. Hamming centre problem is a binary problem. Although the improvement is considerable but it is less than the continuous functions.

Hence, in order to get the best out of our proposed method, we have to select the type of surrogate model which is well suited for function approximation in a particular type of optimisation problem.

7. CONCLUSION

This paper proposes a framework for automatically tuning population-based EA. The proposed framework is comprised of two components; PSO that searches the parameters space and a RBFN surrogate model to guide it. The main advantage of our model is that it optimises the EA parameters in a way to ensure that EA searches the problem within a limited number of evaluations. We use PSO as a top-level search to explore the parameters space for population-based EA, and RBFN surrogate is used to guide the search. The framework tune the population size, number of generation and search operators rates.

In our experiments we have used GA as an example of population-based EA to demonstrate our framework capabilities. Experiments with three different benchmark problems, namely, NK-landscape, Hamming Centres and three different continuous optimisation problems demonstrate that our proposed framework managed to assist the GA to optimise its parameters and achieves better solutions than, using Standard PSO without surrogate assistance to optimise the GA parameters, Standard GA applied directly to the problem with fixed parameters settings, Standard 1+1 Evolutionary Strategy (ES) applied directly to the problem and simple Random Search. The results confirms that parameter tuning is useful and a key factor of the success of the search. In addition, our experiments confirm that the use of SM has assisted the PSO because it makes sense of the available information and point out promising areas in the parameters space. Thus, our framework was able to successfully optimise the parameters using a very small number of exploration in the parameters space.

There are many directions where we can extend this research. As mentioned previously, an interesting future study is to investigate the relationship between the problem's difficulty (that suggested the use of an EA) and the sensitivity of the parameters settings. We would also like to explore the model behaviour with different EA representations (e.g., optimising the parameters of GP) as each EA has different level of parameters sensitivity to its performance. Another direction of this research is to allow the model to tune the parameters of the EA during the run as to maintain its diversity and prevent the premature convergence problem.

8. REFERENCES

[1] K. DeJong, “Parameter setting in eas: a 30 year perspective.” in *Parameter Setting in Evolutionary Algorithms*, ser. Studies in Computational Intelligence, F. G. Lobo, C. F. Lima, and Z. Michalewicz, Eds. Springer, 2007, vol. 54, pp. 1–18.

[2] M. E. Samples, J. M. Daida, M. Byom, and M. Pizzimenti, “Parameter sweeps for exploring GP parameters,” in *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, vol. 2. Washington DC, USA: ACM Press, 2005, pp. 1791–1792.

[3] T. Bäck and H. Schwefel, “An overview of evolutionary algorithms for parameter optimization,” *Evolutionary computation*, vol. 1, no. 1, pp. 1–23, 1993.

Table 4: Summary of 400 **runs of Hamming Centres (20 independent runs)**

Algorithm	*Mean*	*Best*	*Std*
m = 50			
PSO-Surrogate	**8.600**	**10.000**	0.883
PSO	8.350	9.000	0.587
Standard GA (Fixed settings)	6.500	8.000	0.827
1+1 ES (Fixed settings)	4.500	6.000	0.607
Random Parameter search	3.400	4.000	0.503
m = 75			
PSO-Surrogate	**10.950**	**12.000**	*0.759*
PSO	10.100	11.000	0.718
Standard GA (Fixed settings)	9.450	11.000	0.759
1+1 ES (Fixed settings)	5.250	6.000	0.444
Random Parameter search	4.150	5.000	0.366
m = 100			
PSO-Surrogate	**14.000**	**12.905**	*0.700*
PSO	13.000	11.286	0.956
Standard GA (Fixed settings)	13.000	11.857	0.793
1+1 ES (Fixed settings)	7.000	6.000	0.447
Random Parameter search	6.000	5.095	0.625
m = 125			
PSO-Surrogate	**15.000**	**13.048**	*1.244*
PSO	13.000	11.238	1.221
Standard GA (Fixed settings)	14.000	11.619	1.322
1+1 ES (Fixed settings)	6.000	5.238	0.436
Random Parameter search	5.000	4.333	0.577

***Bold** numbers are the highest.

Table 5: Improvement in percentage

Problem			
Chromosome Size	*Continuous Optimisation*		
	Discon&Price	*Michalewics*	*Rastrigin*
m=20	4.7%	2.4%	8.4%
m=30	2.6%	6.2%	12%
m=40	5.0%	25.3%	22.8%
m=50	7.4%	27.5%	17.6%
	Hamming Centres		
m=50	3%		
m=75	8.4%		
m=100	7.7%		
m=125	7.1%		
	NK-Landscape		
NK(20,4)	1.9%	NK(30,15)	0.7%
NK(20,10)	0.3%	NK(30,20)	0.3%
NK(20,15)	1.1%	NK(40,8)	0.3%
NK(30,6)	0.5%	NK(40,20)	0.14%
NK(40,25)	0.6%		

[4] A. Forrester, A. Sóbester, and A. Keane, *Engineering design via surrogate modelling: a practical guide*. Wiley, 2008, vol. 226.

[5] A. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 3, no. 2, pp. 124–141, 1999.

[6] S. Smit and A. Eiben, "Comparing parameter tuning methods for evolutionary algorithms," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009, pp. 399–406.

[7] M. Preuss, G. Rudolph, and S. Wessing, "Tuning optimization algorithms for real-world problems by means of surrogate modeling," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 401–408.

[8] G. Rudolph, M. Preuss, and J. Quadflieg, "Two-layered Surrogate Modeling for Tuning Optimization Metaheuristics," Faculty of Computer Science, Algorithm Engineering (Ls11), Technische Universität Dortmund, Germany, Algorithm Engineering Report TR09-2-005, Sep. 2009.

[9] S. Bird and X. Li, *Improving local convergence in particle swarm by fitness approximation using regression*. Springer Berlin Heidelberg, 2010.

[10] H. Gorshy, X. Chu, L. Gao, and P. Li, "An approach combined response surface method and particle swarm optimization to ship multidisciplinary design and optimization," in *Industrial Engineering and Engineering Management, 2009. IEEM 2009. IEEE International Conference on*. IEEE, 2009, pp. 1810–1814.

[11] M. Reyes-Sierra and C. Coello, "A study of fitness inheritance and approximation techniques for multi-objective particle swarm optimization," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 1, 2005, pp. 65–72.

[12] R. Carrese, A. Sóbester, H. Winarto, and X. Li, "Swarm heuristic for identifying preferred solutions in surrogate-based multi-objective engineering design," *AIAA journal*, vol. 49, no. 7, 2011.

[13] M. Parno, K. Fowler, and T. Hemker, "Framework for particle swarm optimization with surrogate functions," Technical Report TUD-CS-2009-0139, Department of Computer Science, Technische Universität Darmstadt, Tech. Rep., 2009.

[14] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4. IEEE, 1995, pp. 1942–1948.

[15] A. G. Bors, "Introduction of the radial basis function (rbf) networks," Department of Computer Science, University of York, UK, Tech. Rep., 2001.

[16] S. A. Kauffman, *The Origins of Order – Organization and Selection in Evolution*. New York: Oxford University Press, 1993.

[17] M. Frances and A. Litman, "On covering problems of codes," *Theory of Computing Systems*, vol. 30, pp. 113–119, 1997, 10.1007/BF02679443.

[18] M. Molga and C. Smutnick, "Test functions for optimization needs," *Test functions for optimization needs*, 2005.

[19] S. Kauffman, *The origins of order*. Oxford University Press New York, 1993, vol. 209.

[20] T. Jones and S. Forrest, "Fitness distance correlation as a measure of problem difficulty for genetic algorithms." in *ICGA*, L. J. Eshelman, Ed. Morgan Kaufmann, 1995, pp. 184–192.