Polynomial Regression Using Set-Based Particle Swarm Optimization

J. van Zyl
Computer Science Division
Stellenbosch University
Stellenbosch, South Africa
Email: 20706413@sun.ac.za

A.P. Engelbrecht
Department of Industrial Engineering,
and Computer Science Division
Stellenbosch University
Stellenbosch, South Africa
Email: engel@sun.ac.za

***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.**

The polynomial regression is defined as a multi-objective optimization problem, with the goals to find the optimal number of terms in the polynomial and the optimal values of the coefficients of the terms, such that the approximation error is minimized. A set-based PSO is developed to find the optimal number of terms, and the adaptive coordinate descent algorithm is used to find optimal coefficient values. It is shown that the set-based PSO approach to polynomial regression has potential, performing well on lower dimensional regression problems.

***Index Terms*—particle swarm optimization, polynomial regression, adaptive coordinate descent, set-based particle swarm optimization**

I. 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 algorithm to find polynomial mappings.

The polynomial regression problem is defined as a multi-objective optimization problem, using a set-based solution representation. The objectives are to find:

1) the smallest number number of terms and polynomial order, and
2) the optimal coefficient values for these terms such that the approximation error is minimized.

It is proposed that a set-based particle swarm optimization (SBPSO) algorithm is used to achieve the first objective, by finding an optimal term set. For each optimal term set, the adaptive coordinate descent (ACD) algorithm is used to find optimal coefficient values for the terms in the term set. The proposed algorithm therefor performs an interleaved, dual optimization process, namely with respect to term space and with respect to the coefficient space.

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, and compared to a genetic algorithm approach to polynomial regression, a standard non-set based particle swarm optimization (PSO) and a neural network. SBPSO is shown to be able to find the optimal set of terms by itself, ACD is able to find the optimal coefficients by itself. 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 algorithm is able to approximate the source polynomial from the input data both in structure and in coefficients.

Section II discusses the concepts needed to implement the work in this paper, section III outlines how existing optimization algorithms can be combined to solve for polynomial mappings, section IV contains the empirical procedures followed, section V discusses the results followed by the conclusion in section VI.

II. Background

The purpose of this section is to outline the necessary concepts used in this paper. This includes background information on polynomial regression, PSOs, SBPSOs, swarm diversity, ACD, genetic algorithms and neural networks.

A. Polynomial Regression

Polynomials are made of consistuent 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, $D = \{(\boldsymbol{x}_p, y_p)|p = 1, \dots, n_p\}; \boldsymbol{x}_p = (x_{1p}, x_{2p}, \dots, x_{n_x p})$ 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.

Uni-variate polynomials have $n_x = 1$, and are of 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
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 structure (or complexity) of the polynomial. The structure of the polynomial is minimized to prevent overfitting, while underfitting is prevented by minimizing the approximation error. For the purposes of this paper, approximation error is estimated using the mean squared error (MSE), defined as

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

In essence polynomial approximation is a multi-objective optimization problem,

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

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

$$P(f(\boldsymbol{x})) = \frac{1}{2} \sum_{i=0}^{n_t} a_i^2 \tag{6}$$

referred to as ridge regression, or weight decay in neural network terminology [1].

B. Particle Swarm Optimization

Particle swarm optimization is a well-established swarm-based optimization method [2]. Since the original proposed algorithm, many modifications have been proposed to improve its performance and its application on different problem types. [3] [4] [5]

1) Basic Particle Swarm Optimization: The first PSO proposed by Eberhart and Kennedy [2] 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.

Let n_s denote the swarm size, 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 $\hat{\boldsymbol{y}}(t)$, with each being d-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 each iteration using

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

where $v_{ij}(t)$ is the velocity, or step size and direction, calculated for each dimension j using [6]

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

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 $i \in \{1, \ldots, n\}$ and $d \in \{1, \ldots, d\}$.

The control parameters ω, c_1 and c_2 control the exploration-exploitation trade-off in PSOs. This trade-off is managed to determine whether the goal of the swarm is to discover new potential solutions or to refine already found optima. The effect of these control parameters has been studied in [5].

2) Set-based Particle Swarm Optimization: The set-based PSO, as implemented in this article, was developed to solve the multi-dimensional knapsack problem [7]. This is a discretised version of the standard PSO which makes use of a set-based search space instead of a d-dimensional continuous search space. Particle positions consist of elements from the universal set, U, while the velocity is a set of operation pairs which add or remove elements to 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 [8] [9] 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, as follows:

Denoting $\mathcal{P}(U)$ as the powerset of the universal set meaning the set of all possible subsets, and $A \times B$ as the Cartesian product of two sets A and B.

The addition of two velocities, $V_1 \oplus V_2$, is a mapping, $\oplus : \mathcal{P}(\{+,-\} \times U)^2 \to \mathcal{P}(\{+,-\} \times U)$ that takes two velocities and yields one velocity. Implemented as set operations, velocity adding velocity is interpreted as the union operator:

$$V_1 \oplus V_2 = V_1 \cup V_2 \tag{9}$$

The difference between two positions, $X_1 \ominus X_2$, is a mapping, $\ominus : \mathcal{P}(U)^2 \to \mathcal{P}(\{+,-\} \times U)$ that takes two positions and yields a velocity. The result is effectively the set operation steps which are required to convert X_2 into X_1:

$$X_1 \ominus X_2 = (\{+\} \times (X_1 \backslash X_2)) \cup (\{-\} \times (X_2 \backslash X_1)) \tag{10}$$

The scalar multiplication of a velocity, $\eta \otimes V$, is a mapping, $\otimes : [0,1] \times \mathcal{P}(\{+,-\} \times U) \to \mathcal{P}(\{+,-\} \times U)$ which takes a scalar and a velocity, yielding a velocity. The mapping results in a randomly selected subset of size $\lfloor \eta \times |V| \rfloor$ from V. Note that $0 \otimes V = \emptyset$ and $1 \otimes V = V$

The addition of a velocity and a position, $X \boxplus V$, is a mapping, $\boxplus : \mathcal{P}(U) \times \mathcal{P}(\{+,-\} \times U) \to \mathcal{P}(U)$ that takes a position and velocity and yielding the resultant position.

$$X \boxplus V = V(X) \tag{11}$$

which involves applying the operation associated with each v_i from $V = \{v_1, \ldots, v_n\}$ to X by adding or removing each e_i as dictated by the terms in the velocity.

The removal of elements, $\beta \odot^- S$ from a position $X(t)$ where S is shorthand for $X(t) \cap Y(t) \cap \hat{Y}(t)$, is the mapping, $\odot^- : [0, |S|] \times \mathcal{P}(U) \to \mathcal{P}(\{+,-\} \times U)$ that takes a scalar and a set of elements and yields a velocity. The operator is implemented by randomly selecting a subset of elements from S, with a size determined by β, to be removed from $X(t)$:

$$\beta \odot^- S = \{-\} \times \left(\frac{N_{\beta,S}}{|S|} \otimes S \right) \tag{12}$$

The number of elements selected, $N_{\beta,S}$ is defined as:

$$N_{\beta,S} = \min \left\{ |S|, \lfloor \beta \rfloor + \mathbf{1}_{\{r < \beta - \lfloor \beta \rfloor\}} \right\} \tag{13}$$

for a random number $r \sim U(0,1)$; $\mathbf{1}_{\{\text{bool}\}}$ is 1 if bool is true and 0 if bool is false.

The addition of elements, $\beta \odot^+ A$ to a position $X(t)$ where A is shorthand for $U \backslash (X(t) \cup Y(t) \cup \hat{Y}(t))$, is a mapping $\odot^+ : [0, |A|] \times \mathcal{P}(U) \to \mathcal{P}(\{+,-\} \times U)$ that takes a scalar and a set of elements and yields a velocity. The operator is implemented by randomly selecting a subset of elements from A, with a size determined by β, to be added to $X(t)$:

$$\beta \odot^+ A = \{+\} \times k\text{-Tournament Selection}(A, N_{\beta,A}) \tag{14}$$

where $N_{\beta,A}$ is the number of elements to be added to $X(t)$ as defined in equation (13) and k is a user defined parameter.

The velocity update equation is defined as:

$$\begin{aligned} V_i(t+1) = & c_1 r_1 \otimes (Y_i(t) \ominus X_i(t)) \oplus c_2 r_2 \otimes \left(\hat{Y}_i(t) \ominus X_i(t) \right) \\ & \oplus \left(c_3 r_3 \odot^+ A_i(t) \right) \oplus \left(c_4 r_4 \odot^- S_i(t) \right) \end{aligned} \tag{15}$$

where $A_i(t)$ and $S_i(t)$ is calculated independently for each particle; $c_1, c_2 \in [0,1]$ and $c_3, c_4 \in [0, |U|]$; each r_i is independently drawn from the distribution $U(0,1)$.

The position update equation is then defined by

$$X_i(t+1) = X_i(t) \boxplus V_i(t+1) \tag{16}$$

3) Diversity Measures: The exploration ability of a swarm is often indicated by the swarm diversity. If a swarm has converged, it will be unable to search new areas of the search space to discover better optima, leading to stagnation.

A high swarm diversity indicates that the swarm is geared towards exploring the search space, where as a low diversity indicates a more exploitative swarm. The rate at which the diversity decreases gives an indication of how quickly the search moves from exploration to exploitation.

To quantify swarm diversity for set-based representation, the swarm diversity can be measured by averaging the similarity between all pairs of particles (i.e. set-based positions) in the swarm, i.e. [8].

$$\mathcal{D} = \frac{1}{n} \sum_{qp} J(\boldsymbol{x}_p, \boldsymbol{x}_q) \tag{17}$$

where $J(\boldsymbol{x}_p, \boldsymbol{x}_q)$ is the Jaccard distance between two sets [10]:

$$J(\boldsymbol{x}_p, \boldsymbol{x}_q) = 1 - \frac{|\boldsymbol{x}_p \cap \boldsymbol{x}_q|}{|\boldsymbol{x}_p \cup \boldsymbol{x}_q|} \tag{18}$$

and n is the total number of particle pairs.

C. Adaptive Coordinate Descent

Adaptive coordinate descent was developed by Loshchilov *et al* [11] in 2011 as an improvement to the covariance matrix adaptation evolutionary strategy (CMA-ES). ACD adds Adaptive Encoding (AE), developed by Hansen [12], 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 for problems where traditional CD fails. The basic steps of AE are outlined in Algorithm 1.

Utilising AE, ACD is performed by the steps outlined in Algorithm 2. The parameters a and b dictate the initial search range in each dimension; k_{succ} and k_{unsucc} are user-defined parameters which control the behaviour of the search, the recommended values for a robust search are 2 and 0.5 respectively; $\boldsymbol{a}_x$ and $\boldsymbol{a}_f$ are archive stores for the positions and their function evaluations in each dimension.

D. Genetic Algorithm

Genetic algorithms (GA) are a form of nature-inspired optimization algorithms which model aspects of biological evolution to solve given problems [13] [14].

Algorithm 1 Adaptive Encoding

```
Input: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\mu$
if Uninitialized then
  Initialize the parameters as follows:
  $w_i = \frac{1}{\mu}$; $c_p = \frac{1}{\sqrt{d}}$; $c_1 = \frac{0.5}{d}$; $c_\mu = \frac{0.5}{d}$.
  $\boldsymbol{p} = \boldsymbol{0}$; $\boldsymbol{C} = \boldsymbol{I}$; $\boldsymbol{B} = \boldsymbol{I}$;
  $\boldsymbol{m} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_i$.
end if
Calculate new encoding
$\boldsymbol{m}^- = \boldsymbol{m}$
$\boldsymbol{m} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_i$
$\boldsymbol{z_0} = \frac{\sqrt{d}}{||\boldsymbol{B}^{-1}(\boldsymbol{m}-\boldsymbol{m}^-)||}(\boldsymbol{m} - \boldsymbol{m}^-)$
$\boldsymbol{z_i} = \frac{\sqrt{d}}{||\boldsymbol{B}^{-1}(\boldsymbol{x}_i-\boldsymbol{m}^-)||}(\boldsymbol{x}_i - \boldsymbol{m}^-)$; $i = 1, \ldots, \mu$
$\boldsymbol{p} = (1 - c_p)\boldsymbol{p} + \sqrt{c_p(2 - c_p)}\boldsymbol{z}_0$
$\boldsymbol{C}_\mu = \sum_{i=1}^{\mu} w_i \boldsymbol{z}_i \boldsymbol{z}_i^T$
$\boldsymbol{C}_\mu = (1 - c_1 - c_\mu)\boldsymbol{C} + c_1 \boldsymbol{p}\boldsymbol{p}^T + c_\mu \boldsymbol{C}_\mu$
$\boldsymbol{B}^o, \boldsymbol{EV}$ = eigendecomposition($\boldsymbol{C}$)
$\boldsymbol{B} = \boldsymbol{B}^o \sqrt{\boldsymbol{EV}}$
return $\boldsymbol{B}$
```

GAs make use of concepts such as crossover, mutation and elitism to evolve the next generation of individuals of the population. These stochastic processes contribute to the exploration-exploitation trade-off which allow the algorithm to find its optimal solution.

Potgieter and Engelbrecht [15] have successfully used GAs to evolve polynomials.The proposed algorithm is referred to as genetic algorithm for structurally optimal polynomial evolution (GASOPE). This algorithm is used in the empirical comparisons in Section V

E. Neural Network

Neural networks (NN) have successfully been used to learn the mappings between input and target data [16]. NNs consist of an input layer, an output layer and a number of hidden layers (or none). The performance of a NN and its ability to learn complex mappings is greatly impacted by the topographical structure of its nodes. Non-linear relationships cannot be described by a NN with no hidden layers. NNs with a single hidden layer are universal approximators and can learn any non-linear mapping, provided the layer has enough hidden units [17]. A NN is used in the empirical comparison in Section V.

III. Set-based Particle Swarm Optimization Polynomial Regression

When using a well-established technique like neural networks to solve regression problems, the output is an

Algorithm 2 Adaptive Coordinate Descent

```
$\boldsymbol{m}[i] = a + U(a, b)$, $\forall i = 1, \ldots, d$
$\boldsymbol{\sigma}[i] = (b - a)/4$, $\forall i = 1, \ldots, d$
$f_{\text{best}} = f(\boldsymbol{m})$
while Stopping conditions not true do
    for i = 1 to d do
        $\boldsymbol{x}^{'} = \boldsymbol{0}$
        $\boldsymbol{x}^{'}[i] = -\boldsymbol{\sigma}[i]$; $\boldsymbol{x}_1 = \boldsymbol{B}\boldsymbol{x}^{'}$; $f_1 = f(\boldsymbol{x}_1)$
        $\boldsymbol{x}^{'}[i] = \boldsymbol{\sigma}[i]$; $\boldsymbol{x}_2 = \boldsymbol{B}\boldsymbol{x}^{'}$; $f_2 = f(\boldsymbol{x}_2)$
        succ = False
        if $f_1 < f_{\text{best}}$ then
            $f_{\text{best}} = f_1$; $\boldsymbol{m} = \boldsymbol{x}_1$; succ = True
        end if
        if $f_2 < f_{\text{best}}$ then
            $f_{\text{best}} = f_2$; $\boldsymbol{m} = \boldsymbol{x}_2$; succ = True
        end if
        if succ = True then
            $\boldsymbol{\sigma}[i] = k_{succ}\boldsymbol{\sigma}[i]$
        else
            $\boldsymbol{\sigma}[i] = k_{unsucc}\boldsymbol{\sigma}[i]$
        end if
        $\boldsymbol{a}_x[2i-1] = \boldsymbol{x}_1$; $\boldsymbol{a}_f[2i-1] = f_1$
        $\boldsymbol{a}_x[2i] = \boldsymbol{x}_2$; $\boldsymbol{a}_f[2i] = f_2$
    end for
    Sort $\boldsymbol{a}_x$ and $\boldsymbol{a}_f$ according to $\boldsymbol{a}_f$
    Adapt encoding with $(\boldsymbol{a}_x[1], \ldots, \boldsymbol{a}_x[d])$
end while
```

incomprehensible set of weight vectors which map input data to their targets. The idea of combining the SBPSO and ACD algorithms is to be able to learn the polynomial that generates this mapping, instead of only providing the mapping. This is achieved by minimizing the error with respect to both the term space and the coefficient space. The term space is all the terms in the polynomial and the coefficient space is the real valued coefficients for each of the terms. The dimensionality of the ACD problem is the carnality of the position set being evaluated, which means that the variable number of monomials in the SBPSO position allow the dimensionality of the ACD optimization problem to be reduced.

A standard PSO can be used to attempt to learn a polynomial by solving for the coefficient space of the universal set of term combinations. This can be achieved by interpreting the position vector of the PSO as the coefficients for each term combination in the universal set. However, because the universal set consists of all combinations of the input dimensions of the original data it quickly becomes a high dimensional problem. PSOs often suffer from the curse of dimensionality and have shown poor performance when applied to problems in higher dimensions [18].

To improve performance in these high dimensions, a more scalable solution is needed. The proposed solution is to use a SBPSO to select which terms from the universal set form part of the target polynomial, and then to use ACD to find the values of the coefficients of each of these selected terms. The possible positions of a given SBPSO particle are the subsets of the power set of the universal set.

A SBPSO is suited to this application as it allows for a varied number of dimensions among positions. While a basic PSO will have fixed n_x dimensional positions, a SBPSO implementation will allow the complexity of higher dimensional problems to be reduced by allowing positions to be subsets of the universal set. SBPSOs provide an additional advantage over NNs as the results are more interpretable and the curve used to describe the mapping is found, instead of just tuned weights. NNs also become convoluted when fully connected layers are used, resulting in an rapid increase in the number of weights needed to tune as extra neurons are added. GAs have been applied to polynomials quite successfully, but have the drawback of needing both the maximum order of the polynomial and the maximum number of terms in the polynomial specified.

The effect of increasing the number of dimensions or the maximum degree of a polynomial on the universal set size is seen in Figure 1.

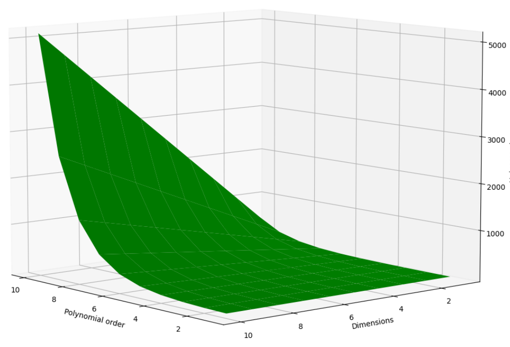

Fig. 1. Effect of polynomial order and dimensionality on universe size

Algorithm 3 Set-Based Particle Swarm Optimization

```
Generate the universal set
Create a swarm containing $n_s$ particles
Initialize particle positions as random subsets of $U$
Initialize local and global best values
while Stopping conditions not true do
    for each particle $i = 1, \dots, n_s$ do
        Use ACD to find the coefficients of the monomial, as outlined in algorithm 2.
        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 $i = 1, \dots, n_s$ do
        Update particle $i$'s velocity using equation (15)
        Update particle $i$'s position using equation (16)
    end for
end while
```

When using ACD to solve for the optimal solution in the term space, it is important to consider how the ACD algorithm influences the exploration-exploitation trade-off in SBPSO. Two changes to the ACD algorithm are proposed to manage the exploration-exploitation trade-off.

The authors recommend using a dynamically allocated number of ACD iterations, dependent on the current progress of the SBPSO iterations. Initially limiting the number of ACD iterations allows sub-optimal and unrefined solutions to be accepted by the SBPSO. This increases the exploration in areas of the search space that might appear less desirable but hold better solutions.

An additional stopping condition was also added to the ACD algorithm, one which terminates the search if there is no improvement in the found optimum for a given number of iterations.

Let τ be the required number of iterations without a solution improvement before termination. Making the initial value of τ small allows sub-optimal solutions to be explored by the SBPSO, thereby increasing the exploration of the search space in early iterations. The value of τ was linearly increased during the search process to encourage more exploitation as iterations progressed.

IV. Empirical Process

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

A. Benchmark problems

In order to test the regression abilities of the algorithm, ten benchmark problems were created with varying characteristics. The polynomials in these problems had a known order, and the user-defined maximum was set to two more than the order of the given polynomial. The combinatorics for calculating the size of the universal set is outlined in equation (19)

$$|U| = 1 + \sum_{p=1}^{P} \sum_{i=1}^{n_x} \binom{n_x}{i} \tag{19}$$

where P is the maximum order, n_x is the number of input variable and the constant of one accounts for the bias term of a_0

The following polynomials were used to create synthetic benchmark problems:

1) Function 1: Function f_1 is a simple polynomial in two dimensions:

$$f_1(\boldsymbol{x}) = x_1^3 + x_2^2 + x_2 + 1 \tag{20}$$

Approximation of this polynomial using a maximum degree of 4 results in a universal set size of $|U| = 13$.

2) Function 2: Function f_2 has the same structure as f_1, but with different coefficient values:

$$f_2(\boldsymbol{x}) = 1.5x_1^3 - 3x_2^2 + 0.4x_2 + 5.5 \tag{21}$$

Approximation of this polynomial using a maximum degree of 5 results in a universal set size of $|U| = 16$.

3) Function 3: Function f_3 has the same values as f_2, but has coefficients outside of the range of $[-10, 10]$ provided to ACD:

$$f_3(\boldsymbol{x}) = 15x_1^3 - 20x_2^2 + 11.5x_2 + 5.5 \tag{22}$$

Since the initial search points for ACD is randomly selected within $[a, b]^{n_t}$, testing coefficients outside of this Initialization range tests ACD ability to explore outside of this range. Approximation of this polynomial using a maximum degree of 5 results in a universal set size of $|U| = 16$.

4) Function 4: Function f_4 is a polynomial with one dependent variable and a low maximum degree:

$$f_4(x) = 0.5x^3 + 2x^2 - x \tag{23}$$

Approximation of this polynomial using a maximum degree of 5 results in a universal set size of $|U| = 6$.

5) Function 5: Function f_5 has one dependent variable and a high maximum degree:

$$f_5(x) = 3x^7 - 9x^5 - x^4 + 6 \tag{24}$$

Approximation of this polynomial using a maximum degree of 9 results in a universal set size of $|U| = 10$.

6) Function 6: Function f_6 has one dependent variable, a high maximum degree and many contributing terms:

$$f_6(x) = 3x^8 - 9x^7 + 6.5x^6 - 3x^5 - x^4 + x^3 + 0.5x^2 + x - 7 \tag{25}$$

Approximation of this polynomial using a maximum degree of 10 results in a universal set size of $|U| = 11$.

7) Function 7: Function f_7 has three dependent variables and a low polynomial degree:

$$f_7(\boldsymbol{x}) = x_1^2 - 2x_2^2 + 3x_1^2x_3^2 - 1.5x_3 \tag{26}$$

Approximation of this polynomial using a maximum degree of 4 results in a universal set size of $|U| = 29$.

8) Function 8: Function f_8 has three dependent variables and a high degree:

$$f_8(\boldsymbol{x}) = 3x_1^6 + 3x_2^5 + 3x_3^4 - 2x_1 - 7x_2^2x_3^2 - x_3 + 5.5 \tag{27}$$

Approximation of this polynomial using a maximum degree of 8 results in a universal set size of $|U| = 57$.

9) Function 9: Function f_9 is in five dimensions and has a low degree:

$$\begin{aligned} f_9(\boldsymbol{x}) = &-3x_1^2 - 3x_2^2 - 3x_3^2 - 3x_4^2 \\ &- 3x_5^2 + 2x_1x_2x_4x_5 + 5x_3x_4 - 6.2 \end{aligned} \tag{28}$$

Approximation of this polynomial using a maximum degree of 4 results in a universal set size of $|U| = 125$.

10) Function 10: Function f_{10} has ten dependent variables:

$$f_{10}(\boldsymbol{x}) = 10x_1^5 - 7x_3^4 + 3x_2^3 - 8x_6^2x_7^2x_8^2 - 2.5x_8x_9x_{10} + 3 \tag{29}$$

Approximation of this polynomial using a maximum degree of 7 results in a universal set size of $|U| = 7162$.

B. Tuning Algorithm Configurations

This subsection outlines the parameters involved in each algorithm, and the process used to find the best values for these parameters.

Each of the benchmark problems were tuned using quasi-randomly generated Sobol sequences [19]. These sequences are generated to provide good coverage of the hypercube generated by the control parameter search space. The SBPSO and ACD algorithms have a total of seven parameters to tune, in this case, 128 sequences were generated for all tuning purposes. All tuning was done for 500 iterations.

1) Set-Based Particle Swarm Optimization: Of the seven parameters, four are from the SBPSO algorithm, namely the acceleration coefficients c_1, c_2, c_3 and c_4. These coefficients control the contribution of each velocity component in the velocity update equation.

2) Adaptive Coordinate Descent: A further two parameters are positive coefficients from the ACD algorithm, namely k_{succ} and k_{unsucc}. The original authors recommend the values of $k_{succ} = 2$ and $k_{unsucc} = 0.5$ to ensure robustness in the optimization process. It is necessary that $k_{succ} > 1$ to ensure that the search can extend past the originally defined boundaries in the event that the optimum lies outside of the given values. It is also necessary that $k_{unsucc} < 1$ holds to ensure that the potential optima can be exploited by narrowing the search space. The values for the search range bounds of ACD, i.e. a and b, were fixed to -10 and 10 respectively. These values are user specified for each problem, but the algorithm has the ability to extend the search beyond the initial range as shown by the results of function f_3 as discussed in Section V.

3) Particle Swarm Optimization: PSOs generally have three control parameters which need to be tuned, namely the inertia weight, ω, and the acceleration coefficients c_1 and c_2. In order to guarantee the swarm's convergence to an equilibrium state, these parameters have to be chosen to satisfy the convergence conditions as proven by [20], namely

$$\omega > \frac{1}{2}(c_1 + c_2) - 1 \tag{30}$$

For tuning purposes, 128 Sobol sequences were generated to satisfy the condition in equation (30)

4) Genetic Algorithm for Structurally Optimal Polynomial Evolution: The parameters tuned for GASOPE are derived from the required tuning for the underlying implemented genetic algorithm (GA). These parameters are the mutation rate p_m, which controls the probability that an individual will randomly have new genetic material added to its chromosomes; the crossover rate p_c, which dictates the probability that two individuals will go through the process of genetic recombination; and the elite rate p_e which determines the level of elitism applied in each generation.

5) Neural Network: The NN requires both an optimal structure and tuned parameters for the training algorithm used to adjust the weights, for each problem. The structure of the NNs implemented in this paper had an input layer with d neurons, one hidden layer with ν neurons and an output layer with one neuron. The parameters tuned for each problem was: ν, tuned by doubling the hidden layer size, starting with 1, until over-fitting occurs and the choosing an optimal size. The second parameter is the learning rate of the network η which dictates the step sizes taken during the search process of finding the optimal weight values. Finally, the momentum allows weight changes calculated from batches in the training process to be averaged. This momentum, α can improve convergence times by "smoothing out" the search path to the optimum.

6) Objective Function: The λ coefficient is used to control the influence of the weight decay penalty term in the objective function.

This paper makes use of a time-dependent penalty coefficient, $\lambda(t)$. This coefficient increases linearly from zero as the iterations progress. This reduces the effect of the penalty term in early iterations in order to encourage more exploration by not limiting exploration of sub-optimal candidate solutions.

The parameters tuned, and their ranges are outlined in Tables I and II.

TABLE I
TABLE OF THE PARAMETERS TUNED FOR THE PROPOSED ALGORITHM

SBPSO		ACD	
Parameter	**range**	**Parameter**	**range**
c_1	$[0,1]$	k_{succ}	$[0,1]$
c_2	$[0,1]$	k_{unsucc}	$[1,2]$
c_3	$[0.5,5]$	λ	$[0,1]$
c_4	$[0.5,5]$		

TABLE II
TABLE OF THE PARAMETERS TUNED FOR THE COMPARISON ALGORITHMS

PSO		GASOPE		NN	
Param	**range**	**Param**	**range**	**Param**	**range**
ω	$[0,1]$	p_c	$[0,1]$	ν	$\in \mathbb{N}$
c_1	$[0,2]$	p_m	$[0,1]$	η	$[0,1]$
c_2	$[0,2]$	p_e	$[0,1]$	α	$[0,1]$
λ	$[0,1]$				

C. Performance Measures

For each problem, 10000 d-dimensional data points were generated to form the complete data-set. These sets were split into training and test sets with the training set being 40% of the total and the test 60%. In order to quantify the performance of the algorithm, the MSE over the test set is reported on.

D. Statistical Process

Each benchmark problem was tuned by running 128 different parameter combinations for 500 iterations. The best parameter combination was selected as the one that had the best approximation ability, as represented by the lowest MSE over the test set.

The final tests were run with the selected parameter combinations, for 2000 iterations over 30 independent runs, with the results summarised in Section V. SBPSO, PSO and GASOPE are population-based optimizers, while NN is not. Swarm sizes were fixed at 30. Therefore, for each of the independent runs of the NN, the NN was executed 30 times and the best run chosen in order to provide a fairer comparison between the algorithms.

Before the performance of the full SBPSO algorithm was evaluated, the two search components were separately evaluated as follows:

- The SBPSO was evaluated on polynomials where the coefficients of the monomials were all set to 1. This was done to evaluate the SBPSO's ability to select the optimal set of monomials.
- the ACD was run on a fixed set of monomials to determine optimal coefficients. This was done to test that the ACD algorithm works well on the coefficient space.

The ability of the SBPSO to solve for the optimal combination of monomials in the term space was evaluated on function f_1. The universe size of the search space was 13

Applying SBPSO to the polynomial f_1 defined in equation (20) without incorporating ACD to find optimal coefficients, allows one to test the ability of SBPSO to solve the term space. The universe size was 13, creating the space of possible term combinations to be 2^{13}. Figure 2 shows the error profile, and illustrates that the SBPSO managed to find the optimum terms with an MSE of 2.9.

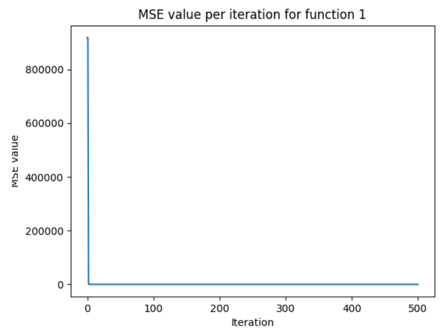

Fig. 2. Performance of SBPSO in isolation

The ACD algorithm's ability to find the optimal coefficients was evaluated by applying ACD to the pre-set polynomial structure of f_2 in equation (21). Figure 3 shows very fast convergence to an MSE of 3.051, producing the polynomial $1.5x_1 - 2.998x_2^2 + 0.402x_2 + 1.503$.

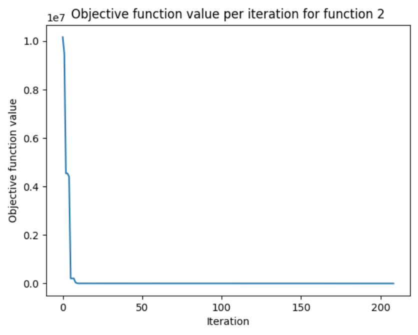

Fig. 3. Performance of ACD given the term combination

These individual tests provide evidence that the two components work well individually.

V. Results

The MSE results obtained on each test problem by the different optimization algorithms is presented in Table III, the rankings of the tested algorithms is presented in Table IV and the polynomials found by the SBPSO is shown in Table V.

A. *Benchmark problem results*

Table V shows the induced polynomials for each problem by the best of the 30 SBPSO runs.

B. *SBPSO performance analysis*

The results show that SBPSO performs very well on problems which generate a smaller universal set. The size of search space generated by the possible position sets is the carnality of the powerset of the universal set, which increases exponentially as the universal set size increases.

Examining problem f_5, it is apparent that the performance is significantly worse than expected after comparison to other problems with a similar universe size. The results from problem f_5 exposes a lack of exploration ability of the SBPSO, causing the search space to be insufficiently explored and the algorithm settling on a sub-optimal local minimum. This is seen when comparing f_5, which is a uni-variate problem, with f_4, which is also a uni-variate problem The difference is that f_5 has a maximum degree of seven while f_4 has a maximum degree of three. This means that the resulting universe size was ten for f_5 and six for f_4.

Figure 4 plots the size of the global best position against the target polynomial size averaged over all 30 independent runs. The plot clearly shows variation in the first 500 iterations but a stagnation in position size thereafter. This trend is mirrored in Figure 5, showing the average local best position sizes for each iteration. The fact that the average lbest position size was similar to the gbest size indicates that the local bests take on the same position as the gbest extremely quickly after a new gbest has been discovered. This is creates a tendency to exploit the found solutions instead of exploring and could be as a result of poorly chosen control parameters which favour fast convergence.

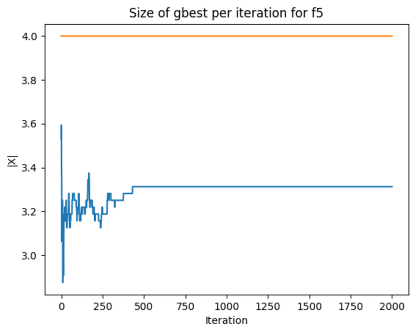

Fig. 4. The global best position size compared to the target size

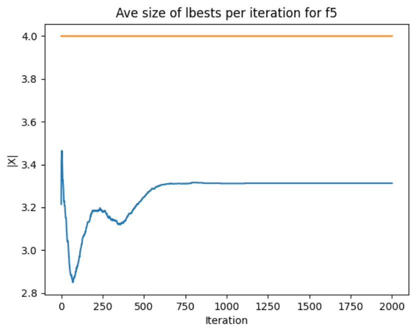

Fig. 5. The average local best position size compared to the target size

The gbest and lbest positions remain undersized during the optimization process, indicating a lack of suitable terms being added to these positions in order to improve the MSE as there is not exploration of the entire space generated by U.

The sum of the square coefficients of the gbest particle also provide insight into the behaviour of the optimizier. Because it is known that the position sizes contain few monomials, large weight decay values are an indication

TABLE III
MSE VALUES ACHIEVED BY EACH ALGORITHM ON EACH PROBLEM

Problem	**MSE**			
	SBPSO	**PSO**	**GASOPE**	**NN**
f_1	1.45 ± 4.29	$2.6e10 \pm 1.3e10$	0.01 ± 0.02	1499 ± 943
f_2	0.56 ± 2.79	$4.7e12 \pm 4.1e12$	0.01 ± 0.02	$6.1e5 \pm 8.0e5$
f_3	185.4 ± 553.8	$2.3e12 \pm 2.0e12$	0.1 ± 0.03	$2.7e7 \pm 2.8e7$
f_4	0.01 ± 0.03	259 ± 172	0.01 ± 0.01	339 ± 319
f_5	$1.5e5 \pm 21e5$	$2.8e8 \pm 2.3e8$	0.2 ± 0.1	$7.5e9 \pm 4.5e9$
f_6	$5.8e9 \pm 4.1e9$	$5.8e9 \pm 4.1e9$	13.58 ± 9.5	$1.8e17 \pm 6.0e17$
f_7	1.61 ± 6.77	$3.7e28 \pm 1.9e28$	0.1 ± 0.3	$1.9e11 \pm 1.4e11$
f_8	$1.7e20 \pm 4.6e20$	$5.0e62 \pm 4.2e62$	$9.1e14 \pm 5.1e15$	
f_9	$9.3e6 \pm 1.8e7$	$8.5e31 \pm 1.1e31$	0.9 ± 1.1	$2.0e8 \pm 1.3e8$
f_{10}	$1.4e13 \pm 8.3e12$	$6.5e123 \pm 3.2e123$	8.6 ± 4.0	$1.7e11 \pm 1.5e11$

TABLE IV
RANKING ACHIEVED BY EACH ALGORITHM ON EACH PROBLEM

Problem	**Ranking**			
	SBPSO	**PSO**	**GASOPE**	**NN**
f_1	2	4	1	3
f_2	2	4	1	3
f_3	2	4	1	3
f_4	2	4	1	3
f_5	2	3	1	4
f_6	2	3	1	4
f_7	2	4	1	3
f_8	2	3	1	4
f_9	2	4	1	3
f_{10}	2	4	1	3

that the small number of coefficients were large in magnitude. These coefficients obtained large values in order to compensate for the lack of good quality terms in the gbest position. The penalty term of the gbest is plotted in Figure 6 and the average penalty terms of the lbests is plotted in Figure 7. The close close mirroring of the behaviour in Figure 7 to that in Figure 6 reinforces the idea that the lbests of the particles quickly take on the value of the gbest.

Analysing the swarm diversity over time, it is evident that even though there are exploratory mechanisms in the SBPSO (the addition term in the velocity update equation), the swarm's general trend is to converge too quickly. The rapid spikes in the average Jaccard index seen in Figure 8 are as a result of the mentioned exploratory mechanism, but the effect was not enough to discover the global optimum.

The average Jaccard index of the lbest values seen in Figure 9 also indicate that all lbest positions converged to the found gbest set, confirming that the swarm's exploration capabilities diminished to the point that performance was negatively impacted. This once again

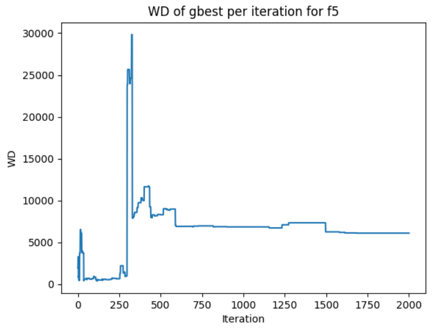

Fig. 6. Weight decay term of gbest

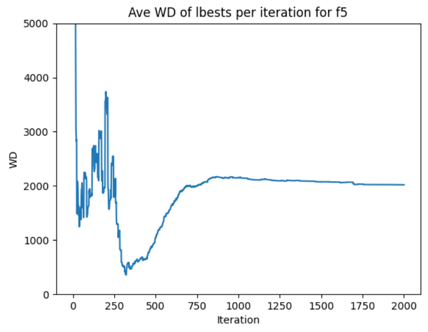

Fig. 7. Average weight decay term of lbests

indicates that the swarm was unable to sufficiently explore the search space to discover the true optimum.

In order to further explain the exploratory abilities of the SBPSO, the frequency with which terms appear in the gbest needs to be examined. Figure 10 shows the number of times each term appears in the gbest's position, using a unique index assigned to each monomial. It is apparent that certain terms dominate the gbest in each iteration, while other terms do not get added to the gbest very often. This is once again attributed to the SBPSO's lack of exploration on this problem, as it shows not all

TABLE V

POLYNOMIALS FOUND BY SBPSO

F	Target polynomial	Found polynomial
f_1	$x_1^3 + x_2^2 + x_2 + 1$	$0.99x_1^3 + 0.99x_2^2 + 0.99x_2 + 0.99$
f_2	$1.5x_1^3 - 3x_2^2 + 0.4x_2 + 5.5$	$1.5x_1^3 - 2.99x_2^2 + 0.39x_2 + 5.15$
f_3	$15x_1^3 - 20x_2^2 + 11.5x_2 + 5.5$	$15x_1^3 - 20x_2^2 + 11.48x_2 + 5.4$
f_4	$0.5x^3 + 2x^2 - x$	$0.49x^3 + 2x^2 - x$
f_5	$3x^7 - 9x^5 - x^4 + 6$	$3x^7 - 9x^5 - 0.99x^4 + 4.6$
f_6	$3x^8 - 9x^7 + 6.5x^6 - 3x^5 - x^4 + x^3 + 0.5x^2 + x - 7$	$2.99x^8 - 8.99x^7 + 6.49x^6 - 2.93x^5 - 1.33x^4 + 10.5x^2$
f_7	$x_1^2 - 2x_2^2 + 3x_1^2x_3^2 - 1.5x_3$	$x_1^2 - 2x_2^2 + 2.99x_1^2x_3^2 - 1.5x_3 - 0.00x_1x_3$
f_8	$3x_1^6 + 3x_2^5 + 3x_3^4 - 2x_1 - 7x_2^2x_3^2 - x_3 + 5.5$	$2.99x_1^6 + 3.0x_2^5 + 2.99x_3^4 - 0.0x_1^3 - 7.0x_2^2x_3^2 - 0.91x_3 + 5.5$
f_9	$-3x_1 - 3x_2^2 - 3x_3^2 - 3x_4^2 - 3x_5^2 + 2x_1x_2x_4x_5 + 5x_3x_4 - 6.2$	$-2.99x_2^2 - 54.35x_4^1 - 2.75x_5^2 + 2x_1x_2x_4x_5 + -2.4x_3x_5 + 3.4x_5$
f_{10}	$10x_1^5 - 7x_3^4 + 3x_2^3 - 8x_6^2x_7^2x_8^2 - 2.5x_8x_9x_{10} + 3$	$-0.002x_1x_4x_5x_7x_8x_{10} + 2.24x_3^6x_9^6 - 6952.68x_1x_4 - 7.99x_6^2x_7^2x_8^2 - 2.73x_1x_2x_6x_7x_8$

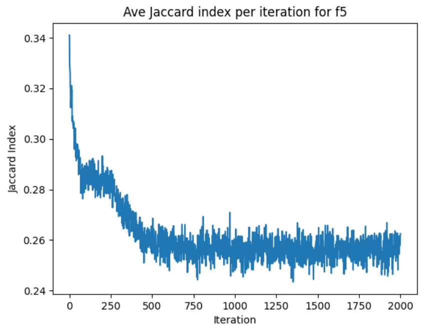

Fig. 8. Swarm diversity as measured by average Jaccard index

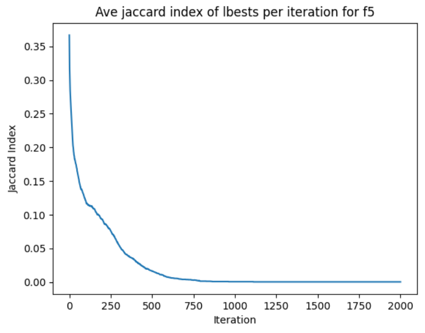

Fig. 9. Lbest position diversity as measured by average Jaccard index

possible term combinations were tested well.

VI. Conclusions and Future Work

The purpose of this paper was to propose a novel set-based approach to polynomial regression using well-established optimization algorithms. The proposed hybrid algorithm consisted of the SBPSO algorithm to solve for the optimal combination of monomials in the polynomial and the ACD algorithm to solve for the optimal coefficients of these monomials.

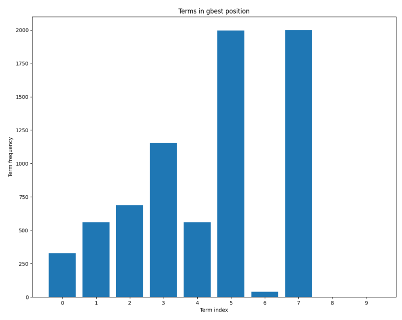

Fig. 10. Frequency of terms in gbest

The application of the SBPSO and ACD shows promise as a well suited set-based solution for polynomial regression. Preliminary results show good performance on low dimensional and low order polynomials where the universal set size remains relatively small, with some performance drawbacks with larger search spaces.

The proposed algorithm also provides the advantage over existing algorithms in that it has easily interpretable results and the most potential for improvement and scalablity due to being based on the well studied field of set theory.

In many of the problems tested, at least one of the independent runs approximated the source polynomial structure correctly, as seen in Table V and by the large deviations in Table III. This indicates that is able to solve more complicated regression problems, but not consistently.

Future work includes further testing the descriptive capabilities of the SBPSO algorithm when applied to real-world data-sets. The algorithm can also be extended to work in dynamic environments by introducing quantum-PSO inspired effects.

Acknowledgment

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

References

[1] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems*. Morgan-Kaufmann, 1992, vol. 4, pp. 950–957.

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

[3] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, 1997, pp. 4104–4108 vol.5.

[4] G. Pampara, N. Franken, and A. P. Engelbrecht, "Combining particle swarm optimisation with angle modulation to solve binary problems," in *2005 IEEE Congress on Evolutionary Computation*, 2005, pp. 89–96 Vol.1.

[5] F. Van Den Bergh and A. P. Engelbrecht, "An analysis of particle swarm optimizers," Ph.D. dissertation, University of Pretoria, 2002.

[6] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence*, 1998, pp. 69–73.

[7] J. Langeveld and A. P. Engelbrecht, "Set-based particle swarm optimization applied to the multidimensional knapsack problem," *Swarm Intelligence*, vol. 6, pp. 297–342, 2012.

[8] K. Erwin and A. P. Engelbrecht, "Set-based particle swarm optimization for portfolio optimization," in *Swarm Intelligence*, 2020, pp. 333–339.

[9] A. P. Engelbrecht, J. Grobler, and J. Langeveld, "Set based particle swarm optimization for the feature selection problem," in *Engineering Applications of Artificial Intelligence*, vol. 85, 2019, pp. 324–336.

[10] P. Jaccard, "Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines." *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, pp. 241–72, 1901.

[11] I. Loshchilov, M. Schoenauer, and M. Sebag, "Adaptive coordinate descent," in *Genetic and Evolutionary Computation Conference, GECCO'11*, 2011, pp. 885–892.

[12] N. Hansen, "Adaptive encoding: How to render search coordinate system invariant," in *International Conference on Parallel Problem Solving from Nature*, 2008.

[13] A. Fraser, "Simulation of genetic systems by automatic digital computers ii. effects of linkage on rates of advance under selection," *Australian Journal of Biological Sciences*, vol. 10, pp. 492–500, 1957.

[14] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[15] G. Potgieter and A. P. Engelbrecht, "Genetic algorithms for the structural optimisation of learned polynomial expressions," *Applied Mathematics and Computation*, vol. 186, p. 1441–1466, 2007.

[16] D. F. Specht, "A general regression neural network," *IEEE transactions on neural networks*, pp. 568–576, 1991.

[17] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989.

[18] E. T. Oldewage, "The perils of particle swarm optimization in high dimensional problem spaces," in *MSc thesis, University of Pretoria*, 2019.

[19] I. M. Sobol, "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112, 1967.

[20] F. van den Bergh and A. P. Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information Sciences*, vol. 176, no. 8, pp. 937 – 971, 2006.