Hybrid Particle Swarm Optimization and GMDH System

Abstract ***This chapter describes a new design methodology which is based on hybrid of particle swarm optimization (PSO) and group method of data handling (GMDH). The PSO and GMDH are two well-known nonlinear methods of mathematical modeling. This novel method constructs a GMDH network model of a population of promising PSO solutions. The new PSO-GMDH hybrid implementation is then applied to modeling and prediction of practical datasets and its results are compared with the results obtained by GMDH-related algorithms. Results presented show that the proposed algorithm appears to perform reasonably well and hence can be applied to real-life prediction and modeling problems.***

5.1. Introduction

The GMDH is a heuristic self-organizing modeling method which Ivakhnenko [22] has developed for modeling purpose as a rival method of stochastic approximation. GMDH is ideal for complex, unstructured systems where the investigator is only interested in obtaining a high-order input-output relationship [1]. GMDH algorithm can be applied to given data set of a system where it tries to find relation between input data and output data without much interference/involvement of an investigator. Hence this can be treated as a good data mining tool where data is transformed into knowledge for decision making. Data mining is specifically used on those data sets where no priori knowledge is available by applying any appropriate data mining tool to extract some hidden knowledge. GMDH works well for its purpose but it is infected with some shortcomings. Generally we use data mining on very large multi-dimensional datasets. GMDH struggles to find good model solution where dimension is very big because of it combinatorial behavior to solve the problem. A novel algorithm has been proposed here for modeling and prediction purpose that tries to overcome the existing discrepancies of traditional GMDH algorithm. This novel algorithm is actually hybridization of GMDH with an adaptive heuristic Particle Swarm Optimization algorithm. This proposed novel algorithm is named PSO-GMDH which indicates the hybridization of two separate heuristic algorithms. Heuristic optimization algorithms are normally applied to problems where no specific algorithm of a problem exists or a known specific problem has very high time complexity that does not work for large size problems. Since, GMDH is unable to deal with large size problems the ideas of PSO heuristic algorithm has been combined with GMDH algorithm. Specifically, the selection process of individual variables (nodes) in traditional GMDH has been replaced with heuristic selection process of PSO algorithm which also provides the termination criteria of the algorithm.

5.2. The Group Method of Data Handling (GMDH)

5.2.1 Overview of traditional GMDH

This section describes the original GMDH modeling that was proposed by A. G. Ivakhnenko in 1960s. This method is particularly useful in solving the problem of modeling multi-input to single-output data. It just requires the data set of a particular area of application for training and testing in order to realize a mathematical model.

The details of original Group Mehod of Data handling (GMDH) modeling has been described in an article by S. J. Farlow [1]:

GMDH algorithm is basically works on *interpolation* that is used to find the approximate values of a complex function using some other easier function.

The simplest form of interpolation uses a straight line as if it were the given function $f(x)$ whose values are need to be approximated. For this *linear interpolation* the particular straight line was chosen to pass through the two end points of the interval, where we knew the values of the function as shown in Fig. 5.1 [2].

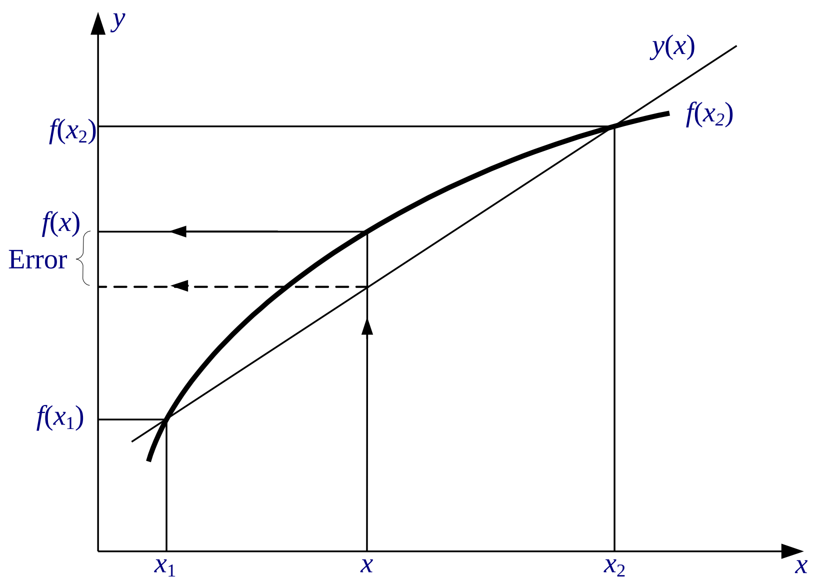


Fig. 5.1 Linear Interpolation.

Function $f(x)$ can be approximated by a straight line through the two points. The line is given by

$$y(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) \tag{5.1}$$

Eq. 5.1 is the desired formula for estimating the value of $f(x)$ from the given value of x.

The values that we are given of a function are sometimes spaced so far apart that linear interpolation is not sufficiently accurate for our purposes. In such cases we use nth order polynomials.

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \tag{5.2}$$

and chooses a polynomial which passes through selected samples i.e. given data set of the function as shown in Fig. 5.2.

In the particular cases of polynomial interpolation in a table of values i.e. data set of a function *y(x)*, the condition that the polynomial pass exactly through the point (x_i , y_i) is that

$$P_n(x_i) = y_i = a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_n x_i^n \tag{5.3}$$

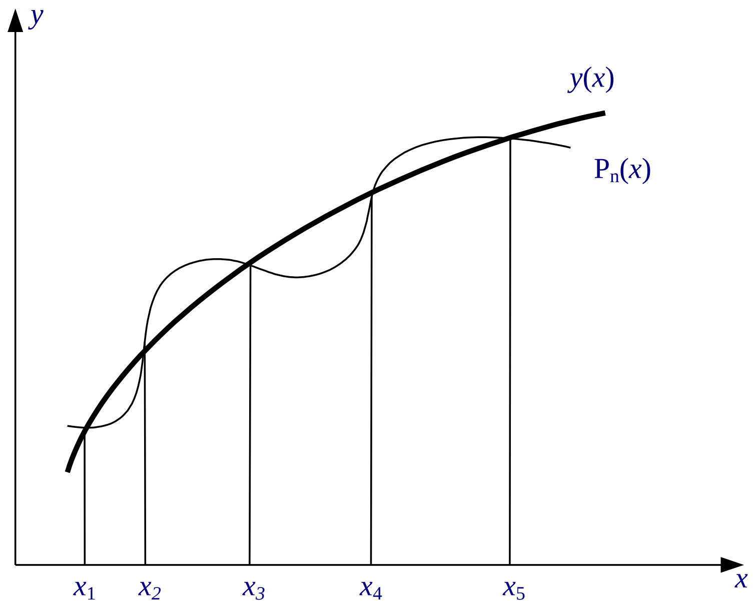


Fig. 5.2 Polynomial Interpolation.

Hence the approximation of y(x) is a polynomial $P_n(x)$ instead of a straight line. With a given set of values the coefficients can be found using mathematical methods which are not given here. Interested readers can find more about interpolation and its applications.

Traditional GMDH uses similar regression methods to find a model from the given data set. It uses a high-order polynomial of the form shown below [1]:

$$y = a + \sum_{i=1}^{m} b_i x_i + \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x_i x_j + \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{m} d_{ijk} x_i x_j x_k + \ldots \tag{5.4}$$

Which relates m input variables $x_1, x_2, x_3, \ldots, x_m$ to a single output variable y.

Preamble: collect regression-type data of n-observations and divide the data into n_{tr} training and n_{te} testing sets as shown in Table 5.1 [5].

Table 5.1 Data Set Format.

	x_{11}	x_{12}	x_{13}	....	x_{1m}	y_1
	x_{21}	x_{22}	x_{23}	....	x_{2m}	y_2
n_{tr}	x_{31}	x_{32}	x_{33}	....	x_{3m}	y_3
	...	...	...	....	....	....
	$x_{nt,1}$	$x_{nt,2}$	$x_{nt,3}$	....	$x_{nt,m}$	y_{nt}
n_{te}	....	....	....	....	....	....
	x_{n1}	x_{n2}	x_{n3}	....	x_{nm}	y_n

Mathematically we have set X of input data set and set Y of output data set.

Step 1: construct ${}^{m}C_2$ new variables $z_1, z_2, z_3, \ldots, z_{\binom{m}{2}}$ in the training dataset for all independent variables (columns of X), two at a time $\left(x_{i,k-1}, x_{i,k} : i \in [1, m] \, and \, k \in \left[2, \binom{m}{2}\right]\right)$ and construct the regression polynomial:

$$z_1 = A + Bx_1 + Cx_2 + Dx_1^2 + Ex_2^2 + Fx_1x_2 \text{ at points } (x_{11}, x_{12}) \tag{5.5}$$

$$z_k = A + Bx_{k-1} + Cx_k + Dx_{k-1}^2 + Ex_k^2 + Fx_{k-1}x_k \text{ at points } (x_{i,k-1}, x_{i,k}) \tag{5.6}$$

Step 2: for each of these regression surfaces, evaluate the polynomial at all n data points (i.e. A, B, C, D, E, and F obtained from $x_{i,k-1}, x_{i,k}; y_i$ for training). The coefficients for the polynomial are found by least square fitting as given in [20], or singular value decomposition (SVD) for singular-value problems as given in [21] using the data in the training set.

Step 3: eliminate the least effective variables: replace the columns of X (old variables) by those columns of Z (new variables) that best estimate the dependent variables y in the testing dataset such that

$$d_k^2 = \sum_{i=nt+1}^{n} (y_i - z_{i,k})^2 \quad \text{where } k \in \left[1,2,\ldots.\binom{m}{2}\right] \tag{5.7}$$

Order Z according to the least square error $d_k \mid \|d_j\| < R$ where R is some prescribed number chosen a priori. Replace columns of X with the best Z's ($Z_{<R}$); in other words $X_{<R} \leftarrow Z_{<R}$

Step 4: Test for convergence. Let $DMIN = d_l$ where l = number of iterations. If $DMIN_l = DMIN_{l-1}$ go to Step 1, else stop the process.

5.2.2 Drawbacks of traditional GMDH

The original GMDH is entailed with some discrepancies such as it generates quite complex polynomial as it progresses through the layers. Every addition of a layer increases the order of polynomial exponentially. Hence it becomes very difficult to keep extending the layers. To avoid the complete explosion of this algorithm GMDH has taken *greedy approach* where it chops off unfit solutions in each layer and only allows predefined number of fit solutions to move to next layer. However, greedy approach itself is inclined to its ineffectiveness in finding global best solution and most of the time it stuck into local best solution. Its usage of greedy selection approach to get the best polynomial makes it bias towards those individual variables that are unfit at early stage but might become fit on later stage. Another drawback can be pointed out on its termination criteria of algorithm where it terminates the algorithm as soon as it receives solution of poor quality. It assumes good solution in past will always give good solution in future which is again *greedy approach* that is not always true.

Besides suffering from few shortcomings, GMDH is still an effective algorithm to determine a model from a given data set. Diminishing its shortcomings might improve its performance. As indicated above that it has two major drawbacks. First is its combinatorial behavior of algorithm which is unlikely to be solved but its impact can be minimized. Generic optimization algorithms are known to be quite effective in solving combinatorial problems where it searches for solution by taking all possible combinations of fit and unfit solutions from one generation/layer/iteration to another generation/layer/iteration. Generic algorithms are based on heuristic approach where it can solve any optimization problem. In the case of GMDH, the selection of nodes from one layer to another to get the optimum model solution is actually an optimization problem where an optimum solution is need to be found from infinite possible solution. Particle Swarm Optimization is one of the generic optimization algorithms which could be applied in GMDH to replace its greedy approach based selection process with heuristic selection approach. The combination of these two separate algorithms into one could be named hybrid PSO-GMDH algorithm.

5.3. Particle Swarm Optimization Algorithm

Particle Swarm Optimization is a heuristic algorithm that solves continuous and discrete optimizing problem of a large domain. It is a generic algorithm; that means it can be used for any discrete/combinatorial problem for which good specialized algorithms do not exist. This algorithm is inspired by the social behavior of bird flocking or fish schooling which is described with its mathematical functions in details in this section.

The idea of optimization plays a major role in engineering, computer science, system theory, economics and other areas of science [6]. Optimization principles are of increasing importance in modern design and system operation in various areas [6]. The contents of this chapter are mostly from our published work in [18].

To solve a particular optimizing problem, algorithms are developed. These algorithms solve only specific problems. Computer scientists often make use of generic algorithms to work on a wide range of problem domain, e.g. tabu search algorithms [25], simulated annealing [26], genetic algorithms [24], ant colony optimization [27] and particle swarm algorithms [7].

Particle swarm optimization (PSO) was originally designed by R. C. Eberhart, and J. Kennedy [7], which solves many kinds of continuous and binary problems of large domain. It is certainly not as powerful as some specific algorithms, but, on the other hand, it can easily be modified for any discrete/combinatorial problem for which good specialized algorithms do not exist [8]. To grasp the abstract phenomenon behind this powerful algorithm, a simple analogy is given.

The algorithm is inspired by the social behavior of bird flocking or fish schooling. The analogy involves simulating social behavior among individuals (particles) “flying” through a multidimensional search space, each particle representing a single intersection of all search dimensions. The particles evaluate their positions relative to a goal (fitness) at every iteration; particles in a local neighborhood share memories of their “best” positions, then use those memories to adjust their own velocities, and thus subsequent positions [9].

The original PSO formulae define each particle as potential solution to a problem in D-dimensional space. The position of particle i is represented as

$$X_i = (x_{i1}, x_{i2}, \dots, x_{iD}) \tag{5.8}$$

Each particle also maintains a memory of its previous best position, represented as

$$P_i = (p_{i1}, p_{i2}, \dots, p_{iD}) \tag{5.9}$$

A particle in a swarm is moving; hence, it has a velocity, which can be represented as

$$V_i = (v_{i1}, v_{i2}, \dots, v_{iD}) \tag{5.10}$$

At each iteration, the velocity of each particle is adjusted so that it can move towards the neighborhood's best position known as lbest (*Pi*) and global best position known as gbest (*Pg*) attained by any particle present in the swarm [9].

After finding the two best values, each particle updates its velocity and positions according to Eq. 5.11 and Eq. 5.12 weighted by a random number *c1* and *c2* whose upper limit is a constant parameter of the system, usually set to value of 2.0 [10]. *c1* is known as cognitive influence factor and *c2* is social influence factor.

$$V_i(t+1) = V_i(t) + c_1 \cdot (P_i - X(t)) + c_2 \cdot (P_g - X_i(t)) \tag{5.11}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{5.12}$$

All swarm particles tend to move towards better positions; hence, the best position (i.e. optimum solution) can eventually be obtained through the combined effort of the whole population.

The methodology of obtaining the velocity vector $v_i(t+1)$ is illustrated in Fig. 5.3, and it has been observed that particle *i* has moved towards the neighborhood's best position and its own best position.

Fig. 5.3 illustrates the operators' modification until the finest form is not obtained. A few points can be noted during the modification: (1) the new position is converging towards the best position region; (2) the velocity could explode towards infinity. Hence new position on that situation is undefined.

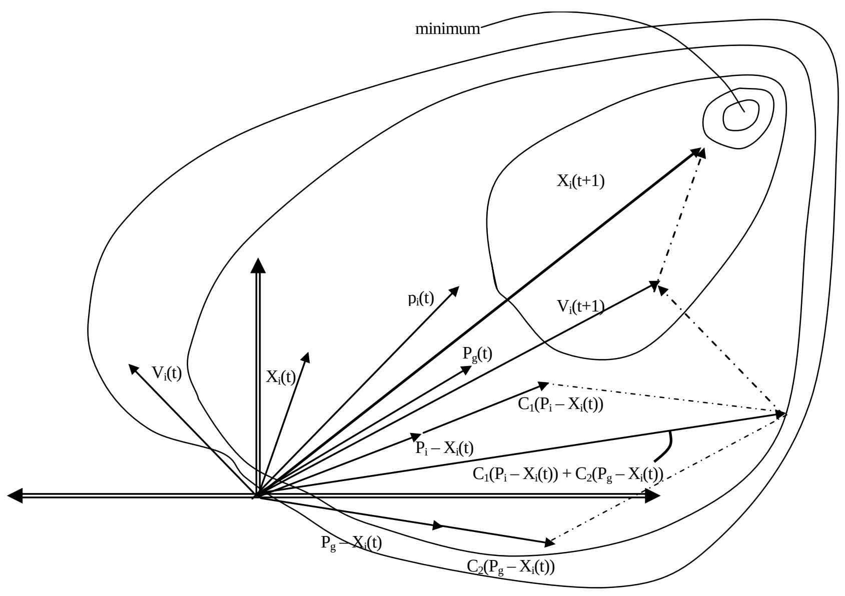


Fig. 5.3: Contour Lines and the Process for Generating New Locations in PSO Scheme

All swarm particles tend to move towards better positions; hence, the best position (i.e. optimum solution) can eventually be obtained through the combined effort of the whole population.

The following section describes how the movement of a particle is restrained through mathematical function to search near the specified space where the solution is likely to be found rather than diverse it away from possible solution space.

5.3.1 Explosion Control

Constriction parameters have a great impact on the operations. Varying these parameters has the effect of varing the strength of the pull “towards” the two best position, which could be verified by observing Fig. 5.3. If accidentally the coefficients *c1* and *c2* exceeds the value 4.0, both the velocities and positions explode towards infinity. Thus almost all implementation of the particle Swarm limit each of the two coefficient *c1* and *c2* to 2.0 [11]. To control the explosion of the system a new constriction coefficient is used, which is called inertia weight. A large inertial weight facilitates global exploration while a small inertial weight tends to facilitate local exploration to fine tune the current search area [12]. Hence, equation 4.4 can be modified to have a place for new constriction constant, inertial weight (α) [13].

$$V_i(t+1) = \alpha \cdot V_i(t) + c_1 \cdot (P_i - X(t)) + c_2 \cdot (P_g - X_i(t)) \qquad (5.13)$$

where inertial weight (α) = $\dfrac{k}{abs\left[\dfrac{1-\frac{a}{2}-\sqrt{|a^2-4a|}}{2}\right]}$

$k \in (0,1]$ and $a = c1 + c2$ such that $a > 4$

Equation 3.2.10 is generic operator for all types of objective functions. This is to tune up these coefficients for any particular kind of problem domain.

5.3.2 *Particle Swarm Optimization Operators*

This section describes all predefined operators that are used to move a particle from one position to another.

5.3.2.1 Position minus Position: subtraction

$(position, position) \xrightarrow{\text{minus}} velocity$

Particles move from one place to another in search for a better position. To move a particle it must have some velocity to move in specified direction (towards better position).

Let say particle p_1 has position x_1 and particle p_2 has position x_2. To move the particle p_1 from position x_1 to x_2 velocity v is given.

$$v = x_2 \Theta x_1 \qquad (5.14)$$

Suppose position x_1 and x_2 are represented in the form of array of dimension 4.

$$x_1 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 0 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

Then the velocity v will be:

$$v = \begin{bmatrix} 0 \\ 2 \\ 3 \\ 1 \end{bmatrix} \Theta \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \leftrightarrow 2 \\ 1 \leftrightarrow 3 \end{bmatrix}$$

Meaning: on x_1, exchange 1 and 2, then exchange 1 and 3. The final result would be x_2. The mathematical idea behind this procedure is to memorize the transformation the velocity will eventually do when applied to position (see "Position plus velocity"). Another important operator is described next.

5.3.2.2 Coefficient times velocity

This operator is stochastic, and defined only for a coefficient between 0 and 1. For a coefficient greater than 1, say *coeff* = *k* + *c*, with *k* is integer part and *c* is decimal part, then simply *k* times velocity_plus_velocity and one times coefficient_times _velocity is used [13].

Suppose a velocity *v* and coefficient *c* is given, then *c* x *v* can be computed for a coefficient *c* given as

$$\begin{cases} c \in [0,1], c' = random(0,1) \\ c' \leq c \Rightarrow (i \leftrightarrow j) \rightarrow (i \leftrightarrow i) \\ c' > c \Rightarrow (i \leftrightarrow j) \rightarrow (i \leftrightarrow j) \end{cases} \tag{5.15}$$

The following example shows how this operator works for a given *v*, 0.5*v* (c < 1 = 0.5) is given as:

$$0.5 \otimes \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 3 & 0 \\ 4 & 4 \\ 0 & 2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 2 & 2 \\ 3 & 3 \\ 4 & 4 \\ 0 & 2 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 \leftrightarrow 1 \\ 0 \leftrightarrow 2 \end{bmatrix}$$

This operator is mainly used to change the velocity. The size of velocity is also changed when this operator is imposed. The operator is helpful when velocity of a particle needs to be changed. Either move away or move closer to another particle. For the previous sections the velocity of the particle was deduced. Now particle has a velocity to move into new position. One other important operator is velocity plus velocity which is discussed below.

5.3.2.3 Velocity plus velocity: addition

This operator is required only when coefficient is greater than one, suppose a particle p has two velocities v_1 and v_2, then the new velocity v_{new} will be:

$$v_1 \oplus v_2 = v_{new} \tag{5.16}$$

The technique of addition is as follows. The sequence of transpositions describing v_2 is simply "added" to the one describing v_1. An example of this operator is shown below.

$$\begin{array}{ccccccc} v_1 & & v_2 & & v_{new} & & v_{new} \\ \begin{bmatrix} 0 \leftrightarrow 1 \\ 2 \leftrightarrow 3 \end{bmatrix} & \oplus & \begin{bmatrix} 0 \leftrightarrow 2 \\ 3 \leftrightarrow 4 \\ 3 \leftrightarrow 1 \end{bmatrix} & = & \begin{bmatrix} 0 \leftrightarrow 1 \\ 2 \leftrightarrow 3 \\ 0 \leftrightarrow 2 \\ 3 \leftrightarrow 4 \\ 3 \leftrightarrow 1 \end{bmatrix} & = & \begin{bmatrix} 0 \leftrightarrow 3 \\ 1 \leftrightarrow 2 \\ 1 \leftrightarrow 4 \end{bmatrix} \end{array}$$

This operator is not commutative, we usually do not have $v_1 \oplus v_2 = v_2 \oplus v_1$.

5.3.2.4 Position plus velocity

Particles can be moved according to their current velocity. Suppose particle *p* has position *x* and velocity *v* then the new position willl be

$$x_{new} = x + v \tag{5.17}$$

This function is used to obtain global best particle of the swarm. Let say particle *p* has position *x*, velocity *v* and new position x_{new}. Suppose postion *x* has some arbitrary values of independent variables (*a, b, c, d, e, f*) and velocity *v* has values (*b, d*) in first component and values (*a, f*) in second component. Then the *x_new* is shown as

$$x \qquad v \qquad x_{new}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} \oplus \begin{bmatrix} b \to d \\ a \to f \end{bmatrix} = \begin{bmatrix} f \\ d \\ c \\ b \\ e \\ a \end{bmatrix}$$

The technique of transformation is that each component of the velocity (that is a transposition) is successively applied, first to x, then to the position obtained.

As discussed above PSO algorithm is based on social psychological metaphor where swarm particles interact with each other in their neighborhood and their whole society. Section 5.3.3 describes the different ways of neighborhood defined in PSO algorithm.

5.3.3 Particle Swarm Optimization Neighborhood

The particle swarm algorithm is an adaptive algorithm based on a social-psychological metaphor. Each particle is influenced by a success of their topological neighbors [17]. This external function provides a particle its neighbor of a given type. Also, there are many ways to define a "neighborhood" [14], but we can distinguish three classes:

5.3.3.1 Social Neighborhood

The social neighborhood, just takes *relationship* into account. In practice, for each particle, its neighborhood is defined as a list of particles at the very beginning, and does not change. Note that, when the process converges, a social neighborhood becomes a physical one. Al social neighborhood is a type when a particle chooses k nearest particles according to its location. Mathematically, social neighborhood is defined for a particle to simply get $k/2$ particles on each side of 1-D array.

5.3.3.2 Physical neighborhood

The physical neighborhood, takes distances into account. In practice distances are recomputed at each time step, which is quite costly, but some clustering techniques need this information in this type of neighborhood, a particle "chooses" k best particles from the entire swarm; the distance between a particle and k particles in the globe is normally calculated. Normally the social neighborhood becomes a physical neighborhood during the process of algorithm.

5.3.3.3 Queen

Instead of using these two types of neighborhood. An extra particle can be used, which "summarizes" the neighborhood. This method is combination of the (unique) queen method defined in [14] and of the multi-clustering method described in [16]. For each neighborhood, we iteratively build a gravity center and take it as best neighbor. This method needs some mathematical computations. The coefficient is obtained as follows:

$$(c_i) = \sum_j \frac{(f_0 + 1)}{(f_0 + 1) + (f_j + 1)} \tag{5.18}$$

Since PSO is an adaptive algorithm based on social-psychological metaphor. The population of individuals adapt by returning stochastically towards previous successful regions in the search space. Move towards is the most important external function in PSO, because it explains the movement of the particles [17].

Sometimes PSO algorithm is unable to output the desired or expected value, then the swarm is known to be in *no-hope* state. Like other generic optimization techniques PSO has some strategies to avoid being stuck in local maximum or minimum. PSO has a well-defined procedure to move out from the *no-hope* state which is called *re-hope*. PSO uses *no-hope* and *re-hope* processes to improve search performances. The following section covers procedures to determine when PSO is in *no-hope* state and the details of *re-hope* process.

5.3.4 Particle Swarm Optimization Improvement Strategies

As in other generic optimization techniques such as genetic algorithm, differential evolution, ant colony optimization, etc., PSO has some strategies to avoid being stuck in local maximum or minimum. For PSO, in particular, the *No-Hope/Re-Hope* process is used in order to improve search performances; in other words, for the search to be adaptive.

5.3.4.1 No-Hope tests

For any objective function of any type, the decision has to be made about the optimum value. The PSO algorithm has a task to decide whether the optimum value is achievable or not, and if achievable than how good is it.

If the PSO algorithm is unable to output the desired or expected value, then the swarm is in *no hope* state. There are some reasons behind this status. Firstly, if not single particles are moving then there is no question of movement of swarm, hence no better position is expected. Secondly, no effective movement is occurring i.e. either the swarm has reduced very much or the movement is extremely low. Finally when the algorithm is producing the same best value greater than the *threshold* times, then the better solution than the *same best* value is unexpected and the swarm is in *no hope* state. When the PSO algorithm gets into a *no-hope* state, the only way out is to either accept the result of the current situation or re-hope for better result. The following criteria are useful for the *no-hope* test [15].

Criterion 0

If a particle has to move *towards* another one, which is at distance 1, wither it does not move at all or it goes exactly to the same position as this other one, depending on the social/confidence coefficients. It may be possible that all moves computed according to equation 16 are null. In this case there is absolutely no hope to improve the current best solution.

Criterion 1

The *No-Hope* test defined in [15] is *swarm too small*. In this test, the swarm diameter is computed at each time step, which is costly. But in a discrete case, as soon as the distance between two particles tends to become too small, the two particles become identical, usually first by positions and then by velocities. Hence, at each time step, a *reduced* swarm is computed in which all particles are different, which is not very expensive, and the No-Hope test becomes *swarm too reduced*, say by half the original size.

Criterion 2

Another criterion has been added to the *no-hope* test criterion is the *swarm too slow*. This criterion compares the velocities of all particles to a threshold, either individually or globally. In one version of the algorithm, this threshold is in fact modified at each time step, according to the best result obtained so far and to the statistical distribution of arc values.

Criterion 3

Another very simple criterion that is defined is the *no improvement for too many times*. In practice, it appears that criteria 1 and 2 are sufficient.

5.3.4.2 Re-Hope Process

PSO has a well-defined procedure to move out from the *no-hope state* is called *Re-hope*. As soon as there is *no hope*, the swarm is re-expanded. The idea behind this method is to check if there is still hope to reach the better solution. If there is no hope, then swarm is moved [15]. The particles are moving slowly and continuously to get better position. The movement of particles is categorized in four: (i) lazy descent method, (ii) energetic descent method, (iii) local iterative leveling, and (iv) Adaptive re-hope method. There are a number of re-hope strategies defined for PSO. The re-hope strategies in [15] and [19] inspire the first two methods described here.

5.3.4.2.1 Lazy Descent Method (LDM)

Each particle goes back to its pervious best position and, from there, moves randomly and slowly (i.e. size of velocity is 1) and stop as soon as it finds a better position or when a maximum number of moves (problem size) is reached. If the current swarm is smaller than the initial one, it is completed by a new set of particles randomly chosen.

5.3.4.2.2 Energetic descent method (EDM)

Each particle goes back to its previous best position and, from there moves slowly (i.e. velocity size is 1) as long as it finds a better position in at most maximum number of move (problem size). If the current swarm is smaller than the initial one, it is completed by a new set of particles randomly chosen. The only drawback of this method is it is more expensive than LDM.

5.3.4.2.3 Local Iterative Leveling (LIL)

This method is more expensive and more powerful. This method is used when EDM fails to find a better position. For each immediate physical neighbor y (at distance 1) of the particle p, a temporary objective function value f(y) is computed by using the following algorithm:

- Find all neighbors at a distance 1;
- Find the best distance, i.e. $y_{\min}$;
- Assign y to the temporary objective function value $f(y) = \frac{f(y_{\min}) + f(x)}{2}$;
- Move p towards its best neighbor;

Usually this algorithm's big O order ranges from polynomial to exponential hence this procedure is only used when the swarm is in no-hope state [15].

5.3.4.2.4 Adaptive Re-Hope Method (ARM)

The three above methods can be automatically used in an adaptive way, according t how long (number of times steps) the best solution has not been improved. An example of adaptive *re-hope* strategy is shown in Table 4.1.

Table 5.2: Adaptive Re-Hope conditions

Same best*	Re-Hope type
0	No Re-Hope
1	Lazy Descent Method (type = 0)
2	Energetic Descent method (type = 1)
≥ 3	Local Iterative Leveling (type = 2)

*Number of time steps without improvement.

5.3.4.2.5 Parallel and Sequential Versions

The algorithm can run either in (simulated) parallel mode or in sequential mode. In the parallel mode, at each time step, new positions are computed for all particles and then the swarm is globally moved. In sequential mode, each particle is moved at a time on a cycling way. So, in particular, the best neighbor used at time t+1 may be not anymore the same as the best neighbor at time t, even if the iteration is not complete. Eq. 5.12 implicitly supposes a parallel mode, but in practice there is no clear difference in performances, and the sequential method is a bit less expensive.

5.4. The proposed hybrid PSO-GMDH Algorithm

5.4.1 Overview

Particle Swarm Optimization is used to solve any optimizing problems whereas GMDH itself is a self-organizing modeling method which heuristically determines the optimum model of a given problem. The reason for bringing the idea of hybridization was to overcome the existing shortcomings of traditional GMDH which is mainly of its combinatorial behavior of progressing through layers to find the optimum model of a given problem. Firstly, it makes the assumption that good approximation quality in the past guarantees the good approximation in the immediate future which is a *greedy approach* [3]. As discussed in section 5.2.1 GMDH only picks pre allocated number of best solutions of the current layer to move to next layer. It ignores all other solutions which are unfit in early stages but might generate very fit solution in later stages. Hence choice of solution is always locally best [4]. That means traditional GMDH has high chances of being trapped into local best solution. Secondly, the termination condition of traditional GMDH process depends on the quality of output value in layer by layer approach of nodes selection. It also uses a *greedy approach* that keeps track of local best solutions in the current layer. The iteration process is stopped as soon as new layer generates poorer solution than previous layer. GMDH tries to refine the model in each layer until the best estimation of the model that predicts the output with least error is obtained. The process of termination condition is shown in Fig. 5.4. The Y axis shows the best solution in the corresponding layer. Initially the curve declines then starts inclining after iteration 5 which indicates GMDH was getting better solutions up to iteration 5 then the poor solution is obtained in iteration 6. Hence the iteration has to be stopped in iteration 5.

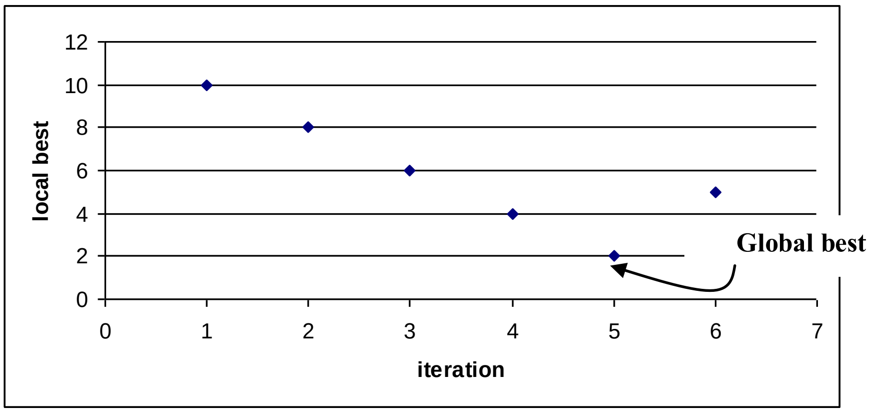


Fig. 5.4 Termination Criteria of Traditional GMDH

The principal approach of modeling of PSO-GMDH and traditional GMDH is more or less same. In PSO the whole population (of constant size) of swarm particles progresses iteratively until the optimum solution is found. Iterative process of PSO can be compared with layered approach of GMDH where swarm particles search for better position in each iteration as GMDH nodes look for better solution in each layer.

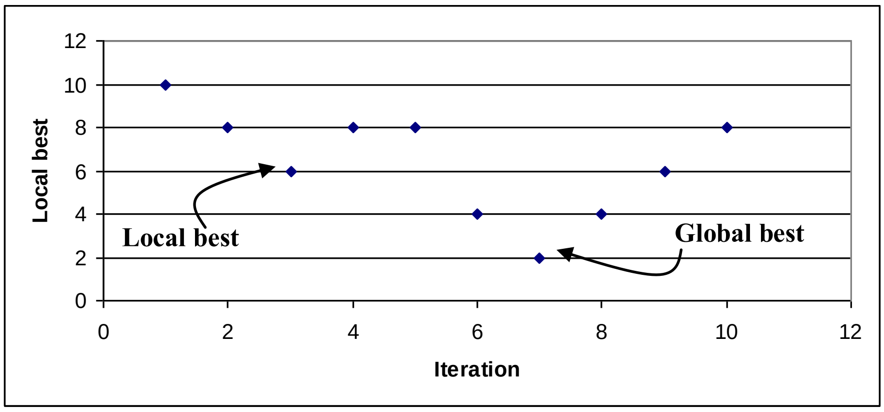


Fig. 5.5 Termination Criteria of Hybrid PSO-GMDH Algorithm

The proposed hybrid PSO-GMDH uses a heuristic search process which makes it more attractive for efficiently searching for large and complex search spaces. Fig. 5.3 shows how PSO can achieve global best solution by the combined effort of the whole population. It is likely that solution found by traditional GMDH is trapped into local minimum whereas PSO's domain of search space is infinitely large and it has its internal mechanism to avoid being trapped into local minimum. The tendency of PSO is that each particle keeps searching for better solution. Unlike traditional GMDH PSO doesn't stop the search process if the best solution of the next layer is not better than previous layer. Fig. 5.5 shows that PSO-GMDH uses its own termination conditional which doest not follow *greedy approach*. Unlike GMDH which moves only predefined number of fit solutions to the next layer, PSO's selection process is based on heuristic approach where combination of fit and unfit solutions are moved to next layer. In every iteration it keeps looking for better solution and doesn't stop the search process even after getting unfit solution because these unfit solutions at initial stage can make fitter solution at the later stage.

Now the question that is raised here is in which part of GMDH process could PSO be applied? In GMDH the model is created through making polynomials of various orders and combinations of features which approximate the given output. Creation of polynomials is part of an iterative process where some given combinations of nodes from current layer make the polynomials for next layer. These polynomials are candidates for model functions which are then used to calculate the output values that are used as input nodes for next layer. So we have the iterative approach where generated outputs for current layer work as input nodes for next layer of GMDH network. Traditional GMDH selects all possible combinations of nodes from previous layer to make polynomials for next layer. Out of which only best *M* (predefined number) is selected to stay in the next layer. This step is necessary to avoid the explosion of combinatorial behavior of the algorithm. This selection process is the first drawback of traditional GMDH. Suppose a given dataset has very large dimension *D* and GMDH algorithm tends to move *D* nodes from one layer to another. Suppose it combines *d* nodes from previous layer to make a polynomial for current layer then it will end up in making total of ${}^{D}C_{d}$ polynomials from which best *D* will be selected. In every iteration total of ${}^{D}C_{d}$ combinations would prove very expensive in terms of processing time and memory. For example if *D* is 50 and *d* is 4 for each layer then there are total of ${}^{50}C_{4}$ i.e. 230,300 polynomials will have to be made from different combinations of 4 nodes for each layer which will then be sorted according to their fitness value and from which only best 50 will be picked to move to next layer. On the other hand PSO-GMDH would heuristically determine which combinations of nodes will be moved to next layer. If PSO generates *k* particles then each particle will specify its set of nodes from which a polynomial will be made. After generating total of *k* polynomial best 50 polynomials (or new nodes) will be moved to next layer. Hence in each layer we will have same number of nodes. Suppose we have 100 particles in the above case then we have maximum of 100 possible combination of nodes/layer from which we will choose 50 nodes (combination of fit and unfit nodes) which is far less than 230,300 nodes generated in traditional GMDH. Hence PSO-GMDH diminishes the curse of dimensionality phenomenon.

To solve this problem and additionally provide greater variations in selection of nodes, PSO comes into picture. The detail selection process is described in section 5.4.2 but in brief it provides the list of nodes from current layers to be combined to make polynomials in the next layer. The selection process doesn't have to take every possible combination of previous layer's data as traditional GMDH does. PSO-GMDH generates a limited workable amount of particles that remain constant throughout the modeling process. Besides, PSO-GMDH offers a new selection process where PSO stochastically determines how many nodes need to be combined and which nodes need to be combined to form the polynomials. The selection is based on heuristic manner hence combinations of fit and unfit individuals from previous layers are selected. Unfit individuals can survive the selection process for next layer if they produce fit solutions. Only *D* polynomials (not necessarily best ones) will be retained for next layer as traditional GMDH does. Then rest of modeling process is same as mentioned above that it will take these calculated polynomials as the input nodes for next layer. So even after applying PSO to manage the selection of nodes the core of GMDH process remains same.

Traditional GMDH makes the assumption about the stopping criteria that good approximation quality in the past guarantees the good approximation in the immediate future [3]. It looks for better approximation in each iteration until it gets poorer solution which indicates no better solution is possible in the coming layers. Hence the search for the solution stops as soon as poorer solution compared to previous layer is computed. This kind of approach is known as greedy approach where an algorithm concentrates on arriving at a solution by making a sequence of choices, each of which simply looks the best at that iteration/layer. Hence choice of solution is always locally best [4]. On the other hand PSO-GMDH uses different termination criterion. Its termination criterion totally depends on PSO's internal heuristic approach of finding the global best solution. Its optimization process determines when to stop the search for solutions. The search process can also be terminated after reaching maximum number of iterations predetermined by the user or if the system reaches *no-hope* stage where no further better solution is possible.

5.4.2 Technical view

PSO-GMDH is simply a GMDH process where selection of nodes to move to next layer is determined by PSO. Technically speaking PSO-GMDH runs two algorithms simultaneously and interacting with each other. PSO tells GMDH which variables (nodes) to select for next layer then GMDH uses those selected nodes to make polynomials for its modeling process. GMDH then returns the computed output value of polynomials back to PSO which it uses as fitness function that is used to determine the quality of polynomial (model function). Progress in each iteration of PSO makes GMDH to progress one layer.

This section describes the technical view of PSO-GMDH algorithms using the following notations:

P_T	Total number of particles used in PSO-GMDH algorithm
M	Total number of nodes to be moved from one layer to another in traditional GMDH algorithm.
X	Only input variables of data set.
Y	Only output variable of data set.
x_i	i^{th} feature of data set.
n_{tr}	size of training data set.
n_{te}	size of testing data set.
n	size of dataset.
P_1, P_2, P_3	System variables exists in each particle.
c_i	i^{th} coefficient of a polynomial function.
E_j	mean square error between actual output and output from j^{th} polynomial.

The PSO-GMDH algorithm starts with generation of swarm particles of size P_T which is normally greater than constant M the number of variable to move from one layer to another. Each particle is initialized with some random positions which are updated on each iteration by PSO. Now the question that is raised here is how every layer of GMDH process integrates with PSO (iterative process) process? The detailed framework of PSO-GMDH algorithm is given below which describes the step by step process:

5.4.2.1 System's input variables:

Modeling of a system requires its corresponding data set. Modeling is a kind of knowledge discovery from existing data set where we need to have some predefined number of features and several combinations of different features making up the dataset. The dataset is consists of input variables (i.e. feature vector) and one output variable only. For a given input vector $X = (x_1, x_2, x_3, \dots, x_n)$ there has to be one output variable Y. $x_1, x_2, x_3, \dots, x_n$ indicate different feature or input variable of the dataset.

5.4.2.2 Forming training and testing data:

As mentioned above modeling required a dataset from which some knowledge has to be discovered. The quality of the derived model has to be tested with separate set of dataset which has not been used in derivation of a model. Hence the derivation of a model requires two separate sets of dataset of same feature vectors. One data set is used to make the models and one is used to test those models. The separate sets of datasets can be made simply by dividing the given dataset of size n into two sets of training and testing data sets whose sizes can be denoted by n_{tr} and n_{te} respectively, where total size $n = n_{tr} + n_{te}$. Training data set is used to construct various candidates of model functions whereas testing data set is used to evaluate the quality of those models at each iteration of PSO-GMDH.

5.4.2.3 Network realization:

The traditional GMDH is based on a heuristic approach where it selects some predefined number M of relatively fit individuals from one layer to another until the termination condition is met. It keeps track of local best solutions in each layer until it finds poorer solution than previous layer which is the termination criterion for the algorithm. As mentioned above that GMDH tries to refine the model in each layer until the best estimation of the model that predicts the output with least error is obtained. The principal approach of modeling of PSO-GMDH and traditional GMDH is more or less same. In PSO the whole population (of constant size) of swarm particles progresses iteratively until the optimum solution is found. Iterative process of PSO can be compared with layered approach of GMDH where swarm particles search for better position in each iteration as GMDH nodes look for better solution in each layer.

As mentioned above in section 5.3 PSO uses swarm particles as its unit search agent. The position (sequence of numbers) of each particle is used to determine which salient combinations of input variables of previous layer to be moved to next layer. Each particle must contain 3 system parameters (P_1, P_2, P_3).

$P_1 \in [1, 3]$ is randomly generated and represents the order of polynomial that will be generated from previous layer, which can be either 2 or 3 but for simplicity we normally take 2 in each layer. The advantage of doing this is to generate more complex universe of data permutations [5]. Traditional GMDH normally just combines 2 or 3 variables from previous layers to next layers but PSO-GMDH varies with different combinations from one layer to another. $P_2 \in [1, r], r = \min(D,5)$ is again randomly generated number which determines number of input variables to be taken from previous layer where D is the width of the input dataset; the default lower bound is r =2. $P_3 = \{a \in Z^+ \mid 1 \le a \le D\}$ is a sequence of integers stored as position of a particle represents the entire candidates in the current layer of the network. The determination of combinations of nodes to be moved to next layer is determined by all three parameters. Firstly *P₁* determines the order of polynomial to be formed then *P₂* determines the number of combinations of nodes forming the polynomial which in turn will be selected from *P₃* as first *P₂* nodes from the whole sequence. Fig. 5.6 depicts the usage of all 3 system parameters to determine the polynomial. The polynomial has the following form:

$$y = a + \sum_{i=1}^{m} b_i x_i + \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x_i x_j + \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{m} d_{ijk} x_i x_j x_k + \ldots$$

Using the above form the polynomial of order 2 can be written in the following simplified form which is normally used in GMDH:

$$y = A + Bu + Cv + Du^2 + Ev^2 + Fuv$$

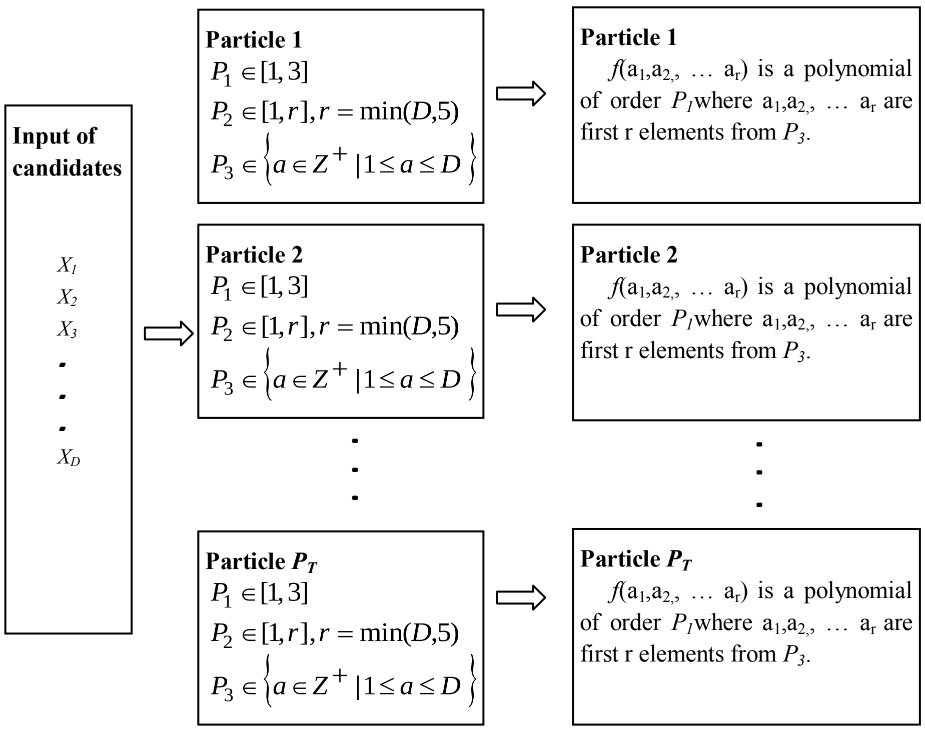

Input of candidates
X_1
X_2
X_3
X_D
Particle 1
$P_1 \in [1,3]$
$P_2 \in [1,r], r = \min(D,5)$
$P_3 \in \{a \in Z^+ \mid 1 \le a \le D\}$
Particle 2
Particle P_T
$f(a_1,a_2, \ldots a_r)$ is a polynomial of order P_1 where $a_1,a_2, \ldots a_r$ are first r elements from P_3.

Fig. 5.6 Formation of Polynomials by Swarm Particles using 3 System Parameters P_1, P_2 and P_3.

Fig 5.6 shows PSO-GMDH process of each layer. Each particle consists of separate set of system parameters which are used to generate the polynomial. These polynomials are actually used as objective function for PSO. It is a candidate for model function which determines how promising the model is.

***Generic Behavior of PSO*:** PSO is a generic optimizing algorithm which requires two external information of a given problem, its input dataset and a tailor made objective function of the problem. Fig 5.7 shows the generic behavior of PSO and how GMDH has fit-in into PSO to make it hybrid PSO-GMDH. To find the model of a given problem a dataset of that problem is fed into PSO-GMDH system where PSO interacts with GMDH which behaves as objective function of PSO to produce the solution. The solution is a polynomial function (or a model) that can predict very closely to the output of the given problem.

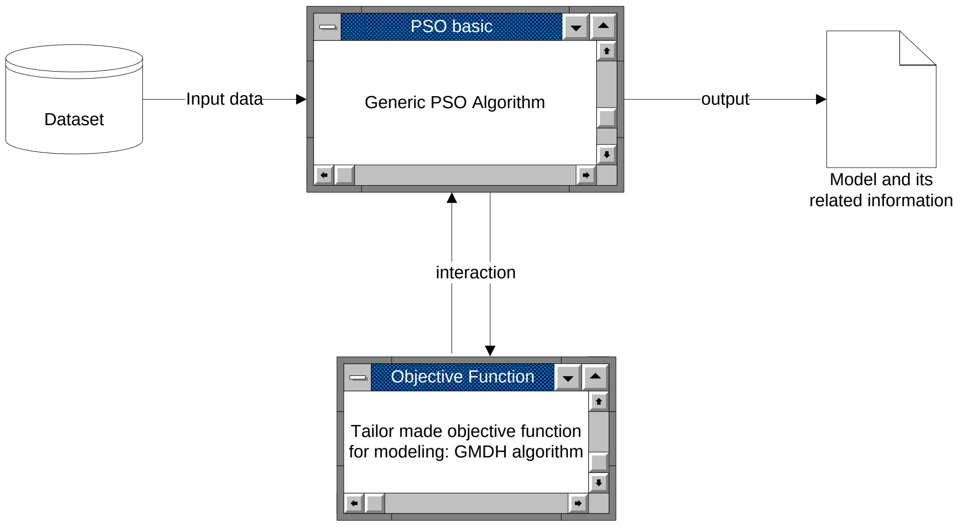
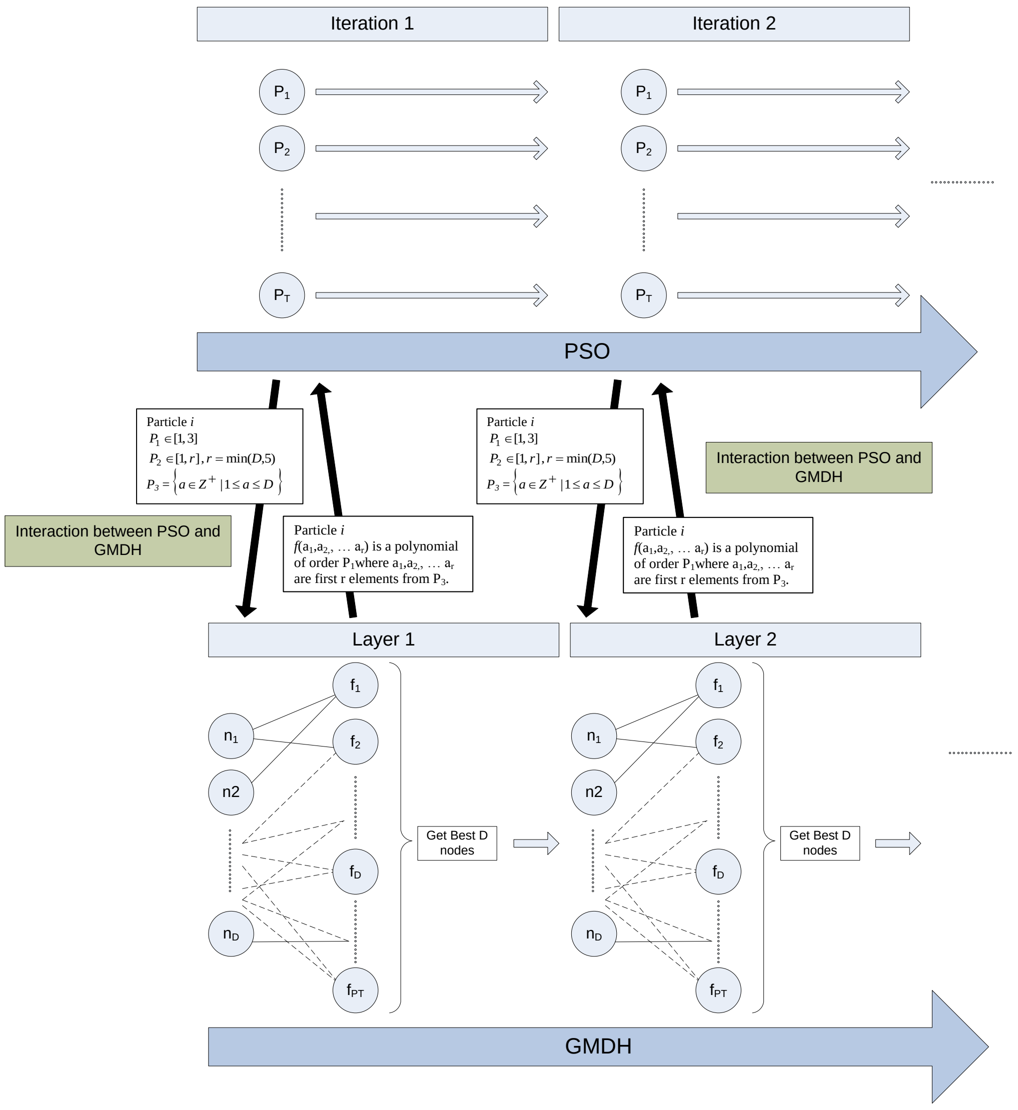

Dataset
Input data
PSO basic
Generic PSO Algorithm
output
Model and its related information
interaction
Objective Function
Tailor made objective function for modeling: GMDH algorithm

Fig. 5.7 Generic Behavior of PSO depicted using GMDH as a tailor made objective function

***Parallel Process*:** the most important part of the hybrid PSO-GMDH is interaction between PSO algorithm and GMDH algorithm. Conceptually, both algorithms run in parallel and interact with each other as shown in Fig. 5.8. It shows that each iteration of PSO corresponds to one layer of GMDH. The particles of PSO carry all 3 system parameters which determine which nodes of GMDH would make up the polynomial; therefore PSO's particles send this set of information to the nodes of GMDH where it picks only those combinations of nodes that have been indicated by PSO to make up the polynomial. The functional value of these polynomials is then compared with the given output and then error value is returned back to particles of PSO which use them as their fitness value for objective function.

It should not be misunderstood that we have confluence of two separate unaltered algorithms PSO and GMDH that just interacts with each other in parallel where iteration *i* of PSO corresponds to layer *i* of GMDH. Actually the selection of nodes process from GMDH has been replaced with PSO algorithm that provides the information for selection of nodes to move to next layer. Even the stopping criterion of GMDH is replaced with generic stopping criteria of PSO. Those researchers who have experience with heuristic algorithms can view this novel hybridization as transformation of GMDH algorithm into objective function of PSO where PSO controls the termination criteria as well as the fitness function (here GMDH algorithm). Each call to fitness function results in polynomial formation of nodes that returns the fitness value (mean square error between polynomial functional value and actual output value) back to PSO algorithm.



Iteration 1
Iteration 2
P1
P2
PT
PSO
Particle i
$P_1 \in [1,3]$
$P_2 \in [1,r], r = \min(D,5)$
$P_3 = \{a \in Z^+ \mid 1 \le a \le D\}$
Interaction between PSO and GMDH
Particle i
f(a1,a2, … ar) is a polynomial of order P1 where a1,a2, … ar are first r elements from P3.
Layer 1
Layer 2
n1
n2
nD
f1
f2
fD
fPT
Get Best D nodes
GMDH

Fig. 5.8 Hybridization of GMDH and PSO shown as parallel process

5.4.2.3.1 Illustrative Example

To understand network realization let us take an example. Suppose we have input dataset of 5 dimensional vectors. Each vector has only one output. PSO-GMDH process would require number of particles to be specified by the user. We can take 10, 15, 20 or any other appropriate size of the swarm particles. Here we can take just 5 for simplicity. Take a scenario where a particle determines the type of polynomial for a current iteration using its 3 system parameters namely P_1, P_2 and P_3. Suppose value of P_1 is 2 which means polynomial will be of order 2. Value of P_2 is also 2 which will tell the particle to choose first 2 integers from the sequence of integers stored in parameter P_3 (recall that P_3 contains the position of a particle). Various possible values of P_3 for all 5 particles representing positions of particles at any given instance are given in Table 5.3 together with dummy Error values E_k i.e. mean square error between actual output and output from polynomial $f(x_i, x_j)$. PSO-GMDH uses these error values for its fitness value, hence the polynomial with least error value is the optimum solution.

Table 5.3: Attributes of Particles in first layer

Particles	**Parameter P_3 (or position of corresponding particles)**	**Selected first P_2 i.e. 2 nodes to make a polynomial**	***E_k = MSE btw $f(x_i, x_j)$ & actual output***
Particle 1	1, 3, 5, 4, 2	1, 3	15.2
Particle 2	5, 4, 3, 2, 1	5, 4	4.5
Particle 3	**2, 1, 3, 5, 4**	**2, 1**	**8.9**
Particle 4	4, 3, 1, 5, 2	4, 3	5.2
Particle 5	1, 5, 2, 4, 3	1, 5	7.2

For instance, the generated polynomial for particle 3 in the first iteration would be:

$f(x_2, x_1) = c_1 + c_2 x_2 + c_3 x_1 + c_4 x_2 x_1 + c_5 x_2^2 + c_6 x_1^2$ where c1, c2, ... c6 are the constants evaluated using training dataset.

Hence Particle 3 has following attributes:

$P_1 = 2$

$P_2 = 2$

$P_3 = \{2, 1, 3, 5, 4\}$

$f(x_2, x_1) = c_1 + c_2 x_2 + c_3 x_1 + c_4 x_2 x_1 + c_5 x_2^2 + c_6 x_1^2$

E_3 = mean square error between actual output and the output from poly nomial $f(x_2, x_1)$. Lets say the error value is 8.9.

In the same manner other particles also assign the values for their attributes.

The step by step process of PSO-GMDH using this example is described next. PSO-GMDH is also compared alongside with traditional GMDH.

Layer 1 of the process:

All possible combinations of nodes and their corresponding dummy error values of the first layer are shown in Table 5.4.

Table 5.4: All combinations of nodes

Order No	x_i	x_j	E_k *val for* $f(x_i, x_j)$
1	**1**	**2**	**8.9**
2	1	3	15.2
3	1	4	6.6
4	1	5	7.2
5	2	3	6.0
6	2	4	7.0
7	2	5	6.0
8	3	4	5.2
9*	3	5	4.0
10	4	5	4.5

Traditional GMDH uses the *greedy* approach to pick the best 5 nodes which is shown in Table 5.5 (a) with grey color whereas PSO heuristically determine which 5 nodes to be picked. Each swarm particle contains 3 main system parameters where parameter P_3 contains information about which nodes will be picked for next iteration as shown in Table 5.3. Table 5.5 (b) shows picked nodes from all available nodes in grey color. It could be noted that nodes combination of 3 and 5 of order No 9 is the best polynomial which is picked by traditional GMDH algorithm but not by PSO-GMDH. Nodes combination of 4 and 5 is the best polynomial for PSO-GMDH at this stage. It is not necessary that PSO will pick the best 5 nodes of current layer. It can pick non promising solutions initially which might become very promising later on. Observe a node of order No 1 which has been rejected by traditional GMDH but is taken by PSO-GMDH. On later stage this rejected (or taken) node becomes the best node. Table 5.5(b) shows that PSO-GMDH has picked a node of order 1 combined through (x_1, x_2) which is not promising at this stage.

Table 5.5 (a): Traditional GMDH Approach

Order No	x_i	x_j	E_k *val for* $f(x_i, x_j)$ **in sorted order**
9*	3	5	4
10	4	5	4.5
8	3	4	5.2
5	2	3	6
7	2	5	6
3	1	4	6.6
6	2	4	7
4	1	5	7.2
1	**1**	**2**	**8.9**
2	1	3	15.2

*best value at this stage

Table 5.5 (b): PSO-GMDH Approach

Order No	x_i	x_j	E_k *val for* $f(x_i, x_j)$ **in sorted order**
9	3	5	4
10*	4	5	4.5
8	3	4	5.2
5	2	3	6
7	2	5	6
3	1	4	6.6
6	2	4	7
4	1	5	7.2
1	**1**	**2**	**8.9**
2	1	3	15.2

*best value at this stage

Layer 2 of the process:

For second layer PSO-GMDH would use the nodes shown in Table 5.6 derived from Table 5.5 (b). Observe the new order numbers after sorting the selected nodes. Now the new order No of node 1 is 4.

Table 5.6: PSO-GMDH Approach: Chosen candidates for next iteration

Prev Order No	***New Order No***	x_i *or* x_j
10*	1	4.5
8	2	5.2
4	3	7.2
1	**4**	**8.9**
2	5	15.2

*best value at this stage

The next iteration of PSO will move its particles from one position to another with calculated velocities in a search space therefor the value of parameter P_3 for each particle will change (recall P_3 stores the current position of a particle). The new attributes for the particles are shown in Table 5.7. The non-promising node of order no. 4 has combined with node 2 and has given a promising node. This new node is shown with bold letters in Table 5.7.

Table 5.7: Attributes of Particles in second layer

Particles	**Parameter P_3 (or position of corresponding particles)**	**Selected first P_2 i.e. 2 nodes to make a polynomial**	***Ek = MSE btw $f(x_i, x_j)$ & actual output***
Particle 1	1, 3, 4, 5, 2	1, 3	3.5
Particle 2	**2, 4, 3, 5, 1**	**2, 4**	**5.9**
Particle 3	2, 3, 1, 5, 4	2, 3	4.4
Particle 4	5, 3, 1, 4, 2	5, 3	5.8
Particle 5	5, 1, 2, 4, 3	5, 1	8.5

All possible combinations of nodes and their corresponding dummy error values in the second layer are shown in Table 5.8. Note that the non-promising node is now promising with order no 6 but it is still not the best solution.

Table 5.8: All combinations of nodes

Order No	x_i	x_j	E_k *val for f*(x_i, x_j)
1	1	2	5.8
2	1	3	3.5
3	1	4	8.6
4	1	5	8.5
5	2	3	4.4
6	**2**	**4**	**5.9**
7*	2	5	3.5
8	3	4	11.0
9	3	5	5.8
10	4	5	4.9

*best value at this stage

All new combinations of nodes provided by P_3 parameter will again be selected for next layer. Table 5.9 shows the selected combinations with grey background color. Table 5.10 shows the 5 selected nodes to be processed for next layer.

Table 5.9: PSO-GMDH Approach

Order No	x_i	x_j	E_k *val for f*(x_i, x_j) in sorted order
2*	1	3	3.5
7	2	5	3.6
5	2	3	4.4
10	4	5	4.9
1	1	2	5.8
9	3	5	5.8
6	**2**	**4**	**5.9**
4	1	5	8.5
3	1	4	8.6
8	3	4	11

*best value at this stage

Table 5.10: PSO-GMDH Approach: Chosen candidates for next iteration

Prev Order No	*New Order No*	x_i *or* x_j
2*	1	3.5
5	2	4.4
10	3	4.9
6	**4**	**5.9**
4	5	8.5

*best value at this stage

Layer 3 of the process:

The third layer will pick the nodes from second layer which is shown in Table 5.10. The new attributes for the particles are shown in Table 5.11. The non-promising node has now become the most promising node captured by particle 1 and is shown with bold in Table 5.11.

Table 5.11: Attributes of Particles in third layer

Particles	Parameter P_3 (or position of corresponding particles)	Selected first P_2 i.e. 2 nodes to make a polynomial	*Ek = MSE btw $f(x_i, x_j)$ & actual output*
Particle 1	**1, 4, 3, 5, 2**	**1, 4**	**3.4**
Particle 2	2, 4, 5, 3, 1	2, 4	6.9
Particle 3	2, 1, 3, 5, 4	2, 3	4.4
Particle 4	4, 3, 1, 5, 2	5, 3	5.7
Particle 5	5, 1, 3, 4, 2	5, 1	8.3

All possible combinations of nodes from layer 2 are now shown in Table 5.12 and selected selected nodes for next layer is shown in Table 5.13. Note very carefully that the best polynomial is of order no 3 which is constructed from that same node which was very non promising at layer 1. Traditional GMDH would have rejected that node at initial stage but PSO has used that same node which was initially non promising but at the layer 3 it has become the most promising and the best node. This node is made from combination of node 1 and 4 of layer 2 where node 4 of layer 2 was made from nodes 2 and 4 of layer 1 that was initially made from input data 1 and 2. Node transformation from non-promising to very promising is shown in Fig. 5.9.

Table 5.12: All combinations of nodes

Order No	x_i	x_j	E_k *val for* $f(x_i, x_j)$
1	1	2	5.8
2	1	3	3.5
3	**1**	**4**	**3.4**
4	1	5	8.3
5	2	3	4.4
6	2	4	6.9
7*	2	5	3.5
8	3	4	11.0
9	3	5	5.7
10	4	5	4.9

Table 5.13: PSO-GMDH Approach: Chosen candidates for next iteration

Prev Order No	*New Order No*	x_i *or* x_j
3*	**1**	**3.4**
1	2	5.8
6	3	6.9
4	4	8.3
8	5	11.0

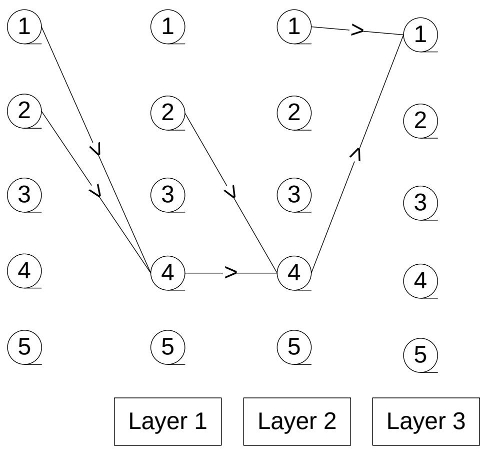


Fig. 5.9: PSO-GMDH selection process of non-promising node 4 of layer 1 which combines with other nodes and becomes promising node in layer 3.

This search for better node will continue until the predefined maximum number of iteration is reached.

5.4.2.4 Coefficient estimation of polynomial corresponding to the selected particle

The estimation of coefficient of each particle is done exactly as traditional GMDH using regression of matrix multiplication [23]. Aim is to find the best possible model such that square of difference between the actual output and the predicted output is minimized.

$$E_j = \sum_{i=1}^{n_{tr}} (y_i - z_{ij})^2 \quad j = 1,2,\ldots P_T \tag{5.19}$$

where z_{ij} denotes the output of the *j-th* particle with respect to the *i-th* data. P_T is the total number of particles in the swarm and n_{tr} is the size of training set. To use the matrix regression first the polynomial function has to be converted to matrix form. For example polynomial of order 2 can be written in the following form:

$$Y = A + Bu + Cv + Du^2 + Ev^2 + Fuv$$

Note that only training dataset is considered for coefficient estimation.

The above function can be transformed into the following matrix form:

$$[A + Bu + Cv + Du^2 + Ev^2 + Fuv] = [Y] \tag{5.20}$$

$$\text{or,}\begin{bmatrix}1 & u & v & u^2 & v^2 & uv\end{bmatrix}_{[nt\times 6]}\begin{bmatrix}A\\B\\C\\D\\E\\F\end{bmatrix}_{[6\times 1]} = \begin{bmatrix}Y\end{bmatrix} \tag{5.21}$$

$$\text{or simply, } [X]\ [\text{coeffs}] = [Y] \tag{5.22}$$

where [coeffs] represents [A B C D E F]^T, vector of coefficients.

The least square technique from multiple-regression analysis provides the formula to get the coefficients in the following form [23]:

$$[\text{coeffs}] = (X^TX)^{-1}X^TY \tag{5.23}$$

5.4.2.5 Selection of input variables through swarm particles for next layer

At each iteration of PSO, swarm particles use their corresponding parameter P_3 which utilizes the position (sequence of integers) of particles, to determine which input variables are to be combined to make a polynomial. These polynomials will become candidates for the selection of nodes for next layer. The selection is based on fitness function of the model which is simply the mean square error:

$$E_r = \frac{1}{n_{te}}\sum_{i=1}^{n_{te}}\left(y_o - y_c\right) \tag{5.24}$$

The lower the error the better the fitness. As shown in the Eq. 5.24 mean square error is calculated using testing dataset only. This provides the independent testing of model function.

Total particles P_T (normally $P_T > M$) in each iteration are supposed to provide the selection guidelines using steps 1, 2 3, and 4 for M nodes of corresponding layer of GMDH process to move to next layer until termination condition is met. PSO will keep P_T particles and GMDH will keep M nodes.

5.4.2.6 Termination criterion:

PSO-GMDH uses PSO's termination method which is predetermined by the user by specifying number of iterations required for the process. PSO also has auto termination method which analyses the system and determines if no better solution is possible then the execution is terminated.

5.5. Experimentation

Few experiments are conducted to determine the feasibility and efficiency of this novel algorithm. The following dataset were used.

5.5.1 Tool wear problem

5.5.1.1 Experimental setup

The problem discussed here is the end-milling experiment which was carried out on the Seiki Hitachi milling machine and also reported in chapter 4. A 16mm Co-high speed (HSS) Kobelco Hi Cut brand new end mill cutter was used to machine the work-piece. The end mill cutter had four flutes and the flute geometry was 30 degrees spiral. The overall length of the cutter was 77mm and the flute length was 26.5 mm. The work-piece machined was mid steel blocks which had a constant length of 100 mm for each trial. The machining was done under dry conditions. The milling experiment was conducted as designed. The work-piece used was mild steel which had a Brinell hardness number of 128.

Monitoring of the tool wear of the end mill was conducted in the toolmakers microscope. The 16mm Co-high speed (HSS) Kobelco Hi Cut brand new end mill cutter having 4 teeth was measured in the toolmakers microscope. A tool holder was designed so that it could hold the tool on the toolmakers microscope table and readings could be taken.

Tool wear for turning was monitored in the toolmakers microscope. The 55 degrees carbide insert with a positive rake angle of 7 degrees was removed from the tool holder and measured in the toolmakers microscope. A reference had to be made such that the distance from the reference line to the tip of the insert could be taken. It was very difficult to make a permanent reference line on the insert thus an insert holder was prepared. The insert holder was designed in such a way that the insert fitted inside the hole perfectly. The height of the surface of the insert was equal to height of the insert holder. Two reference lines were made on the insert holder, one at right angles to the tip of the insert which took into account the wear taking place on the nose of the insert and the other reference line was made parallel to the side of the insert which took into account the flank wear of the insert. A brand new end mill tool was measured on the toolmakers microscope from the reference line to the cutting edge. The tool was then used to machine a block of mild steel on the conditions of trial number 1. After machining the end mill was removed from the milling machine and the amount of wear on the end mill cutter was measured. The difference of the average of the first reading and the average of the current reading gave the extent of tool wear. Four readings were taken from each reference line and average of the readings was done.

5.5.1.2 Design of experiment

The machining parameters were set during experimentation. This data set constituted the input to the self-organizing network and consisted of three inputs and one output. All inputs and were considered candidates to the causality relationship. For the specific application it was found that five replications were sufficient to yield a good approximation. The range of speed, feed, and depth of cut chosen for the experiments are respectively $v \in \{27, 39, 49\}$, $f \in \{0.0188, 0.0590, 0.1684\}$, $d_t \in \{0.5, 1.0, 1.5\}$.

5.5.1.3 Experimental Results and Discussions

All the twenty seven trials were conducted using the same end mill cutter and each time after milling the measurements for wear was taken. The average of the present measurement was subtracted from the previous one and the difference in the measurements gave the amount of wear. The cutting conditions and results obtained for the twenty seven trials using Mild steel blocks as work piece is shown in Table 1.

The turning parameters (see Table 5.13) that were fed into the enhanced-GMDH network shown in Figure 1 as inputs are $x_1 =$ speed (v); $x_2 =$ feed, (f), and $x_3 =$ depth-of-cut (DOC, d_t). The targets for the tool wear are given in the last columns of Table 1. The outputs of the e-GMDH learning network reported in this paper were used to develop the mathematical model of the tool wear in Sections 4.

Table 5.13 Cutting conditions and measured values for the milling operation

Trial #	Speed, v (m/min)	Feed, f (mm/rev)	DOC, d_t (mm)	Wear, VB (μm)
1	27	0.0188	0.5	8.0
2	27	0.0188	1	2.3
3	27	0.0188	1.5	2.6
4	27	0.059	0.5	1.7
5	27	0.059	1	2.45
6	27	0.059	1.5	2.7
7	27	0.1684	0.5	1.95
8	27	0.1684	1	2.55
9	27	0.1684	1.5	2.85
10	36	0.0188	0.5	2.9
11	36	0.0188	1	3.35
12	36	0.0188	1.5	4.35
13	36	0.059	0.5	3.105
14	36	0.059	1	3.55
15	36	0.059	1.5	4.5
16	36	0.1684	0.5	3.196
17	36	0.1684	1	3.95
18	36	0.1684	1.5	4.65
19	49	0.0188	0.5	4.95
20	49	0.0188	1	5.85
21	49	0.0188	1.5	7.7
22	49	0.059	0.5	5.2
23	49	0.059	1	6.25
24	49	0.059	1.5	10.2
25	49	0.1684	0.5	5.45
26	49	0.1684	1	6.75
27	49	0.1684	1.5	19.52

Fig. 5.10 shows the measured and estimated outputs for the tool wear problem. Figure 5.11 shows the GMDH network for the tool wear problem. Table 5.15 shows the database of output results for the tool wear problem.

The interpretation of the network for each of the four layers is as follows:

$$y_1 = f(x_1, x_3);\ y_2 = f(x_1, x_3);\ y_3 = f(x_2, x_3)$$
$$y_4 = f(y_1, y_2);\ y_5 = f(y_2, y_3);\ y_6 = f(y_2, y_3)$$
$$y_7 = f(y_4, y_5);\ y_8 = f(y_5, y_6)$$

$$y_9 = f(y_7, y_8)$$

However, these have to be corrected mapped to the database results shown in Table 5.15. For the database results, layer 1 outputs are first given followed by those for layer 2, and so on until the last layer is reached. In layer 1, y_1, y_2 and y_3 of the network correspond to y_3, y_2 and y_1 of the database respectively. In layer 2, y_4, y_5 and y_6 of the network correspond to z_1, z_3 and z_2 of the database respectively. In layer 3, y_7 and y_8 of the network correspond to w_1 and w_2 of the database respectively. In layer 4, y_9 of the network correspond to v_1 of the database respectively.

The coefficients per node are also shown in Table 5.15. For example the coefficients of node y_3 in layer 1 are {3.53183, 3.41982, -0.43112, 0.091457, 2.60811, and 0.009742}. It therefore becomes easy to determine the connections per layer until the final output as well as the equations connecting the nodes. The polynomial for a node is defined using these coefficients as:

$$f(x_i, x_j) = c_1 + c_2 x_i + c_3 x_j + c_4 x_i x_j + c_5 x_i^2 + c_6 x_j^2$$

As could be observed, very nonlinear set of equations can occur and certainly the degree of the polynomials increases by the power of 2 from layer to layer (l^2) which means that for layer 2, the degree of polynomial is 4, etc.

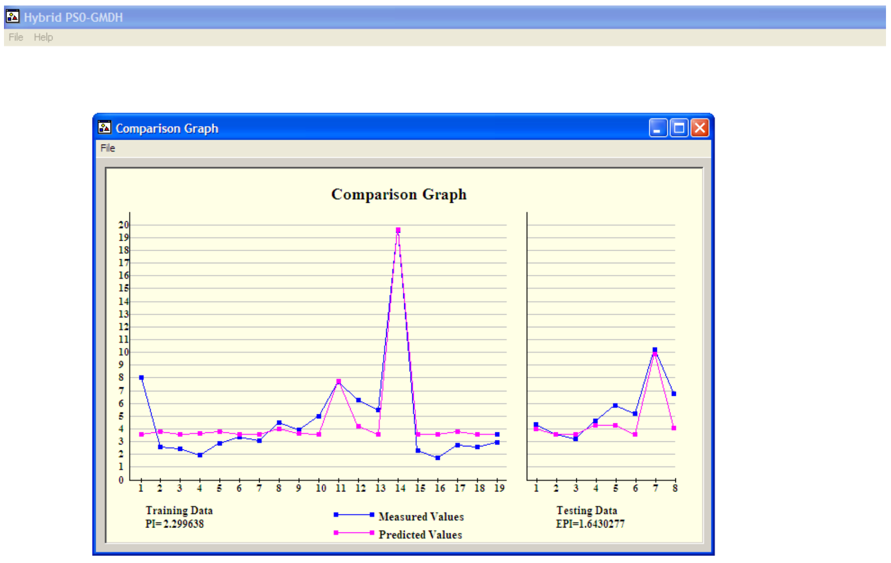


Fig. 5.10 Measured and estimated outputs for the tool wear problem

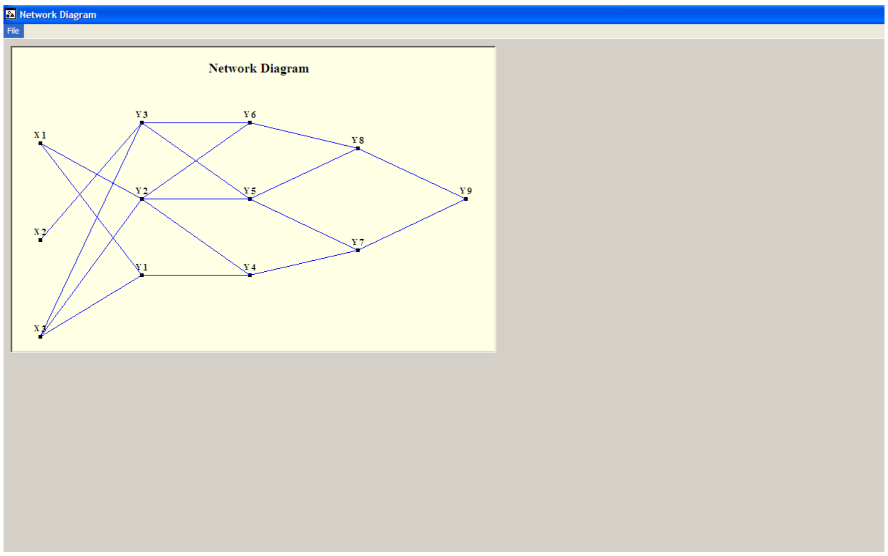


Fig. 5.11 GMDH network for the tool wear problem

Table 5.14 Database of output results for the tool wear problem

PARTICLE SWARM OPTIMIZATION								
============================								
Processing time: 1.891 seconds								
Iterations: 4								
Evaluations: 84								
Optimum value :				1.64303				
Best Layer at:				3				
EPI:				1.64303				
PI:				2.29964				
Network of Layers								
3	3	1	3.53183	3.41982	-0.43112	0.091457	2.60811	0.009742
2	3	1	3.53183	3.41982	-0.43112	0.091457	2.60811	0.009742
1	3	2	3.79263	-2.15968	0.978243	1.54233	2.59583	0.210299

3	3	2	3.69111	-2.34E-06	-0.39864	0.033452	0.000891	9.06E-11
2	2	1	3.811	-0.36014	0.274229	-0.07148	0.049272	-6.18E-05
1	2	1	3.811	-0.36014	0.274229	-0.07148	0.049272	-6.18E-05
2	3	2	4.00562	-7.1202	6.50868	-1.27664	1.36893	1.30E-06
1	1	2	4.00176	-6.68E-06	-0.38546	0.083097	-4.28E-05	-3.64E-10
1	2	1	3.93398	-0.66593	0.30708	-0.02418	0.094024	1.66E-05

---------------- Training Data Set ----------------------		
8	3.59999	
2.6	3.74768	
2.45	3.5648	
1.95	3.61073	
2.85	3.78751	
3.35	3.58753	
3.105	3.5448	
4.5	4.02922	
3.95	3.6568	
4.95	3.56642	
7.7	7.71615	
6.25	4.19115	
5.45	3.55488	
19.52	19.5956	
2.3	3.55567	
1.7	3.60276	
2.7	3.7572	
2.55	3.59692	
2.9	3.54588	
---------------- Testing/Checking Data Set --------------		
4.35	3.96518	

3.55	3.60264	
3.196	3.54198	
4.65	4.25716	
5.85	4.25657	
5.2	3.56175	
10.2	9.84692	
6.75	4.0349	

5.5.2 Gas Furnace Problem

The problem solved is the "Box and Jenkins furnace data" [28] which is a benchmark problem often reported in the literature. There are originally 296 data points {y(t),u(t)}, from t=1 to t=296. In this problem, y(t) is the output CO2 concentration and u(t) is the input gas flow rate. Here we are trying to predict y(t) based on

{y(t-1), y(t-2), y(t-3), y(t-4), u(t-1), u(t-2), u(t-3), u(t-4), u(t-5), u(t-6)}.

This reduces the number of effective data points to 290. Most methods find that the best set of input variables for predicting y(t) is {y(t-1),u(t-4)}. Sugeno and Yasukawa has found that the best set of input variables for predicting y(t) is {y(t-1), u(t-4), u(t-3)}.

The first column below is the output variable y(t), the remaining columns are the input variables {y(t-1), y(t-2),...u(t-6)}. Consequently, we have the inputs and output as:

output y(t);
input y(t-1) y(t-2) y(t-3) y(t-4) u(t-1) u(t-2) u(t-3) u(t-4) u(t-5) u(t-6)

5.5.2.1 PSO-GMDH modeling

The parameters used for the modeling of this problem are:

Swarm size	0
Maximum best column	0
Even row selection	1
Maximum tour	9000000
Intensity ‘n’ = no; ‘y’ = yes	n
Maximum evaluation	4
Convergence case	5

The number of swarm size is given by the user but when PSO-GMDH is required to determine this automatically the value of ‘0’ is given. This is the case for the maximum best column also. For the selection for the test data, three cases exist: random (0), even (1), and odd (2). When random selection is made, PSO-GMDH randomly selects data for testing but when 1 or 2 is selection even or odd rows of data are selected for testing. When a brute-force search is required, then intensity is ‘yes’ or ‘y’ otherwise, the choice is ‘no’ or ‘n’. the maximum evaluation is set by the user or left to PSO-GMDH to determine if ‘0’ is chosen. There are five convergences cases to choose from.

The results generated by the PSO-GMDH after each iteration are reported as:

```
After iteration 1 the best particle is particle [ 1 ]
 position: 7 2 1 9 6 5 3 8 4 10 7
 velocity: (size = 1) : magnitude [ 1 , 7 ]
 objective functional value: 0.0801373
 distance: 0
After iteration 2 the best particle is particle [ 7 ]
 position: 1 6 3 4 2 5 7 8 9 10 1
 velocity: (size = 6) : magnitude [ 9 , 5 ] [ 5 , 4 ] [ 2 , 6 ] [ 4 , 9 ] [ 4 , 2 ] [ 9 , 5 ]
 objective functional value: 0.0714395
 distance: 0
After iteration 3 the best particle is particle [ 7 ]
 position: 1 6 3 4 2 9 7 8 5 10 1
 velocity: (size = 6) : magnitude [ 9 , 5 ] [ 5 , 4 ] [ 2 , 6 ] [ 4 , 9 ] [ 4 , 2 ] [ 9 , 5 ]
 objective functional value: 0.0687775
 distance: 0
After iteration 4 the best particle is particle [ 7 ]
 position: 1 6 3 4 2 9 7 8 5 10 1
 velocity: (size = 6) : magnitude [ 9 , 5 ] [ 5 , 4 ] [ 2 , 6 ] [ 4 , 9 ] [ 4 , 2 ] [ 9 , 5 ]
 objective functional value: 0.0687775
```

Figure 5.12 shows the measured and estimated outputs for the gas furnace problem. The training error of 0.126 and the testing error of 0.068 are displayed. These errors are base on mean square error (MSE). The testing error is often used in the literature to determine the efficacy of a modeling approach; this gives more information than the graph.

Figure 5.13 shows the GMDH network for the gas furnace problem. There are 10 inputs and one output, but we have only three inputs shown in the network. The explanation is simple. Only parameters 2, 3 and 10 are connected to the three best nodes in layer 1. This means that the objective functions of pair-wise combinations of parameters 1, 4, 5, 6, 7, 8, and 9 do not give competitive nodes, so those nodes are relegated to the background; in other words they are dropped.

Table 5.15 shows the database of output results for the gas furnace problem. This database is rich in information because it gives the network topology and the equations for each node. A bit of interpretation of the output data is necessary. Whereas the network shows that y_1 is connected to x_2 and x_3, y_2 is connected x_2 and x_{10} while y_3 is connected x_2 and x_{10} while the database gives the true interpretation that y_7 is connected to x_2 and x_3, y_2 is connected x_2 and x_{10} while y_1 is connected x_2 and x_{10}. The differences are merely due to the ways of numbering on the network and internally in the database. The database gives the exact connections but the network labels are sequential for the active neurons. Non-active neurons are not numbered on the network but internally the numbering is maintained for the database archiving.

Table 5.15 Database of output results for the gas furnace problem

PARTICLE SWARM OPTIMIZATION								
============================								
Optimum value :			0.068778					
Best Layer at :			3					
EPI :			0.068778					
PI :			0.126611					
Network of Layers								
7	2	3	-0.99279	0.315822	0.365218	-0.00781	0.001328	6.38E-11
2	10	2	-0.99713	-0.0828	0.018472	0.018933	0.055631	1.32E-06
1	10	2	-0.99713	-0.0828	0.018472	0.018933	0.055631	1.32E-06
1	7	2	-0.9924	-1.67656	0.083294	0.063762	0.103198	0.002982
6	1	2	-2.6647	-3.86E-06	1.13451	1.03578	1.89E-05	-3.96E-08
2	7	2	-0.9924	-1.67656	0.083294	0.063762	0.103198	0.002982
1	1	6	-1.00367	1.35655	-0.34384	0.048684	0.282728	-0.00344
6	1	2	-2.4166	8.10E-06	1.26712	0.730777	2.12E-05	-5.85E-08
1	1	6	-1.00337	1.51282	-0.43421	-0.04207	0.375178	0.000249
Processing time: 8.188 seconds								
Iterations: 4								
Evaluations: 872								

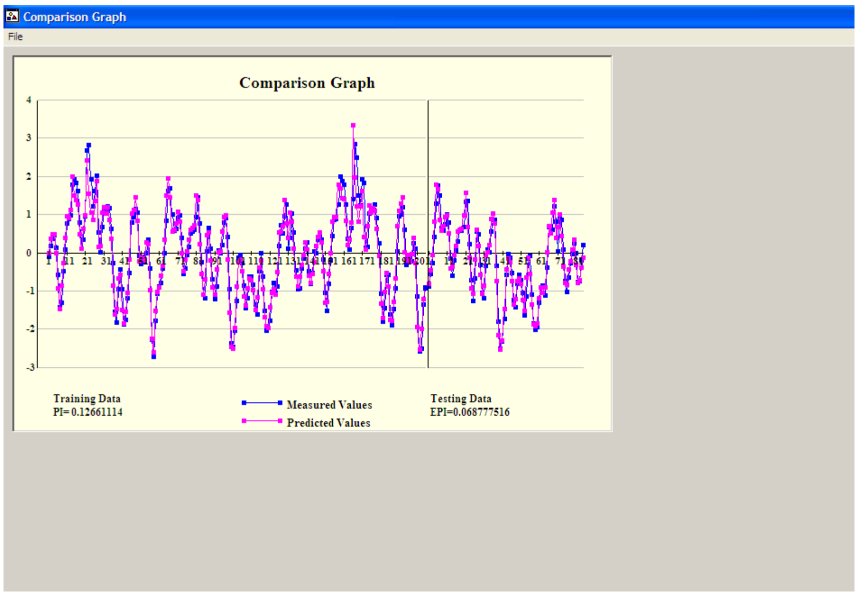


Fig. 5.12 Measured and estimated outputs for the gas furnace problem

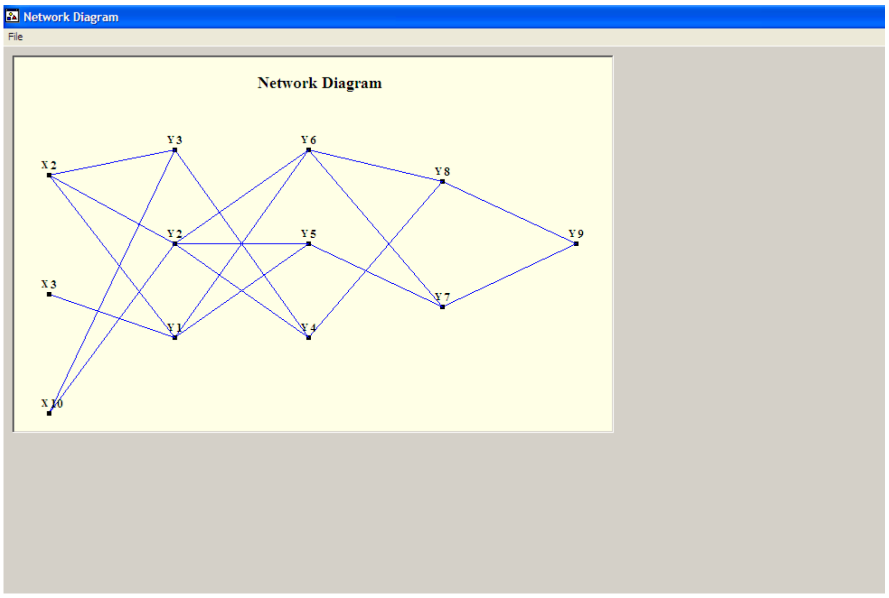


Fig. 5.13 GMDH network for the gas furnace problem

5.6. Conclusion

This chapter has presented a hybrid PSO-GMDH for modeling and prediction of complex, real-life problems. The inherent setbacks found in the classical GMDH are resolved when a hybrid of PSO and GMDH is realized. The hybrid PSO-GMDH architecture realized is more flexible in determining network topology of the problem being solved. The hybrid PSO-GMDH learns much more easily and generalizes much better than the classical GMDH.

In this chapter, we have presented for the first time a hybrid PSO-GMDH framework because we are not aware of any literature that has reported this. Two types of problems have been solved, one is based on experimentation in a manufacturing laboratory in which the controlling parameters were used to generate output response in the form of tool wear; the other is the time series problem. Although the results obtained are not as competitive as those obtained in chapter 4 for tool wear problem, they are quite promising.

The man-machine interface for the hybrid PSO-GMDH software developed and reported in this chapter is very conducive for utilizing as a data mining platform.

Acknowledgments

Amal Shankar, Ashwin Dayal, Deepak Bhartu, and Kenneth Katafono were responsible for integrating the various PSO and GMDH components and writing the GUI codes under the supervision of the two authors of this chapter. The authors of this chapter provided the full PSO and partial GMDH codes.

Reference

[1] Farlow, S. J. (1981). The GMDH Algorithm of Ivakhnenko, *The American Statistician,* Vol. 35, No. 4.

[2] Hamming, R. W. & Feigenbaum E. A. (1971). Interpolation and Roundoff Estimation, *Introduction to Applied Numerical Analysis*, MGraw-Hill (USA), pp. 143-148.

[3] Zaychenko, Y. P., Kebkal, A. G. and Krachkovckii, V. F. (2007). The Fuzzy Group Method of Data Handling and its Application to the Problems of the Macroeconomic Indexes Forecasting, Available at http://www.gmdh.net/.

[4] Neapolitan, R. E. and Naimipour, K. (2003). The greedy approach, Fondation of Algorithms using C++ Pseudocode, Jones and Bartlet publishers Inc.

[5] Onwubolu, G. C. (2007). Design of Hybrid Differential Evolution and Group Method of Data Handling for Inductive Modeling, *International Workshop on Inductive Modeling*, IWIM Prague, Czech, 23-26.

[6] Kreyszig, E. (1993). Unconstrained optimization, linear programming, *Advance Engineering Mathematics*. (2nd Ed), John Willey, INC.

[7] Eberhart, R. C. & Kennedy, J. (1995). A new optimizer using particle swarm theory, *in Proc. Sixth International Symposium on Micro Machine and human science*, Nagoya, Japan, IEEE Service Center, Piscataway

[8] Clerc, M. (2004). Discrete particle swarm optimization illustrated by the traveling salesman problem, *New Optimization techniques in Engineering*. Springer-Verlag, Berlin, Germany.

[9] Carlistle, A. & Dozier, G. (1998). Adapting Particle Swarm Optimization to Dynamic Environments, Available at http://www.CartistleA.edu.

[10] Kennedy, J. & Eberhart, R. C (1999). The particle swarm: social adaptation in information processing systems, In Corne, D., Dorigo, M., and Glover, F., Eds., *New Ideas in Optimization*. London: McGraw-Hill, pp. 379-387.

[11] Kennedy, J. & Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm, *International Conference on Systems, Man, and Cybernetics*.

[12] Kennedy, J. (1997). The particle swarm: social adaptation of knowledge, *IEEE international conference on Evolutionary computation*, indianpolis, Indiana, IEEE Service Center, Piscataway, NJ.

[13] Clerc, M. & Kennedy, J. (2002). The particle swarm: explosion, stability, and convergence in a multidimensional complex space, *IEEE transactions on Evolutionary Computation*, Vol. 6, 58-73.

[14] Kennedy, J. (1999). Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance, *Congress on Evolutionary computation*, Washington D.C, IEEE.

[15] Clerc, M. (1999). The Swarm and the queen: towards a deterministic and adaptive particle swarm optimization, *Congress on Evolutionary Computation*, Washington D.C., IEEE, Service Center, Piscataway, NJ, 1951-1957.

[16] Kennedy, J. (2000). Stereotyping: Improving Particle Swarm Performance with Cluster Analysis, *presented at Congress on Evolutionary Computation*.

[17] Kennedy, J. & Spears, W. (1998). Matching algorithms to problems: An experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator, *In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998)*, Anchorage, Alaska.

[18] Onwubolu, G. C. & Sharma, A. (2004). Particle Swarm Optimization for the assignment of facilities to locations, *New Optimization Techniques in Engineering*, Springer-Verlag.

[19] He, Z. & Wei, C. (1999). A new population-based incremental learning method for the traveling salesman problem, *Congress on Evolutionary Computation*, Washington D.C., IEEE.

[20] Press, W. H., Teukolsky, S.A., Vetterling, W. T. and Flannery, B. P. (1992). Numerical recipes in C: The art of scientific computing, Cambridge University Press.

[21] Nariman-Zadeh, N., Darvizeh, A. and Ahmad-Zadeh, G. R. (2003). Hybrid genetic design of GMDH-type neural networks using singular value decomposition for modeling and predicting of the explosive cutting process, Nariman-Zadeh, Proc. Instn Mech. Engrs Vol 217 Part B: Nariman-Zadeh, 779-790.

[22] Ivakhnenko A. G. (1968). The Group Method of Data Handling-A rival of the Method of Stochastic Approximation. *Soviet Automatic Control, vol 13 c/c of avtomatika*, 1,3, pp. 43-55.

[23] Larson, R., Edward, B. H. and Falvo, D. C. (2004). "Application of Matrix Operations," Elementary Linear Algebra (5th ed.). Houghton Mifflin (New York, USA), pp. 107-110.

[24] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

[25] Glover, F. (1977). Heuristics for interger programming using surrogate constraints, Decision Sciences 8:156-166.

[26] Kirkpatrick, S., Gelatt, C. D. and Vecci, M. P. (1983). Optimization by Simulated Annealing, Science, Vol. 220, No. 4598, pp. 671-680.

[27] Dorigo, M. (1992). Optimization, Learning and Natural Algorithm, PhD thesis, Politecnico di Milano, Italy.

[28] Box, G .E. P., and Jenkins, G. M. (1970). Time Series Analysis, Forecasting and Control San Francisco, Holden Day, pp. 532-533.