Polynomial Approximation Using Set-Based Particle Swarm Optimization

Jean-Pierre van Zyl[1](✉) and Andries P. Engelbrecht[1,2]

[1] Division of Computer Science, Stellenbosch University, Stellenbosch, South Africa
{20706413,engel}@sun.ac.za
[2] Department of Industrial Engineering, Stellenbosch University, Stellenbosch, South Africa

Abstract. This paper introduces a new approach to solving regression problems by using a particle swarm optimization algorithm to find optimal polynomial regressions to these problems. Polynomial regression is defined as a multi-objective optimization problem, with the goals to find both an optimal combination of terms in the polynomial and optimal values of the coefficients of the terms, in order to minimize the approximation error. This paper shows that a set-based PSO works well to find the optimal term structure of the target polynomials in low dimensions, and holds promise for improved performance in higher dimensions. The results of the set-based PSO are compared to the results of a Binary PSO on the same problems. Finally, this paper explores possible solutions to create a hybrid algorithm that can find both the optimal term structure and the coefficients of the found terms.

Keywords: Particle swarm optimization · Polynomial regression · Adaptive coordinate descent · Set-based particle swarm optimization

1 Introduction

A polynomial is a functional mapping, $f : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$, relating an n_x-dimensional input space to a one-dimensional output space. Polynomial regression refers to the process of finding an optimal polynomial that accurately approximates an arbitrary functional mapping. While a number of approaches exist, this paper develops a novel set-based optimization approach to find polynomial mappings.

Polynomial regression is here defined as a multi-objective optimization problem, using a set-based solution representation. The objectives are to find: (1) the smallest number of terms and lowest polynomial order, and (2) optimal coefficient values for these terms in order to minimize the approximation error.

This paper determines the viability of using a set-based particle swarm optimization (SBPSO) algorithm to find an optimal term set in order to achieve the first objective. As a precursor to future improvements to this approach, in order to meet the second objective, the suitability of an interleaved, dual optimization process is investigated to find both optimal term architecture and coefficients.

This is achieved in a preliminary study by applying adaptive coordinate descent (ACD) [10] to find the coefficients of the found term sets' components. To the knowledge of the authors, this a first approach to polynomial regression using a set-based optimization algorithm.

The SBPSO algorithm is empirically evaluated on a number of problems to determine its ability to select optimal combination of terms, and is compared to a binary particle swarm optimization (BPSO) [7] algorithm's ability to select terms. The combined algorithm with ACD is compared to a standard non-set based particle swarm optimization (PSO) [6] algorithm. SBPSO is shown to be able to find an optimal set of terms by itself, and the preliminary results of the proposed hybrid algorithm shows that it is also able to find an optimal set of terms and optimal coefficients.

It is shown that the SBPSO and ACD hybrid algorithm performs well when applied on low dimensional problems and hold promise for improvement in higher dimensions. The hybrid algorithm is able to approximate the source polynomial from the input data both in structure and in coefficients.

Section 2 discusses the concepts needed to implement the work in this paper, and Section 3 outlines how existing optimization algorithms can be combined to approximate polynomial mappings. Section 4 contains the empirical procedure, while Section 5 discusses the results followed by the conclusion in Section 6.

2 Background

This section outlines background information on polynomial regression, PSOs, SBPSOs, and ACD as used in this paper.

2.1 Polynomial Regression

Polynomials are made of constituent parts called terms or monomials. These monomials are defined as the product of one or more input variables, each raised to a power and preceded by a coefficient:

$$a_i \prod x_j^n \tag{1}$$

The goal of polynomial regression is to find the best possible polynomial to accurately approximate a functional mapping, $f : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$, embedded in a data-set, $D = \{(\boldsymbol{x}_p, y_p) | p = 1, \ldots, n_p\}$; where $\boldsymbol{x}_p = (x_{1p}, x_{2p}, \ldots, x_{n_x p})$ is a vector of input variables, y_p is the corresponding desired output value, p refers to a specific data point in D, n_x is the number of input variables, and $n_p = |D|$ is the total number of data points.

Univariate polynomials have $n_x = 1$, and are presented in the general form:

$$f(x) = \sum_{j=0}^{n_o} a_j x_j = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n_o} x^{n_o} \tag{2}$$

where n_o is the order of the polynomial. Multivariate polynomials have $n_x > 1$, and have the general form:

$$f(\boldsymbol{x}) = a_0 + \sum_{t=1}^{n_t} a_t \prod_{q=1}^{n_q} x_q^{\lambda_q} \tag{3}$$

where n_t is the number of monomials, a_t is the coefficient of the t^{th} monomial, n_q is the number of variables in the t^{th} monomial, and λ_q is the order of the corresponding variable.

The goal of finding the best polynomial approximation can be broken down into the following sub-goals: (1) to find optimal monomials, (2) to find the smallest number of monomials, (3) to minimize the order of the monomials, and (4) to find the best coefficient values of these monomials.

The rationale of these sub-goals is to produce a polynomial that minimizes the approximation error and the complexity of the polynomial. The structure of the polynomial is minimized to prevent overfitting, while underfitting is prevented by minimizing the approximation error. Approximation error is estimated using the mean squared error (MSE), defined as

$$\mathcal{E} = \frac{1}{n_p} \sum_{p=1}^{n_p} (y_p - \hat{y}_p)^2 \tag{4}$$

Polynomial approximation is a multi-objective optimization problem, defined as:

$$\text{minimize } F(f(\boldsymbol{x}), D) = \mathcal{E}(f(\boldsymbol{x}), D) + \lambda P(f(\boldsymbol{x})) \tag{5}$$

where $f(\boldsymbol{x})$ is a polynomial from the universe, $\mathcal{U}$, of possible polynomials, D is the data-set of points, $\mathcal{E}$ is the MSE, P is a polynomial complexity penalty function, and λ is a penalty coefficient. An example penalty function is

$$P(f(\boldsymbol{x})) = \sum_{i=0}^{n_t} a_i^2 \tag{6}$$

referred to as ridge regression, or weight decay in neural network terminology [8].

Polynomial regression is a commonly performed task in model induction and machine learning in general and, as a result, various approaches have been tested. Notably, neural networks (NN) have been used for polynomial regression [15] and have been shown to be universal approximators capable of learning any non-linear mapping [5]. However, the output of a NN is not the target polynomial itself, but an uninterpretable list of tuned weights.

2.2 Particle Swarm Optimization

Particle swarm optimization is a well-established swarm-based optimization method [6]. Since its inception, many modifications have been proposed to improve its performance and its application on different problem types. Modifications for discrete environments include the BPSO or the angle modulated PSO [12].

Basic Particle Swarm Optimization. The first PSO, proposed by Kennedy and Eberhart [6], is a swarm-based optimization algorithm that makes use of stochastic optimization techniques inspired by the flocking behaviour of birds. The population of a PSO is called a swarm, and each agent in the swarm is known as a particle. Each particle represents a candidate solution to the optimization problem. These potential solutions are changed to explore the search landscape and attempt to exploit any potential optima that have been found in the process.

In the PSO algorithm, let n_s denote the swarm size, and n_x denote the dimensionality of the problem. Each particle i has a position $\boldsymbol{x}_i(t)$, a velocity $\boldsymbol{v}_i(t)$, a personal best position $\boldsymbol{y}_i(t)$, and a neighbourhood best position $\hat{\boldsymbol{y}}_i(t)$, with each variable being n_x-dimensional vectors. The personal best position is the best optimum discovered by particle i up to iteration t, and the neighbourhood best is the best optimum discovered by any particle in particle i's neighbourhood. Particle positions are updated in each iteration using:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{7}$$

where $v_{ij}(t)$ is the velocity, calculated for each dimension j using [13]:

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)] \tag{8}$$

where ω is the inertia weight, c_1 and c_2 are the acceleration coefficients and $r_{1j}(t) \sim U(0,1)$ and $r_{2j}(t) \sim U(0,1)$ are uniformly distributed random variables for all $i \in \{1,\ldots,n_s\}$ and $j \in \{1,\ldots,n_x\}$.

The control parameters ω, c_1 and c_2 control the exploration-exploitation trade-off in PSOs. This trade-off is adjusted to determine whether the goal of the swarm is to discover new potential solutions or to refine already found optima.

Binary Particle Swarm Optimization. While PSOs were initially developed for continuous search spaces, the binary PSO (BPSO) variant was developed by Kennedy and Eberhart to solve binary problem spaces [7].

The BPSO has a structure similar to the standard PSO, with its velocities still being defined by Eq. (8) in continuous space. However, the velocities are not interpreted as a spatial change in $\mathbb{R}^{n_x}$ space, but as probabilities of bit flips. The position vector is changed to consist of bits, *i.e.* each $\boldsymbol{x}_i \in \mathbb{B}^{n_x}$, and the position update equation is defined as:

$$x_{ij}(t+1) = \begin{cases} 1 & \text{if } r_{3j}(t) < S(v_{ij}(t+1)) \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where $S(v_{ij}(t)) = \frac{1}{1+e^{-v_{ij}(t)}}$ and $r_{3j}(t) \sim U(0,1)$.

Set-Based Particle Swarm Optimization. The set-based PSO, as implemented in this paper, was developed to solve the multi-dimensional knapsack problem [9]. This is a discretised version of the standard PSO which makes use of a set-based search space instead of a n_x-dimensional continuous search space. Particle positions consist of elements from the universal set, $\mathcal{U}$, while the velocity is a set of operation pairs which add to or remove from elements in the position. The set-based representation allows for candidate solutions of various dimensions (a variable number of components), contrary to the basic PSO where all candidate solutions have to be of the same dimension. This optimization algorithm has been successfully applied to real-world problems like portfolio optimization and feature selection [2,3], and performs well in discrete search spaces.

Because there is no concept of spatial structure for a set-based representation, analogies of the velocity and position update equations were developed by Langeveld and Engelbrecht [9]. These new equations contain operators to calculate the attraction to the global best position and to each particle's personal best position as seen in the standard PSO. There is also an operator to add unexplored terms to the position and an operator to remove possibly poorly selected terms. For more detail on the SBPSO and its position and velocity update equations, the reader is referred to [9].

A brief description of the important control parameters follows: Coefficient c_1 controls the particle's attraction to its own previous personal best position, while c_2 controls its attraction to the neighbourhood (global) best position up to iteration t. The additional acceleration coefficients, c_3 and c_4, manage the effect of the operators designed to improve exploration of the search space. The number of terms added to a position is controlled by c_3, and the number of terms from a position is controlled by c_4.

2.3 Adaptive Coordinate Descent

Adaptive coordinate descent (ACD) [10] is an improvement to the covariance matrix adaptation evolutionary strategy (CMA-ES). ACD adds adaptive encoding (AE), developed by Hansen [4], to the coordinate descent (CD) optimization algorithm. AE is applied to an optimization algorithm in a continuous domain to make the search independent from the coordinate system. This allows for performance improvements in non-separable problems and in problems where traditional CD fails. ACD utilises AE to perform its optimization process. For more detail on ACD, the reader is referred to [10].

3 Set-Based Particle Swarm Optimization Polynomial Regression

A BPSO can be used to learn polynomial structure by letting the position vector represent all possible terms in the universal set. A position entry where $x_{ij} = 1$ means that particle i has selected term j to form part of the polynomial structure. However, if the universal set contains n_t terms, the BPSO particles' positions and velocities are fixed at size $n_x = n_t$, which is expected to scale poorly [11].

The proposed solution to this dimensionality problem is to use a SBPSO, outlined in Algorithm 1, with its variable position size to represent the selected terms from the universal set. This allows only the necessary terms to be added to the position set, meaning that particle position sizes are not fixed to be of size n_t, allowing particle dimensions to be kept to a minimum. Positions are initialised from the universal set by selecting a small collection of terms, and velocities are initialised to the empty set. SBPSO velocities are interpreted as the terms which need to be added or removed in order to change the current set to a given target set which is calculated from the personal best, global best, or a randomly chosen set. Therefore, the attractions to the personal and global bests create pressure for position sets to add terms from the personal and global bests, and to remove possibly unnecessary terms. The SBPSO velocities also create pressure to explore the search space by adding terms not currently in the particle position, personal best or global best; while also removing terms from the position that are potentially unnecessary.

The size of the universal set increases exponentially as the number of input dimensions are increased, and linearly as the maximum power of the target polynomial is increased:

$$|U| = 1 + \sum_{p=1}^{n_o} \sum_{i=1}^{n_x} \binom{n_x}{i} \tag{10}$$

where n_o is the maximum order, n_x is the number of input variables and the constant of one accounts for the bias term of a_0.

Algorithm 1. Set-Based Particle Swarm Optimization

```
Generate the universal set
Create a swarm containing $n_s$ particles
Initialise particle positions as random subsets of $U$
Initialise local and global best values
while Stopping conditions not true do
  for each particle $i = 1, \ldots, n_s$ do
    Use an optimization algorithm to find the coefficient values of the selected terms,
    and evaluate the quality of this solution.
    if $f(X_i) < f(Y_i)$ then
      Update local best: $Y_i = X_i$
    end if
    if $f(Y_i) < f(\hat{Y}_i)$ then
      Update global best: $\hat{Y}_i = Y_i$
    end if
  end for
  for each particle $i = 1, \ldots, n_s$ do
    Update particle $i$'s velocity and position.
  end for
end while
```

SBPSO and BPSO algorithms both find only the optimal term structure, and not the coefficients; hence the following approaches are proposed.

A PSO can be used to find the coefficients of a polynomial by setting the position vector to refer to all the possible terms from the universal set. The value of each x_{ij} refers to the coefficient of the j^{th} term, with coefficients close to 0 indicating that the corresponding term does not contribute to the polynomial structure. However, this approach has many drawbacks. The position vectors are fixed to size $n_x = n_t$ and, as with the BPSO, will suffer from poor performance in high dimensions [11]. Additionally, the threshold of when a coefficient is "close to 0" has to be defined and tuned as an additional control parameter.

In a preliminary feasibility study, this paper proposes a hybrid algorithm which uses SBPSO to find the polynomial structure and a separate optimization algorithm to find the optimal coefficients, with preliminary investigations conducted using ACD. This hybrid algorithm will have the advantage of minimizing the dimensionality of the problem using the SBPSO and the ability to find the coefficients of the polynomial using ACD.

The basic overview is as follows: the main concepts of the standalone SBPSO algorithm are still present in the hybrid algorithm; this includes the position and velocity update equations, as well as the additional set operators used to increase exploration. The interleaved optimization process using ACD is achieved in the fitness function of the SBPSO. By using the terms currently being evaluated as the input dimensions for ACD, coefficients can be found for each of the target terms. The whole optimization process, as outlined in [10], is completed for each fitness function evaluation of a SBPSO position.

4 Empirical Process

This section outlines the processes followed to evaluate the proposed polynomial regression algorithms and to compare it to existing regression algorithms.

4.1 Benchmark Problems

The main aim of this paper is to illustrate the feasibility of the SBPSO for inducing optimal polynomial structures and its secondary aim is to illustrate the need for a second optimization algorithm to find both the term structure and coefficients.

In order to test the ability of the SBPSO to find optimal polynomial structures and to compare these results to that of a BPSO, seven benchmark problems with varying characteristics were created. These test functions, f_1 to f_7, have coefficients of 1 to allow the SBPSO and BPSO algorithms to be tested in isolation and to accurately measure their term-choosing abilities. The polynomials for these problems have a known order, allowing this information to be used to calculate the universal set. A further three test functions, f_8 to f_{10}, were created with non-unit coefficients to test and compare the regression abilities of the proposed combined SBPSO and ACD algorithm, as well as a standard PSO algorithm. Table 1 outlines the test functions generated and their universal set characteristics.

Table 1. Proposed test functions and their generated universal set characteristics

Function	Max degree	Universe size
$f_1(\boldsymbol{x}) = x_1^3 + x_2^2 + x_2 + 1$	5	16
$f_2(x) = x^3 + x^2 + x$	5	6
$f_3(x) = x^7 + x^5 + x^4 + 1$	9	10
$f_4(x) = x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$	10	11
$f_5(\boldsymbol{x}) = x_1^2 + x_2^2 + x_1^2 x_3^2 + x_3$	4	29
$f_6(\boldsymbol{x}) = x_1^6 + x_2^5 + x_3^4 + x_1 + x_2^2 x_3^2 + x_3 + 1$	8	57
$f_7(\boldsymbol{x}) = x_1 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_1 x_2 x_4 x_5 + x_3 x_4 + 1$	4	125
$f_8(x) = 0.5x^3 + 2x^2 - x$	5	6
$f_9(\boldsymbol{x}) = x_1^2 - 2x_2^2 + 3x_1^2 x_3^2 - 1.5x_3$	4	29
$f_{10}(\boldsymbol{x}) = -3x_1 - 3x_2^2 - 3x_3^2 - 3x_4^2 - 3x_5^2 + 2x_1 x_2 x_4 x_5 + 5x_3 x_4 - 6.2$	4	125

4.2 Tuning Algorithm Configurations

Each of the control parameters of the algorithms used were tuned using quasi-randomly generated Sobol sequences [14]. These sequences are generated to provide good coverage of the hypercube generated by the control parameter search space.

The control parameters of each algorithm were tuned per problem. This was done by sampling values for the control parameters as specified in Table 2. For each algorithm, on each test problem, 128 Sobol sequences were generated by sampling from the specified ranges and tuned for 500 iterations, with 30 particles in the swarm. The PSO control parameters were sampled to also satisfy the stability conditions as outlined in [1]. The obtained optimal parameters are outlined in Table 3 and Table 4, rounded to four (4) decimal places for brevity. The best parameter combination was selected as the one that had the best generalizable approximation ability, as represented by the lowest MSE over the test set.

Table 2. Table of the parameters tuned for each implemented algorithm

SBPSO		BPSO		ACD		PSO	
Parameter	range	Parameter	range	Parameter	range	Parameter	range
c_1	$[0, 1]$	ω	$[0, 1]$	k_{succ}	$\{2\}$	ω	$[0, 1]$
c_2	$[0, 1]$	c_1	$[0, 2]$	k_{unsucc}	$\{0.5\}$	c_1	$[0, 2]$
c_3	$[0.5, 5]$	c_2	$[0, 2]$	λ	$[0, 1]$	c_2	$[0, 2]$
c_4	$[0.5, 5]$	v_{max}	$[0, 6]$			λ	$[0, 1]$

Table 3. Optimal control parameters for SBPSO, BPSO on f_1 to f_7

Function	SBPSO c_1, c_2, c_3, c_4	BPSO $\omega, c_1, c_2, v_{max}$
f_1	$0.0683, 0.5605, 0.9130, 0.9306$	$0.5249, 1.0498, 0.2548, 5.4755$
f_2	$0.9648, 0.5351, 3.5410, 3.5410$	$0.2758, 1.0478, 1.4716, 3.3603$
f_3	$0.7226, 0.6523, 4.8417, 3.2246$	$0.5063, 0.7431, 0.5107, 3.4306$
f_4	$0.5166, 0.5498, 3.8793, 4.1782$	$0.3706, 0.4912, 1.6259, 0.6123$
f_5	$0.8105, 0.8183, 2.7060, 4.9736$	$0.9243, 1.0322, 0.3935, 4.2509$
f_6	$0.8916, 0.9248, 2.1918, 4.7407$	$0.9858, 0.4091, 1.7392, 4.5849$
f_7	$0.8564, 0.6787, 2.2797, 1.1372$	$0.9936, 1.2373, 0.6923, 3.7880$

Table 4. Optimal control parameters for Hybrid, PSO on f_8 to f_{10}

Function	Hybrid $c_1, c_2, c_3, c_4, \lambda$	PSO $\omega, c_1, c_2, \lambda$
f_8	$0.5830, 0.5146, 1.6118, 0.8208, 0.2998$	$0.7631, 0.9169, 1.7158, 0.0883$
f_9	$0.2041, 0.3623, 0.7856, 4.4594, 0.8818$	$0.7109, 1.4218, 1.2656, 0.9921$
f_{10}	$0.3720, 0.8818, 0.8032, 0.6450, 0.0576$	$0.6909, 0.3818, 1.8291, 0.2973$

4.3 Performance Measures

For each problem, 10000 n_x-dimensional data points were created by calculating the Cartesian product of n_x generated real-valued axes to form the complete data-set. These sets were split into training and test sets with the training set being 70% of the total and the test 30%. In order to quantify the performance of the algorithm, the MSE over the train and test set is reported on, as well as the average size of the found polynomial.

The final tests were run with the selected parameter combinations, with the results summarised in Section 5. For SBPSO and BPSO 2000 iterations over 30 independent runs were used; for the hybrid and PSO algorithms, 500 iterations over 30 independent runs were used due to the high computational complexity of the hybrid algorithm. For the final tests, all algorithms had 30 particles in their swarms.

5 Results

This section outlines the results obtained from applying the SBPSO to the problem of finding the optimal term structure, followed by the preliminary investigation into the feasibility of a hybrid algorithm to find both the optimal term structure and coefficients.

5.1 SBPSO and BPSO Results

Table 5 shows the performance of SBPSO and BPSO on test functions f_1 to f_7 by reporting on the train and test MSEs. Table 6 shows how many independent runs induced the correct target polynomial structure; this was calculated by comparing, term by term, the structure of the found polynomial with the known structure of the target polynomial. For each problem, the found polynomial that was most similar to the target polynomial is reported on, as well as how many of the independent runs induced the correct polynomial. In all test problems, except f_4, all 30 independent runs induced the correct polynomial term structure.

Table 9 compares the average size of the found optimum over the 30 independent runs with the target polynomial size for both the SBPSO and the BPSO. Specifically, the number of terms in the SBPSO global best positions, and the number of one's in the BPSO global best positions were averaged over the 30 independent runs to calculate the induced polynomial size.

Table 5. MSE values achieved by SBPSO and BPSO on problems f_1 to f_7

Problem	SBPSO		BPSO	
	Train MSE	Test MSE	Train MSE	Test MSE
f_1	0.99 ± 0.01	0.99 ± 0.02	0.99 ± 0.01	0.99 ± 0.03
f_2	1.00 ± 0.01	0.99 ± 0.02	1.00 ± 0.01	1.0 ± 0.03
f_3	0.99 ± 0.01	1.0 ± 0.02	0.99 ± 0.01	0.99 ± 0.02
f_4	1.02 ± 0.19	1.03 ± 0.17	0.99 ± 0.01	1.00 ± 0.02
f_5	1.00 ± 0.01	1.00 ± 0.02	1.00 ± 0.01	1.00 ± 0.02
f_6	1.00 ± 0.01	0.99 ± 0.03	0.99 ± 0.01	1.00 ± 0.02
f_7	0.99 ± 0.01	1.00 ± 0.02	9353.03 ± 41904.74	9263.03 ± 41475.77

Table 6. Polynomials induced by SBPSO for problems f_1 to f_7

Function	Best induced polynomial	# correct
f_1	$x_1^3 + x_2^2 + x_2 + 1$	30
f_2	$x^3 + x^2 + x$	30
f_3	$x^7 + x^5 + x^4 + 1$	30
f_4	$x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$	29
f_5	$x_1^2 + x_2^2 + x_1^2x_3^2 + x_3$	30
f_6	$x_1^6 + x_2^5 + x_3^4 + x_1 + x_2^2x_3^2 + x_3 + 1$	30
f_7	$x_1 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_1x_2x_4x_5 + x_3x_4 + 1$	30

The results show that SBPSO performed very well when inducing the optimal term. The SBPSO performs better than BPSO in higher dimensions, as seen for f_7 where BPSO failed and SBPSO succeeded. The results f_4 indicate that SBPSO tends to keep position sizes smaller, as it was unable to induce the correct structure in one run.

5.2 Hybrid and PSO Results

Table 7 shows the performance of the hybrid SBPSO and PSO on test functions f_8 to f_{10} by reporting on the train and test MSEs. Table 8 shows how many independent runs induced the correct target polynomial structure. For each problem, the found polynomial that was most similar to the target polynomial is reported on, as well as how many of the independent runs induced the correct polynomial. The hybrid algorithm often did not induce the exact polynomial term structure of the target polynomial, but was still able to minimize the MSE by compensating for incorrectly chosen terms by varying the coefficients.

Table 10 compares the average size of the found optimum over the 30 independent runs with the target polynomial size. Specifically, the number of terms in the hybrid algorithm's global best positions, and the number of dimensions in the PSO's global best positions were averaged over the 30 independent runs to calculate the induced polynomial size. In the case of the PSO, the particles' dimensionality is always the same as the size of the universal set, but it was added for consistency.

Table 7. MSE values achieved by the hybrid algorithm and PSO on problems f_8 to f_{10}

Problem	Hybrid		PSO	
	Train MSE	Test MSE	Train MSE	Test MSE
f_8	1.00 ± 0.01	1.00 ± 0.03	466018.07 ± 1045346.24	474531.68 ± 1073162.36
f_9	3.22 ± 4.82	17.06 ± 56.73	3.56e21 $\pm$ 5.75e21	3.46e21 $\pm$ 5.58e21
f_{10}	1.01 ± 0.04	1.08 ± 0.11	2.56e28 $\pm$ 7.25e28	2.47e28 $\pm$ 7.06e28

Table 8. Polynomials induced by the hybrid algorithm for problems f_8 to f_{10}

Function	Best induced polynomial	# correct
f_8	$0.50x^3 + 1.99x^2 - 0.99x + 0.02$	27
f_9	$x_1^2 - 2x_2^2 + 3x_1^2x_3^2 - 1.5x_3 - 0.04$	0
f_{10}	$-2.99x_1 - 3.00x_2^2 - 2.99x_3^2 - 3.0x_4^2 + 1.99x_5^2$ $+1.99x_1x_2x_4x_5 + 0.0x_1x_2x_3 + 4.99x_3x_4 + 0.00x_1x_4 - 4.78$	0

The hybrid algorithm performed considerably better than the PSO when searching for optimal coefficients. For each of the problems f_8 to f_{10}, the hybrid algorithm was able to minimize the MSE to an acceptable range. However, the hybrid algorithm did not find the optimal term structure, as ACD was able to minimize the contribution of these incorrectly chosen terms with coefficients close to zero. The PSO algorithm was unable to approximate the correct coefficient values, as seen by the poor MSE results.

Table 9. SBPSO and BPSO induced polynomial sizes

Problem	Target size	Average size	
		SBPSO	BPSO
f_1	4	4.0	4.0
f_2	3	3.0	3.0
f_3	4	4.0	4.0
f_4	9	8.96	9.0
f_5	4	4.0	4.0
f_6	7	7.0	7.0
f_7	8	8.0	8.5

Table 10. Hybrid and PSO induced polynomial sizes

Problem	Target size	Average size	
		Hybrid	PSO
f_8	4	4.1	6.0
f_9	4	8.03	29.0
f_{10}	8	11.57	125.0

6 Conclusions and Future Work

The purpose of this paper was to propose a novel set-based approach to inducing optimal polynomial structures using a well-established optimization algorithm, and to lay the foundation for future improvements and additions to be made to this approach.

The proposal of using a set-based optimization algorithm holds promise for future work, as the SBPSO performed well when tasked with selecting optimal term combinations and showed that it scales better than the existing BPSO method. Because the SBPSO is grounded in set-theory, it holds even more promise to be improved to perform better on the high dimensional problems in which it failed.

The proposed hybrid algorithm consists of the SBPSO algorithm to find an optimal combination of monomials in the polynomial, and the ACD algorithm to find optimal coefficients of these monomials. The application of the SBPSO and ACD shows promise as a well-suited set-based solution for polynomial regression problems. Preliminary results show good performance on low dimensional and low order polynomials where the universal set size remains relatively small, but with some performance drawbacks with larger search spaces. The proposed algorithm also provides the advantage over existing algorithms that it has easily interpretable results and the most potential for improvement. However, this paper's results are not sufficient to draw conclusions about the SBPSO or hybrid SBPSO algorithm's performance on high dimensional problems. More complex polynomials need to be tested to understand how the algorithms will behave in high dimensional spaces.

Future work includes further testing the capabilities of the SBPSO algorithm when applied to real-world data-sets and to investigate the possibility of using computationally cheaper methods to find optimal the coefficients. The algorithm can also be extended to work in dynamic environments by introducing quantum-PSO inspired effects.

Acknowledgements. The authors acknowledge the Centre for High Performance Computing (CHPC), South Africa, for providing computational resources for this research paper.

References

1. van den Bergh, F., Engelbrecht, A.P.: A study of particle swarm optimization particle trajectories. Inf. Sci. **176**(8), 937–971 (2006)
2. Engelbrecht, A.P., Grobler, J., Langeveld, J.: Set based particle swarm optimization for the feature selection problem. Eng. Appl. Artif. Intell. **85**, 324–336 (2019)
3. Erwin, K., Engelbrecht, A.P.: Set-based particle swarm optimization for portfolio optimization. In: Proceedings of the Twelfth International Conference on Swarm Intelligence, pp. 333–339 (2020)
4. Hansen, N.: Adaptive encoding: how to render search coordinate system invariant. In: Proceedings of the International Conference on Parallel Problem Solving from Nature, pp. 205–214 (2008)
5. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989)
6. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
7. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, vol. 5, pp. 4104–4108 (1997)
8. Krogh, A., Hertz, J.A.: A simple weight decay can improve generalization. In: Advances in Neural Information Processing Systems, vol. 4, pp. 950–957. Morgan-Kaufmann (1992)
9. Langeveld, J., Engelbrecht, A.P.: Set-based particle swarm optimization applied to the multidimensional knapsack problem. Swarm Intell. **6**, 297–342 (2012)
10. Loshchilov, I., Schoenauer, M., Sebag, M.: Adaptive coordinate descent. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 885–892 (2011)
11. Oldewage, E.T.: The perils of particle swarm optimization in high dimensional problem spaces. In: MSc thesis, University of Pretoria (2019)
12. Pampara, G., Franken, N., Engelbrecht, A.P.: Combining particle swarm optimisation with angle modulation to solve binary problems. In: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 1, pp. 89–96 (2005)
13. Shi, Y., Eberhart, R.: A modified particle swarm optimizer. In: Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 69–73 (1998)
14. Sobol, I.M.: On the distribution of points in a cube and the approximate evaluation of integrals. USSR Comput. Math. Math. Phys. **7**(4), 86–112 (1967)
15. Specht, D.F.: A general regression neural network. IEEE Trans. Neural Netw. **2**(6), 568–576 (1991)