Scientific Research and Essays Vol. 7(26), pp. 2310-2333, 12 July, 2012
Available online at http://www.academicjournals.org/SRE
DOI: 10.5897/SRE11.2148
ISSN 1992-2248 © 2012 Academic Journals

Full Length Research Paper

Use of neuro fuzzy network with hybrid intelligent optimization techniques for weight determination in parallel Job scheduling

S. V. Sudha[1]* and K. Thanushkodi[2]

[1]Department of Information Technology, Kalaignar Karunanidhi Institute of Technology Anna University of Technology, Coimbatore -641 402, India.
[2]Akshaya College of Engineering, Anna University of Technology, Coimbatore, India.

Accepted 11 April, 2012

This paper is concerned with the performance tuning of the fuzzy logic controller (FLC) used for the process grain sized scheduling of parallel jobs. First, we have proposed a scheduling algorithm called agile algorithm. The performance of the agile algorithm depends on how the processes of the applications are coscheduled. The performance of the scheduling algorithm is evaluated using the features of the scheduling metrics like average waiting time, mean response time, mean reaction time, mean slowdown, turn around time and mean utilization. A rule based scheduling system is framed using the fuzzy logic controller with the help of the above mentioned metrics to schedule all the parallel workload data. The fuzzy controller helps to identify the scheduling strategy using the rule base developed and assign the corresponding scheduling class to the parallel jobs. The fuzzy logic controller uses Mamdani model for classifying the scheduling class and found that the error rate is high during the defuzzification and need to tune the fuzzy controller to reduce the error. The paper concentrates about the tuning of the fuzzy logic controller using a neural network where the rule base of a fuzzy system is interpreted as a neural network. The performance of the neural network in turn depends on the weight determination of its own network and the various optimization techniques like genetic and parallel genetic algorithm, particle swarm optimization, hybrid particle swarm optimization with the tabu search and the parallel implementation of the hybrid approach of the particle swarm optimization with the tabu search are employed to identify the weight determination of the neuro fuzzy network. The paper gives very good performance of the fuzzy controller using the neuro fuzzy system with weights identified using the above optimization techniques. The paper gives a complete analysis of the computational time occurred and the weight identification with the various optimization techniques which guides the neural network to speed up the training process.

Key words: Genetic algorithm, agile algorithm, mean reaction time, particle swarm optimization, tabu search.

INTRODUCTION

The scheduling of parallel jobs has long been an active area of research. Scheduling parallel jobs for execution needs a certain number of processors for a certain time and the schedule have to pack the jobs together. In job scheduling, a parallel job is mapped to a subset of processors. The set of processors dedicated to a certain job is called a partition of the machine. To increase utilization, parallel machines are typically partitioned in to several non overlapping partitions allocated to different jobs running concurrently. Parallel job scheduling is the problem of how to run a workload of multiple parallel jobs in a single parallel machine. The scheduler is responsible for finding the best schedule allocation, both temporal

*Corresponding author. E-mail: svsudha.mvenki@gmailcom or svsudha@rediffmail.com.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
TS_1	$J1_1$	$J1_2$	$J1_3$	$J1_4$						
TS_2	$J2_1$	$J2_2$	$J2_3$							
TS_3	$J3_1$	$J3_2$	$J3_3$	$J3_4$	$J3_5$					
TS_4	$J4_1$	$J4_2$								
TS_5	$J5_1$	$J5_2$	$J5_3$							
TS_6	$J6_1$	$J6_2$	$J6_3$	$J6_4$	$J6_5$					
TS_7	$J7_1$	$J7_2$								
TS_8	$J8_1$									
TS_9	$J9_1$	$J9_2$	$J9_3$	$J9_4$	$J9_5$					
TS_{10}	$J10_1$	$J10_2$	$J10_3$	$J10_4$	$J10_5$					

Figure 1. A scheduling matrix.

and spatial as a function of the existing workload. A good way of improving the performance evaluation of a parallel system is to consider the frequency of synchronization between the processes in a system. The parallel system application consists of multiple processes running on the different processors that communicate frequently. The good performance of the parallel systems mainly depends on how the processes are co scheduled. If the processes are not co scheduled properly, then the system will lead to very poor performance. The various co scheduling techniques are first come first served scheduling algorithm, gang scheduling, Flexible co scheduling. In the first come first served scheduling, when a job arrives, each of its thread is placed consecutively at the end of the shared queue. When a processor becomes idle, it picks the next ready thread, executes it until it completes or blocks. A set of related threads is scheduled to run on a set of processors at the same time on a one to one basis. The concept of scheduling a set of processes simultaneously on a set of processors uses the threads, which is also called as group scheduling or gang scheduling. Flexible co scheduling is used to improve overall system performance in the presence of heterogeneous hardware or software by using dynamic measurement of applications. First come first served and gang scheduling suffer from internal and external fragmentation. Flexible co scheduling saturates at heavy loads. Scheduling is done by partitioning the machine's processor and running a job on each partition. Due to the synchronization between processes in the application, the jobs do not pack perfectly. If the processes are not co scheduled properly, it will harm the performance of the parallel algorithm. Agile algorithm concentrates on the detailed classification of the synchronization granularity of the parallel jobs and the algorithm takes heavy loads up to 10,000 jobs for each granularity and gives better results when compared to the traditional ones. Every job arrival in the ready queue constitutes a scheduling event in the parallel system. For each scheduling event, a new scheduling matrix is computed for the system. The scheduling matrix defines all the tasks executing in each processor and at each time slice. For the implementation of the agile algorithm, a 10 processor system with a multiprogramming level of 10 is considered and it is shown in Figure 1. The matrix is cyclic in that time slice 10 is followed by time slice 1. Scheduling cycle is defined as a cycle which contains an entire row of the matrix values at any instant of time. Each row of the matrix defines a 10 processor virtual machine which runs at 1/10th of the speed of the physical machine. In the scheduling matrix defined in Figure 1. P_1 to P_{10} represent the 10 processor system and TS_1 to TS_{10} represent the time slices and $J1_1$ to $J1_4$ are the partitions of the job J1 and similarly the scheduling matrix represent for the job 2 to job 10 (Dutot et al., 2011; Eitan Frachtenberg et al., 2005).

The paper subsequently explains the scheduling algorithm agile algorithm, after which it talks about the grain sizes and also the workload details. It further gives the complete scheduling metric used for the comparison of the scheduling algorithm, and then states the need for the fuzzy and neuro fuzzy system. Afterwards, the paper explains the rule base system for the job scheduling and the structure of the neuro fuzzy system, and also gives the defuzzification results from the fuzzy controller. This is followed by a presentation of the weight determination and the fitness evaluation, after which the optimization techniques used for the weight determination is presented. Finally the paper discusses the results and the complete analysis of the fitness results and computational results from genetic algorithm and its parallel implementation, particle swarm optimization, and the hybrid approach of the particle swarm optimization with the tabu search, after which its conclusions are drawn.

Agile algorithm –Scheduling technique for parallel job scheduling

The algorithm concentrates on detailed classification of the frequency of synchronization between processes in a system. For the implementation of the agile algorithm four workloads are considered. Each workload may be fine grain, coarse grain, medium grain and independent which contain 10,000 jobs. Each 10,000 jobs are divided in to 1000 slots and each slot contains 10 jobs. The jobs in the slots are scheduled in the scheduling matrix as shown in Figure 1.

The following explains the grain explanation used in the workload (Dan et al., 2007).

Fine grain

Fine grained parallelism represents a much more complex use of parallelism. The processes communicate often and must be co scheduled effectively due to their demanding synchronization.

Medium grain

Medium grain parallelism represents enough synchronization between the processes and the scheduling algorithms should take care of the performance evaluation of the system.

Coarse grain

With coarse Grain, there is synchronization among processes, but at a very gross level. This kind of situation is easily handles as a set of concurrent processes running on a multiprogrammed uniprocessor and can be supported on a multiprocessor with little or no change to the software.

Independent

With independent parallelism, there is no explicit synchronization among processes. Each represents a separate, independent application or job.

Workload characteristics

The simulation studies were performed using the agile algorithm with workload logs available from Feitelson’s Archive (Thanushkodi and Sudha, 2009).

Fine grain workload

The log considered for the experiment was The Los Alamos National Lab (LANL) Log. This log contains two years worth of accounting records produced by the DJM software running on the 1024 node CM-5 at Los Alamos. Total Number of jobs present in the Log is 2, 01,387 Jobs. The Log contains the following details: 1) Job Id 2) Submit Time 3) Start Time and Date 4) End Date and Time.

Medium grain workload

The log considered for the experiment was LLNL Thunder Log. This log contains several months’ worth of accounting records from a large Linux cluster called thunder installed at Lawrence Livermore. Total number of jobs present in the Log is 1, 28,662 Jobs. The Log contains the following details (1) Job ID (2) User ID (3) Name (4) Job State (5) Start Time (6) End Time (Thanushkodi and Sudha, 2009; Minh et al., 2010).

Coarse grain workload

The log considered for the experiment was The Lawrence Livermore National Lab (LLNL) T3D Log. This log contains 4 months worth of accounting record at the Lawrence Livermore National Lab (LLNL). Total number of jobs present in the log is 21323 Jobs. The log contains the following details (1) Start Date (2) Start Time (3) Process ID (4) Partition ID.

Independent

The log considered for the experiment was LPC Log. This log contains 9 months of record. Total number of Jobs present in the Log is 2, 44,821 Jobs. The log contains the

Table 1. Fine grain workload (in seconds).

Algorithm/Metric	FCFS	Gang	FCS	Agile
AWT	2936	2110	2110	2110
MRT	5752	3350	3350	3350
TAT	31660	16810	16810	16810
MRET	4120	3380	3380	3380
MU	0.5	0.6	0.6	0.6
MS	71.10486	49.2	49.2	49.2

following details (1) Job ID (2) Submit time (3) Wait time (4) Run time.

Performance metrics

The synthetic workload generated Feitelson's archive are used as input to the simulation of various scheduling strategies. We monitor the parameters of the arrival time, start time, execution time; finish time etc. Different scheduling algorithms have different properties and may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms. Many criteria have been suggested for scheduling algorithms. The criteria includes the following as presented by Jintao et al. (2010)

Mean utilization

We want to keep the CPU as busy as possible. CPU utilization may range from 0 to 100%. In a real system, it should range from 40% (for a lightly loaded system) to 90%t (for a heavily loaded system). The mean utilization is the ratio of CPU busy time to the number of processors multiplied with total time for execution.

$$\text{Mean utilization} = \frac{\Sigma \text{ CPU busy time}}{\text{Number of processors *Total time}} \quad (1)$$

Mean response time

In an interactive system, turnaround time may not be the best criteria. Often, a process can produce some output fairly early, and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure is called response time.

$$\text{Mean response time} = \frac{\Sigma \text{ Job finish time-Job submit time}}{\text{Number of jobs}} \quad (2)$$

Mean reaction time

The mean job reaction time defined as the mean time interval between the submission and the start of the job.

$$\text{Mean reaction time} = \frac{\Sigma \text{ Job start time-Job submit time}}{\text{Number of Jobs}} \quad (3)$$

Mean slowdown

Mean slowdown is the sum of jobs response times divided by the job's execution times. This metric emerges as a solution to normalize the high variation of the jobs response time.

$$\text{Mean slow down} = \frac{\Sigma \text{ Job response time/ Job execution time}}{\text{Number of jobs}} \quad (4)$$

Turn around time

From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turn around time. Turn around time is the sum of periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O.

Waiting time

The scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.

Tables 1 to 4 represents the scheduling features values (that is) average waiting time (AWT), mean response time (MRT), turn around time (TAT), mean reaction time (MRET), mean utilization (MU) and mean slowdown (MS) during the implementation of the agile algorithm. These

Table 2. Medium grain workload (in seconds).

Algorithm/Metric	FCFS	Gang	FCS	Agile
AWT	45163	11291	9033	6452
MRT	45989	11497	9198	6570
TAT	25894	6138	3600	93
MRET	45163	11291	9033	6452
MU	0.5	0.5	0.6	0.7
MS	1.83	0.46	0.37	0.26

Table 3. Coarse grain workload (in seconds).

Algorithm/Metric	FCFS	Gang	FCS	Agile
AWT	53002	13251	663	221
MRT	53794	13448	672	224
TAT	17110	4278	214	71
MRET	53002	13251	663	221
MU	0.5	0.6	0.7	0.7
MS	3.2	0.8	0.04	0.013

Table 4. Independent grain workload in seconds.

Algorithm/Metric	FCFS	Gang	FCS	Agile
AWT	6	0	0	0
MRT	23	0	0	0
TAT	56590	801	174	16
MRET	6	0	0	0
MU	0.5	0.5	0.6	0.5
MS	3.6912	0.0527	0.0114	0.001036

values are taken for the learning analysis using the neuro fuzzy system in our paper.

For the Log LANL that is for the fine grain application, the overall running time with the first come served algorithm (FCFS) was 2 h, 31 min and 6 s. The overall running time with the Gang scheduling (Gang) was 17 min and 9 s. Flexible co scheduling (FCS) and the agile algorithm give the same figure as Gang scheduling.

For the Log LLNL that is for the medium grain application, the overall running time with the First Come Served Algorithm was 7 h, 11 min and 34 s. The overall running time with the Gang scheduling was 32 min and 27 s. The overall running time for the flexible co scheduling was 16 min and 33 s and for the agile algorithm it was 1 min and 33 s.

For the Log LLNL T3D that is for the coarse grain application, the overall running time with the first come served algorithm was 2 h, 21 min and 7 s. The overall running time with the Gang scheduling was 17 min and 9 s. The overall running time for the flexible co scheduling was 5 min and 13 s and for the agile algorithm it was 35 s.

For the Log LPC Log that is for the independent grain application, the overall running time with the first come served algorithm was 5 h, 56 min and 2 s. The overall running time with the Gang scheduling was 1 min and 10 s. The overall running time for the flexible co scheduling was 40 s and for the agile algorithm it was 16 s.

Need for fuzzy logic controller and neuro fuzzy system

The agile algorithm concentrates mainly on the frequency of synchronization between the processes of the application and the performance of the agile algorithm is compared with the traditional scheduling algorithm like first come first served, gang scheduling and flexible co scheduling with the scheduling metrics like turn around time, average waiting time, mean response time, mean

Table 5. Fuzzy controller output for fine grain workload.

Scheduling parameter / error calculation	Values	Actual	Defuzzification results	Error in %
AWT	2110	18920	19600	3.47
TAT	16810			
MRESP	3350	6730	6650	1.2
MRECT	3380			
MU	0.6	0.6	0.511	17.42
MS	49.2	49.2	51.6	4.65

Table 6. Fuzzy controller output for medium grain workload.

Scheduling parameter/error calculation	Values	Actual	Defuzzification results	Error in %
AWT	6452	6595	13500	51.15
TAT	93			
MRESP	6570	13022	21400	39.15
MRECT	6452			
MU	0.7	0.7	0.522	34.10
MS	0.26	0.26	0.429	39.39

reaction time, mean slowdown and mean reaction time. A rule base is evaluated after the complete scheduling of all the jobs in a workload of fine grain, medium grain, coarse grain and independent grain sized jobs. The applicability of the fuzzy logic controller plays an important role in the scheduling of the parallel system. Furthermore neural network has been used to optimize the fuzzy logic controller's performance. The rule base can be generated using the fuzzy logic controller. For a rule bases scheduling approach, every possible scheduling state must be assigned to a corresponding class that is described using the available scheduling features. A complete rule base RB consists of a set of rules R_i. Each rule R_i contains a conditional and consequence part. The conditional part describes the conditions for the activation of the rule using the defined feature and the consequence part represent the corresponding scheduling strategy. The fuzzy system uses the mamdani model. Each feature is represented by the real values. This value defines a domain in the feature space, where the influence of the rule is very high. In Mamdani model; a Gaussian membership is used to describe the feature description. Fuzzy inference systems exhibit complementary characteristics, offering a very powerful framework for approximate reasoning as it attempts to model the human reasoning process at a cognitive level. Fuzzy system acquire knowledge from domain expects and this is encoded within the algorithm in terms of the set of if-then rules. The fuzzy logic controller use heuristic information to make their rule base and the membership functions. Optimal design of the knowledge base is central to the performance of the fuzzy logic controller and designing a proper knowledge base of a fuzzy logic controller is not an easy task. The controller also depends on the number of variables and that of the linguistic terms used to represent each variable. The paper uses the results of the agile algorithm for the rule based scheduling approach. The fuzzy controller results are shown in Tables 5, 6, 7 and 8. Tables 5, 6, 7 and 8 shows the defuzzification results of the scheduling parameters like AWT (average waiting time) and TAT (Turn around time), MRESP (Mean Response Time) and MRECT (Mean Reaction time), MU (Mean Utilization) and MS (Mean Slowdown). The Mamdani's model takes the input values of the above mentioned parameter combinations in the antecedent part and the consequent part represents the defuzzification values. The error rate is high in the fuzzy system and the performance of the fuzzy system can be tuned using neural network. So the paper concentrates on the performance improvement of the fuzzy logic controller where in the rule base of a fuzzy system is interpreted as a neural network. Neural network offer a highly structured architecture with learning and generalization capabilities. The generalization ability for new inputs is then based on the inherent algebraic structure of the neural network. Neural network solve problems by learning and self organization Fuzzy sets can be regarded as weights whereas the input and the

Table 7. Fuzzy controller for coarse grain workload.

Scheduling parameter / error calculation	Value	Actual	Defuzzification results	Error in %
AWT	221			96.266
TAT	71	292	7820	
MRESP	224			96.261
MRECT	221	445	11900	
MU	0.7	0.7	0.5	40.000
MS	0.013	0.013	0.354	96.328

Table 8. Fuzzy controller for independent workload.

Scheduling parameter / error calculation	Values	Actual	Defuzzification results	Error
AWT	0	16	6110	99.738
TAT	16			
MRESP	0	0	14.5	100.000
MRECT	0			
MU	0.6	0.6	0.511	17.417
MS	0.001036	0.001036	0.108	99.041

output variables and the rules are modeled as neurons. The neuron of the network represents the fuzzy knowledge base. The performance of a fixed architecture neural network is dependent on the weights connecting the neurons of the input and the hidden layer and those if the hidden and the output layers, bias values and the coefficient of the transfer function used. The back propagation algorithm may be utilized to optimize the above parameters in a supervised learning algorithm. A back propagation neural network determines its weight based on the gradient search technique and therefore runs the risk of encountering the local minimum problem. The back propagation network is the most well known and widely used among the currently available neural network system. The learning algorithm behind the back propagation network is a kind of gradient descent technique with backward error propagation. The back propagation network is used the mathematical formula present here can be applied to any network and does not require any special mention of the features of the function to be learn, the computation time is reduced if the weight chosen are small at the beginning, The batch update of weights exist, which provide a smoothing effect on the weight correction terms and these are the main reasons for selecting the back propagation as a learning algorithm in our paper. The paper concentrates on the optimization techniques like genetic algorithm, particle swarm optimization, parallel genetic algorithm and the hybrid Particle swarm optimization with the tabu search to guide the back propagation network in finding the necessary connection weights in order to enhance the speed of the neural training in turn improves the performance of the fuzzy system (Ghedjati et al., 2010; Avi Nissimov and Dror, 2007).

Scheduling strategy based on rule base system

A complete rule base is developed with the set of rules and each rule contains a conditional and a consequent part. The conditional part specifies the activation of the rule using the scheduling features defined in the agile algorithm. The consequence part represents the scheduling class .In order to specify all scheduling classes the scheduling features are partitioned in to some different ranges, which help us to define easily the scheduling class. The conditional part which combines the scheduling features are as follows: (1) Average waiting time and the turn around time (2) Mean response time and Mean reaction time (3) Mean slowdown and Mean Utilization. The various features will take linguistic terms are VS (very small), S (small), M (medium), MH (medium high), H (high) and EH (extreme high) and the rule are formed only with the help of the scheduling

Table 9. Rule table for average waiting time and turn around time (Fine grain).

AWT/TAT	VS	S	M	MH	H	EH
VS	D	A	A	A	A	A
S	A	A	A	A	A	A
M	A	A	A	A	A	A
MH	A	A	A	A	A	A
H	A	A	A	A	A	A
EH	A	A	A	A	A	A

Table 10. Rule table for Average waiting time and turn around time (medium grain).

AWT/TAT	VS	S	M	MH	H	EH
VS	D	C	C	C	C	C
S	C	C	C	C	C	C
M	C	C	C	C	B	B
MH	C	C	B	B	A	A
H	C	B	B	A	A	A
EH	B	A	A	A	A	A

Table 11. Rule table for Average waiting time and turn around time (coarse grain).

AWT/TAT	VS	S	M	MH	H	EH
VS	D	B	B	B	B	B
S	B	B	B	A	A	A
M	A	A	A	A	A	A
MH	A	A	A	A	A	A
H	A	A	A	A	A	A
EH	A	A	A	A	A	A

Table 12. Rule table for average waiting time and turn around time (independent grain).

AWT/TAT	VS	S	M	MH	H	EH
VS	D	B	B	A	A	A
S	B	B	A	A	A	A
M	B	A	A	A	A	A
MH	A	A	A	A	A	A
H	A	A	A	A	A	A
EH	A	A	A	A	A	A

feature partitions and the scheduling classes used are class A (agile algorithm), class B (flexible co scheduling), class C (gang scheduling) and class D (first come first served scheduling), a total of 48 rules are framed for the learning analysis. The rule table must be created to determine which output ranges are used. The table is an intersection of the defined scheduling metrics like average waiting time and the turn around time and Tables 9, 10, 11 and 12 shows the rule table for the fine grain workloads. The rule tables can be created for the other scheduling parameters also (Moratori et al., 2010; Sun et al., 2011).

Figure 2 shows the attempt of using the structure of the feed forward neural network to model the Mamdani approach of the fuzzy logic controller. It consists of five layers. Layer 1 (input layer) Layer 2 (fuzzification layer),

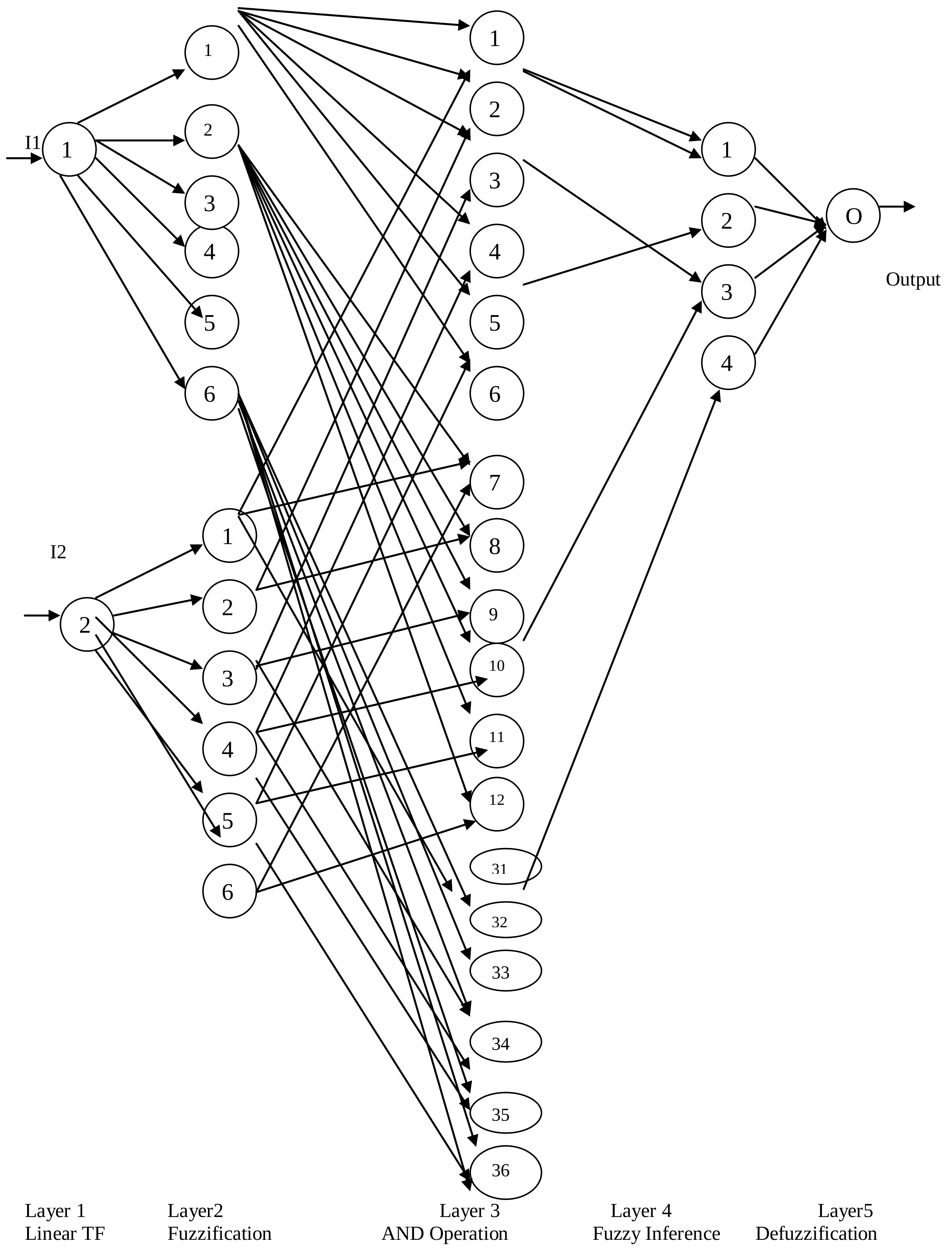

Figure 2. Neuro fuzzy model for learning.

Table 13. A sample chromosome.

Gene number	Value
Gene 1	5573
Gene 2	1380
Gene 3	2985
Gene 4	2985
Gene 5	4181
Gene 6	7622
Gene 7	1346
Gene 8	8076
Gene 9	8399
Gene 10	4722
Gene 11	6026
Gene 12	2179
Gene 13	6644
Gene 14	3360
Gene 15	7930
Gene 16	1440

Layer 3 (and operation implementation layer, Layer 4 (fuzzy inference layer and Layer 5 (defuzzification layer). The neurons in the first layer are nothing but transfer function. Thus the output of the neurons lying on this layer are nothing but their corresponding input values (scheduling metrics considered for the analysis). The second layer performs the fuzzification through which the membership values of the input variables are determined. The third layer represents all possible combination of the input variables (also known as antecedents) and the and operation. The fourth layer identifies the fired rules corresponding to the set of the input variables and the fifth layer performs the defuzzification to convert the fuzzified output in to its corresponding output layer. Thus, the weight determination of the neural network is done with the optimization techniques.

FITNESS EVALUATION

The algorithm for the fitness function is as follows: A back propagation network is taken as the learning algorithm with the configuration 2-36-4 (l-m-n) where l is the number of input neurons, m is the hidden neurons and n is the output neurons. The number of weights to be determined is 108. With each weight being a real number and assuming the number of digits d to be randomly generated for representing a weight value as 4, the string S representing a chromosome of weight is 108*4 = 432. To determine the fitness value for each of the chromosome, we extract weights from each of the chromosomes as shown in the figure. The sample chromosome is shown in Table 13.

Weight extraction

To determine the fitness value for each of the chromosome, we extract weights from each of the chromosomes. Let c_1, c_2, c_3…..c_n represent a chromosome and c_{k+1}, c_{k+2}…….$c_{(k+1)d}$ represent the kth gene in the chromosome. x_{kd+1} represents the position of the digit in the gene.

The actual weight W_k is given by:

$$W_k = \begin{cases} (c_{kd+2} * 10^{d-2} + c_{kd+3} * 10^{d-3} + \ldots\ldots + c_{(k+1)d})/10_{d-2} & \text{if } 5 <= x_{kd+1} <= 9 \\ -(c_{kd+2} * 10^{d-2} + c_{kd+3} * 10^{d-3} + \ldots\ldots + c_{(k+1)d})/10_{d-2} & \text{if } 0 <= x_{kd+1} < 5 \end{cases} \quad (5)$$

The algorithm for the fitness function is as follows. A back

Algorithm fitness neural

Step 1: Let (I_i, O_i) represent the input and output pair of the problem to be solved by the
Back Propagation Network with a configuration of 2-36-4

Step 2: For each chromosome C belonging to the current population

Step 2.1: Extract Weight W from C with the help of the Equation 5

Step 2.2: With the weights extracted, train the back propagation network for the given input and output instances.

Step 2.3: Calculate the net input to hidden unit Z_{inph} and its output Z_h

$$Z_{inph} = W_0 + \sum_{j=1}^{6} \sum_{i=1}^{n} (x_{ji} + x_{j+1i}) \quad (6)$$

$$Z_h = f(Z_{inph}) \quad (7)$$

Step 2.4: Compute the output Y_{ino} and O

$$Y_{ino} = w_1 + \sum_{h=1}^{n} (Z_h) \quad (8)$$

propagation network is taken as the learning algorithm

$$O = f(Y_{ino}) \quad (9)$$

Step 2.3: Calculate error E_i for each of the input and output instances using the

$E_i = \sum j\ (T\ j - O\ j)^2$ (T_j –target and O_j is the output) (10)
$E = \sqrt{E_i}/n$ where n is the number of instances (11)

Step 2.4: Calculate the fitness value F_i for each of the individual string of the population as
$F_i = 1/E$}

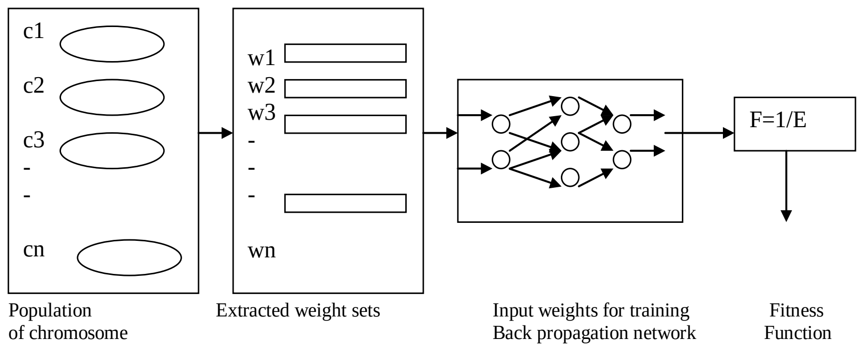

Figure 3. Fitness function computation for the population of chromosomes.

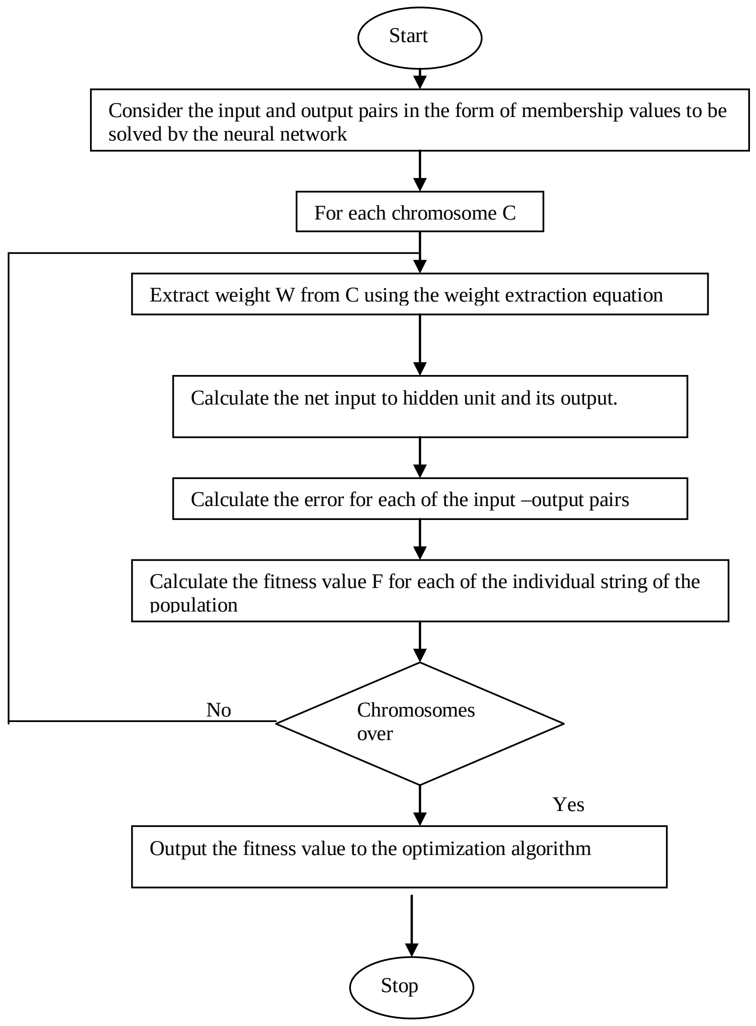
Start
Consider the input and output pairs in the form of membership values to be solved by the neural network
For each chromosome C
Extract weight W from C using the weight extraction equation
Calculate the net input to hidden unit and its output.
Calculate the error for each of the input –output pairs
Calculate the fitness value F for each of the individual string of the population
Chromosomes over
No
Yes
Output the fitness value to the optimization algorithm
Stop

Figure 4. Flowchart for fitness neural network.

Step 3: Output or return the fitness value to the GA/PGA/PSO_Tabu algorithm (Figure 4).

The algorithm gets the membership function inputs of the back propagation neural network and using the weights determined from the various optimization techniques, trains the network and finds the error value. If the error value is less, the chromosomes are considered as the best fittest chromosomes. This is explained neatly in Figure 3 and Table 14 shows the error in the output and the fitness values calculation for a sample of

Table 14. The fitness values from the initial population.

Chromosome	SUM(E)/13	SQRT(E/13)	F=1/E
1	0.00055972	0.023658358	42.26836
2	0.42416368	0.651278494	1.535441
3	0.01007314	0.100365023	9.96363
4	0.6503452	0.806439828	1.240018
5	0.68484746	0.82755511	1.208379

chromosomes.

OPTIMIZATION TECHNIQUES

Genetic algorithm

Genetic algorithm use a direct analogy of natural behavior, work with the population of individual strings each representing a possible solution to determine the weight values of the neural network. Each individual string is assigned a fitness value which is an assessment of how good a solution is to determine the weight values of the neural network. The high fit individuals participate in reproduction by cross breeding with other individuals in the population. This yields new individual strings as offspring which share some features with each parent. The least fit individual are kept out from reproduction and so die out. A whole new population of possible solution to the problem of finding the weight values is generated by selecting the best individuals from the current generation. Genetic and evolutionary algorithms wok based on Darwin's principle of natural selection (that is the survival of the fittest) have been used as a optimization tool for weight determination in the artificial neural network. It is basically an iterative search technique working based on the concept of probability. A Genetic algorithm starts with a population of initial solutions generated at random. The fitness or goodness value of each solution in the population is calculates. The paper uses initial population of real coded chromosomes which represent the weight values for the neural network. Until the population converges, the fitness of the chromosomes are executed using the algorithm Fitness_Neural, which selects the best weight values for the consecutive generations. The genetic algorithm uses the two point cross for best offspring across the generations. The same concept is applied to the parallel genetic algorithm where the population is divided by the master processor and the fitness evaluation is done in the slave processor. By doing the parallel implementation of the genetic algorithm, it guides the back propagation network to speed up the training by finding the weight calculation for the network in a quick time.

Algorithm GA

Step 1: Generate the initial population P of real coded chromosome which represent the weight values for neural network

Step 2: While the current population P has not converged
{Step 2.1: Calculate the fitness value for each of the chromosome using the algorithm Fitness Neural
Step 2.2: Keep the best of the individuals and terminate the worst of the individuals
Step 2.3: Using the two points cross over operation, reproduce the offspring from the current Population
Step 2.4: Call the current population}

Step 3: Extract weight from the current population to be used by the back propagation neural Network (Figure 7)
Algorithm Parallel_GA

Step 1: Generate the initial population P of real coded chromosome, which represent the weight values for the neural network.

Step 2: The population is divided into subpopulations. The master processor stores the population and sends the subset of the population to its slaves .

Step 3: While the current population P has not converged, the master sends the subset of the population to its slaves
Step 3.1: Generate the fitness value of the each of the chromosome in the slave processor by calling the algorithm Fitness Neural
Step 3.2: Keep the best of the individuals and terminate the worst
Step 3.3: Send the fitness value to the master processor
Step 3.4: The master processor does the two point crossover of the subpopulation and sends to the slaves.

Step 4: Extract weight from the current population to be used by the back propagation neural network (Figure 8)

Figures 5 and 6 shows a sample chromosome and the weight determination used by all the optimization techniques. Table 15 shows how the genetic algorithm makes a two point crossover with the probability of more that 50% for the new off springs and Table 16 shows the parameter used by the genetic algorithm and the particle swarm optimization and the same parameters can also be used for the hybrid algorithms.

```
602067558648192021379105492196115340951298642364917052867249936l
0859033952376808602275686989175150017537213343686044196041253640
8498565030867480359302519241001635223821112224859326559326939706
3287365745833491119138446377226253580746846641936205085271648181
```

Figure 5. Initial population of chromosomes.

3.710000	-3.770000	-2.020000	0.940000	-3.760000
-6.390000	5.080000	1.950000	3.900000	-8.730000
2.100000	0.780000	6.020000	-2.650000	9.130000
-1.450000	-1.540000	-7.720000	-2.300000	1.030000

Figure 6. Weights extracted from the initial population.

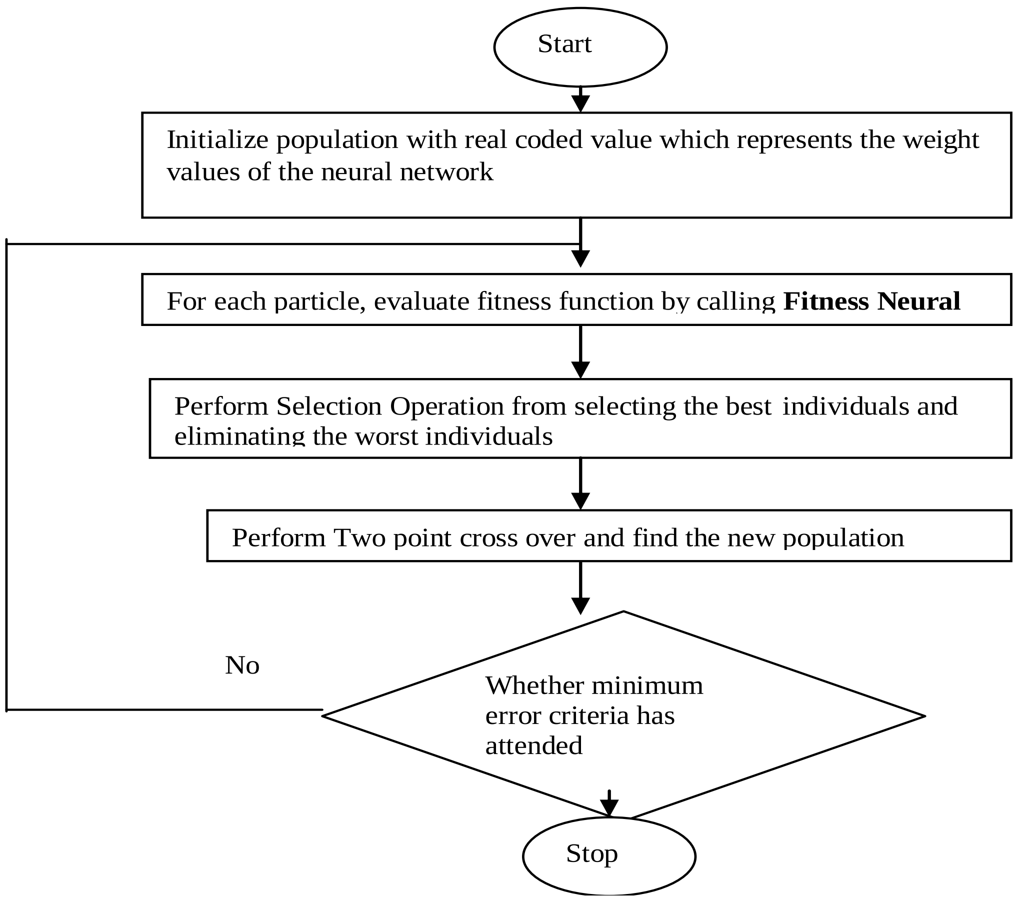
Start
Initialize population with real coded value which represents the weight values of the neural network
For each particle, evaluate fitness function by calling **Fitness Neural**
Perform Selection Operation from selecting the best individuals and eliminating the worst individuals
Perform Two point cross over and find the new population
No
Whether minimum error criteria has attended
Stop

Figure 7. Flowchart of GA optimization for weight determination.

Particle swarm optimization

Particle Swarm optimization is a population based stochastic optimization technique modeled based on swarm intelligence. Particle swarm optimization is a computational method that optimizes a problem by

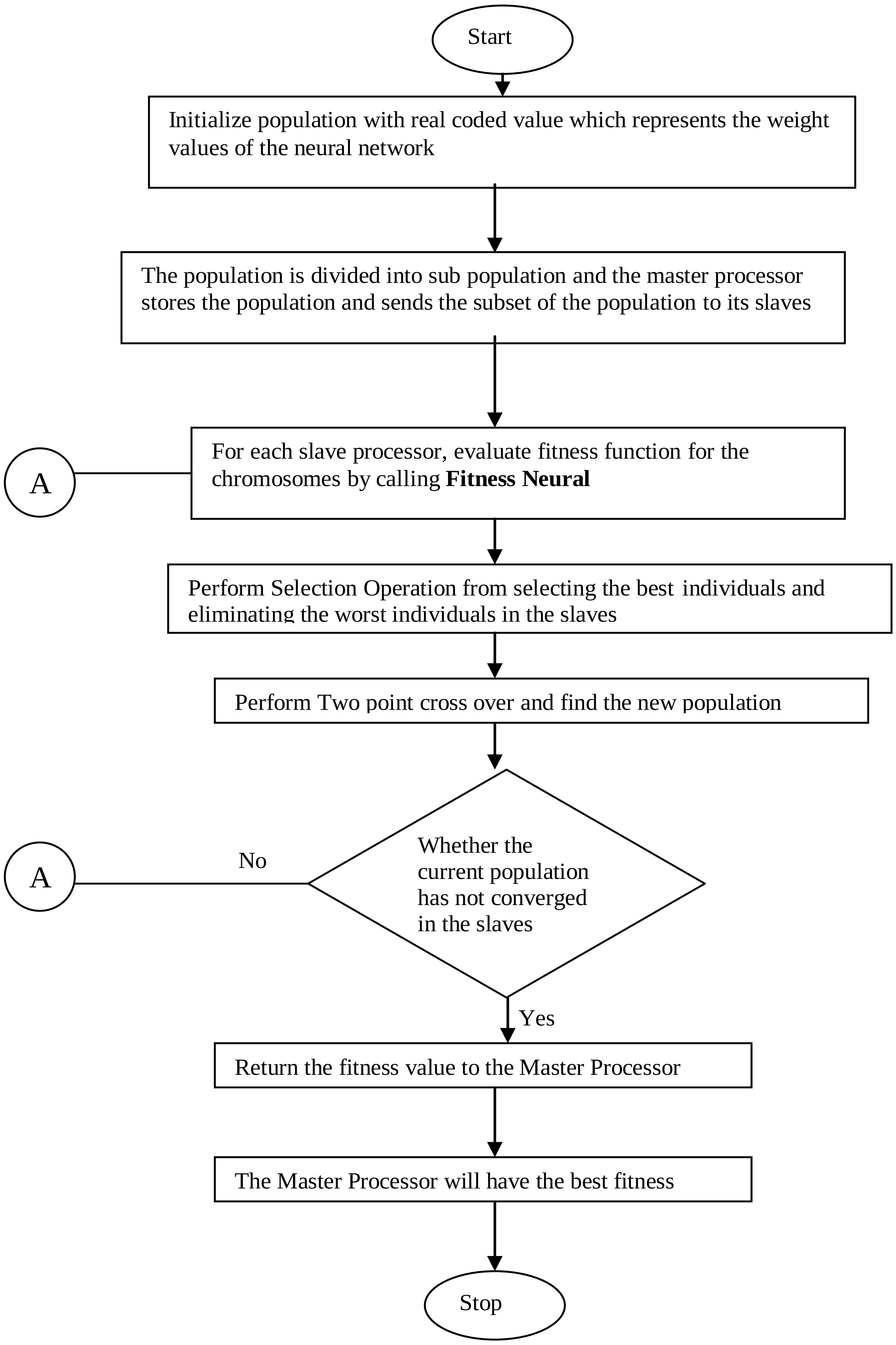
Start
Initialize population with real coded value which represents the weight values of the neural network
The population is divided into sub population and the master processor stores the population and sends the subset of the population to its slaves
For each slave processor, evaluate fitness function for the chromosomes by calling **Fitness Neural**
A
Perform Selection Operation from selecting the best individuals and eliminating the worst individuals in the slaves
Perform Two point cross over and find the new population
Whether the current population has not converged in the slaves
No
A
Yes
Return the fitness value to the Master Processor
The Master Processor will have the best fitness
Stop

Figure 8. Flowchart of PGA optimization for weight determination.

Table 15. Selection of parent pairs and their cross over positions.

S/N	Parent pairs represented by chromosome number	Cross over position	
		Start position	End position
1.	(2, 25)	7	11
2.	(3,10)	2	20
3.	(4,15)	4	10
4.	(5, 6)	3	6
5.	(7, 18)	9	16
6.	(8, 21)	10	15
7.	(9, 27)	20	24
8.	(10, 22)	40	47
9.	(13, 20)	34	40
10.	(17, 28)	36	50

Table 16. Parameters Used in GA and PSO.

S/N	Genetic algorithm	Particle swarm optimization
1.	Population size: 50	Swarm size: 50
2.	Max generations: 100	Max generations: 100
3.	Selection: Normal geometry	c1, c2 = 2
4.	Cross over: Two point	

iteratively trying to improve a candidate solution with regard to a given measure of quality. The idea of the optimizer was inspired by social behavior of bird flocking. The birds travel through the whole feasible search space to find the best flowers based on the objective function. In particle swarm optimization, each single solution is a bird in the search space, we call it as a particle. All of the particles have fitness function to be optimized and have velocities which direct the flying of the particles. Particle swarm optimization in initiated with a group of random solutions (that is) in our paper we have considered real coded particles which represent the weight values for the neural network. The particles search for the optima by updating in the generations. In every generation, each particle is updated by following the two best values. One is the best solution, the particle achieved so far and the other is the best solution of all the particles in the population. This global best value will give us the best optimism solution to the problem considered.

Algorithm PSO

```
Begin PSO
Step 1: Initialize Particles of real coded value with represent the weight values for the neural network
        Initialize the particle with position vector and velocity vector
        Initialize the size of the swarm, swarm size
        Initialize particle list to null
Step 2: do
        {
        For each particle
        Step 2.1: Calculate fitness value
        Step 2.2: If the fitness value is better than the best fitness value in history
         Set the current value as the new best value
        Step 2.3: Choose the particle with the best fitness value of the entire particle as the gBest
        For each particle
        Step 2.4: Calculate particle velocity and update particle position
      } while minimum error criteria is not attended
Step 3: Optimum solution is available in gBest (Figure 11)
```

Hybrid particle swarm optimization with the Tabu search

Tabu search is the metaheuristic local search algorithm that can be used for solving combinatorial optimization problem. Tabu search uses a local or neighboring search procedure to iteratively move from one potential solution y to a improved solution y1 in the neighborhood of y, until some stopping criteria has been satisfied. The tabu search uses a tabu list which is a memory structure used to filter which solutions will be admitted to the neighborhood of y. The paper discusses about the hybridization of PSO and the tabu search. The particle swarm optimization results in a premature convergence and produces poor quality of the solution by considering the local optimum. The tabu search use the adaptive memory processes for guiding search. First the population with feasible solution of particles are considered and the for each particle the pbest solution is being input to the tabu search, where the tabu search identifies the neighboring candidate and checks whether there are any best solutions .By doing so, the tabu search avoids the premature convergence of the particle swarm optimization. The same tabu search technique is done parallel to get the results in a quick time. The population is divided in to subpopulation and the searches are done by giving each one of the subpopulation to the various tabu search. The master processor in the PSO gets all the global solutions and finds the best from the results of the tabu search.

Algorithm hybrid PSO_Tabu

```
Begin PSO
Step 1: Initialize Particles of real coded value with represent the weight values for the neural network
        Initialize the particle with position vector and velocity vector
        Initialize the size of the swarm, swarm size
        Initialize particle list to null
Step 2: Do
    {
    Do
    {
     For each particle
 Step 2.1: Check the particle with the particle list
     Step 2.2: If the particle is present in the particle_list then
         Step 2.2.1 The position is already visited and makes the particle to move to its
                  Neighboring position that is not visited.
                  Else
         Step 2.2.1 Evaluate the fitness function to the particle particle_fit
     Step 2.3: If the fitness function particle_fit is better than the best value of the
              Particle (pbest) in history
   Step 2.4: Set pbest to particle_fit
   Step 2.5: Input pbest to tabusearch
Begin Tabusearch
Step 3: Initialize Tabulist, candidate list to null, sbest =pbest
    Do
    {
    Step 3.1: Generate the neighboring candidates
    Step 3.2: For each candidate
    {
        Step 3.3: If candidate not present in the tabulist then
                Add to the candidate list
     }
Step 4: Find the best candidate
Step 5: Update to the tabulist
Step 6: Update to the particlelist
} while (candidates exist)
Step 7: Update swarm size
} while (particles exist)
Step 8: Choose best of all particles and set as gbest
Step 9: Update particle velocity and particle position using the formula
        v[]= v[] + c1* rand() * (pbest[]-present[]) + c2* rand() * ( gbest [] –present[])
        present [] =present [] +v []
Where v[] is the particle velocity, present [] is the current solution. rand() is a random number between (0,1), c1 and c2 are the learning factors usually takes the value 2.
} while (all particles converge or maximum number of iterations reached)
End PSO (Figure 13)
```

RESULTS

The problem is to determine the weights for a back propagation network with a configuration 2-36-4 using the genetic algorithm parallel genetic algorithm and the swamp optimization techniques like particle swarm optimization a hybrid approach of particle swarm optimization and the tabu search and its parallel implementations. This analysis is done to tune the fuzzy logic controller which is used as a learning step for a parallel job scheduling using the neuro fuzzy system. Figure 9 show the optimization results that is, the fitness value calculation for the genetic algorithm (Figure 10). The figure shows a generation up to 100 and it is clear that the genetic algorithm converges from the generation of 55. Figure 12 shows the comparative results from genetic algorithm, parallel genetic algorithm and the particle swarm optimization. The parallel genetic algorithm and the particle swarm optimization seems to be converged quickly than the genetic algorithm that is from the generation from 45 onwards. Figures 14 and 15 show the optimization results of the hybrid PSO and the tabu search and its parallel implementation. The parallel implementation converges very quickly from the generation of 17 and also produces the maximum fitness value when compares to all the optimization techniques. Table 17 and Figure 16 show the consolidated fitness evaluation of the all the techniques and its error calculations. The error calculations are very minimum for the hybrid algorithms of PSO with the tabu search. Table 18 and Figure 17 show the computational time analysis computational time for the PSO is high when compared to the genetic algorithm. The computational time is improved for the PSO by the hybridization of the particle swarm optimization with the tabu search. The parallel implementation approach of the optimization techniques converges quickly when compared to the other. Thus, a

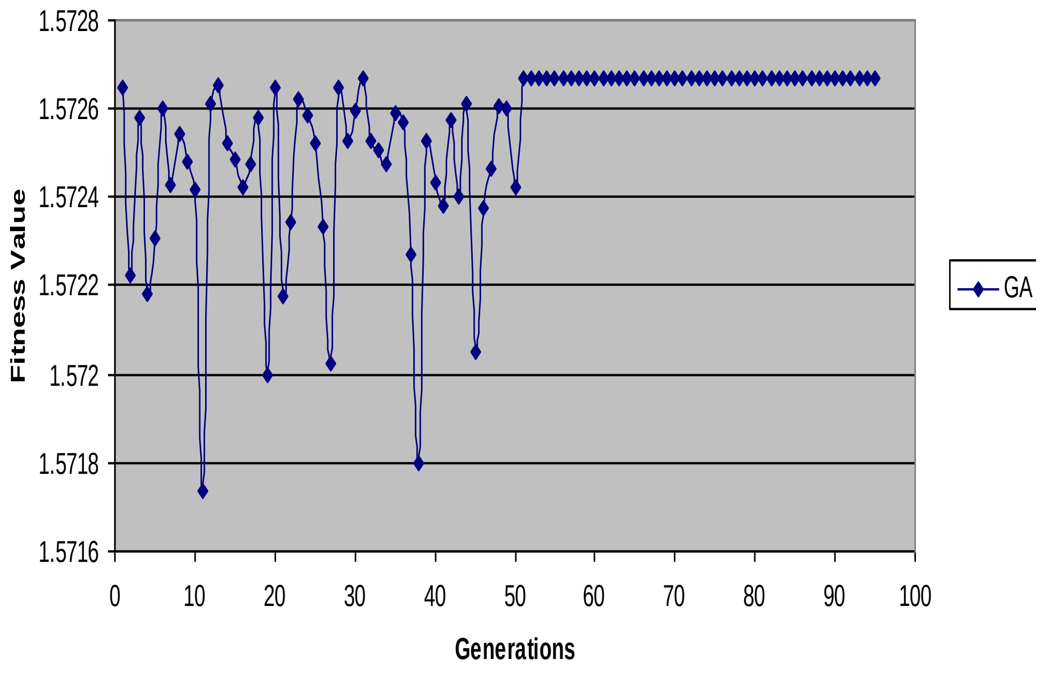

Figure 9. Optimization results of genetic algorithm.

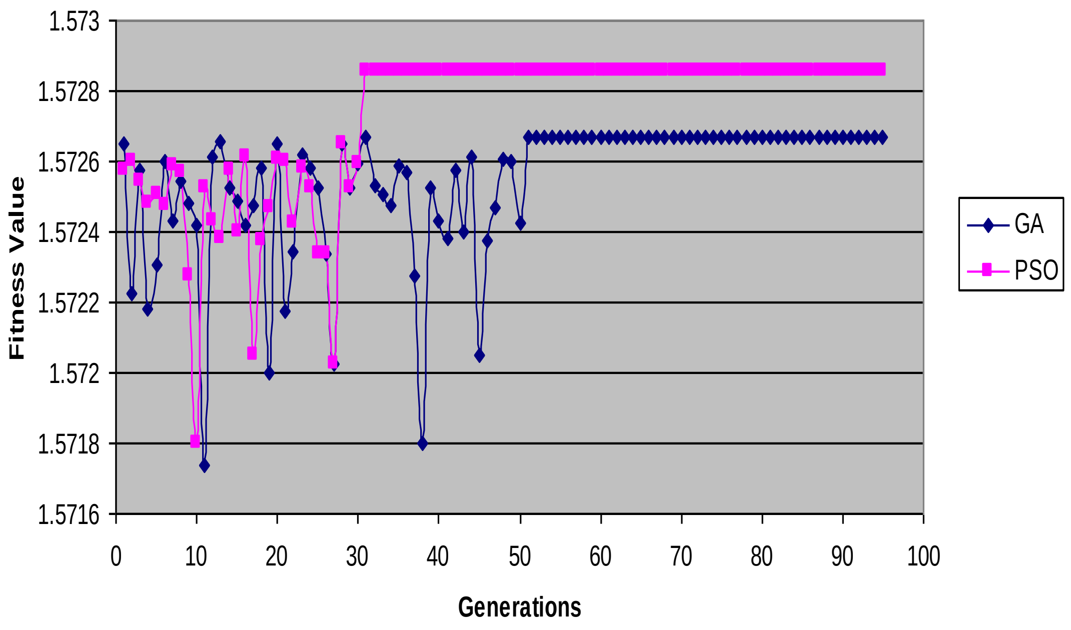

Figure 10. Comparative optimization results of genetic algorithm and particle swarm optimization.

DISCUSSION AND CONCLUSION

From Table 19 and Figure 18, we find that the computational time for the PSO is high when compared to the genetic algorithm. The computational time is improved for the PSO by the hybridization of the particle swarm optimization with the tabu search. The parallel implementation approach of the optimization techniques converges quickly when compared to the other. Thus a complete comparative analysis of the optimization techniques is done for the weight determination of the neural network, which will help the tuning of the fuzzy

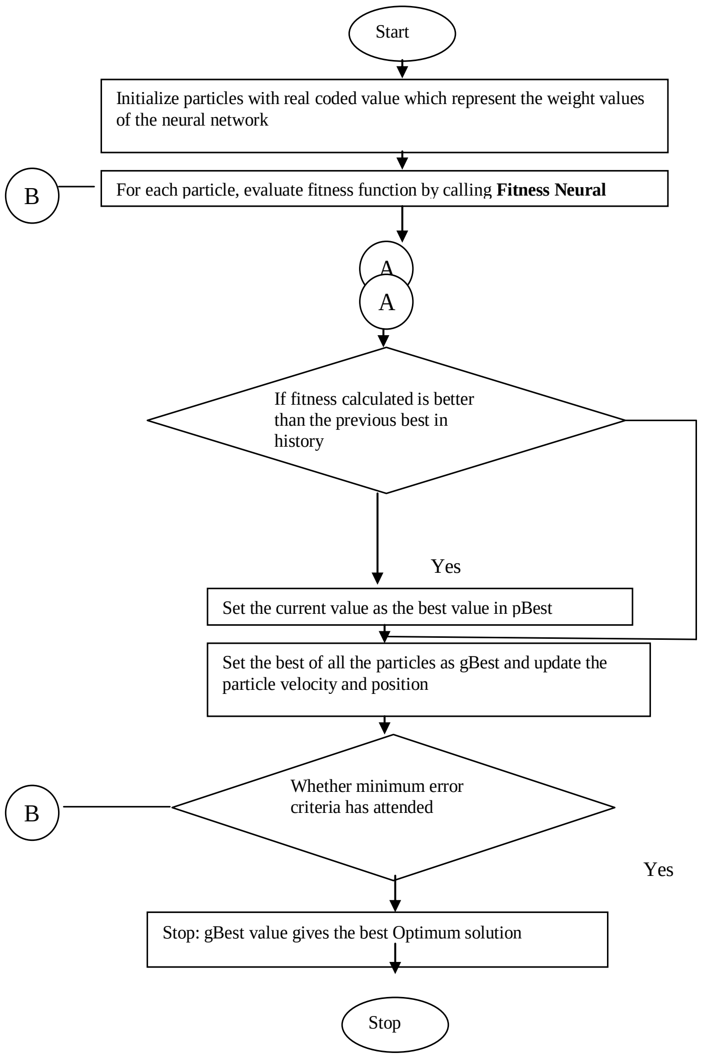
Start
Initialize particles with real coded value which represent the weight values of the neural network
B
For each particle, evaluate fitness function by calling **Fitness Neural**
A
A
If fitness calculated is better than the previous best in history
Yes
Set the current value as the best value in pBest
Set the best of all the particles as gBest and update the particle velocity and position
B
Whether minimum error criteria has attended
Yes
Stop: gBest value gives the best Optimum solution
Stop

Figure 11. Flowchart of PSO Optimization for weight determination.

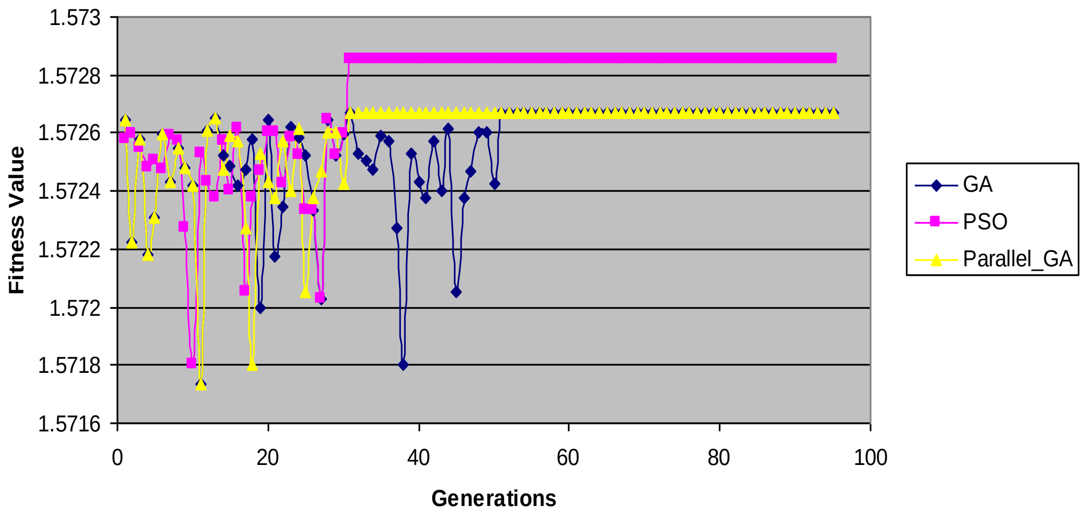

Figure 12. Comparative optimization results of genetic algorithm, parallel genetic algorithm and particle swarm optimization.

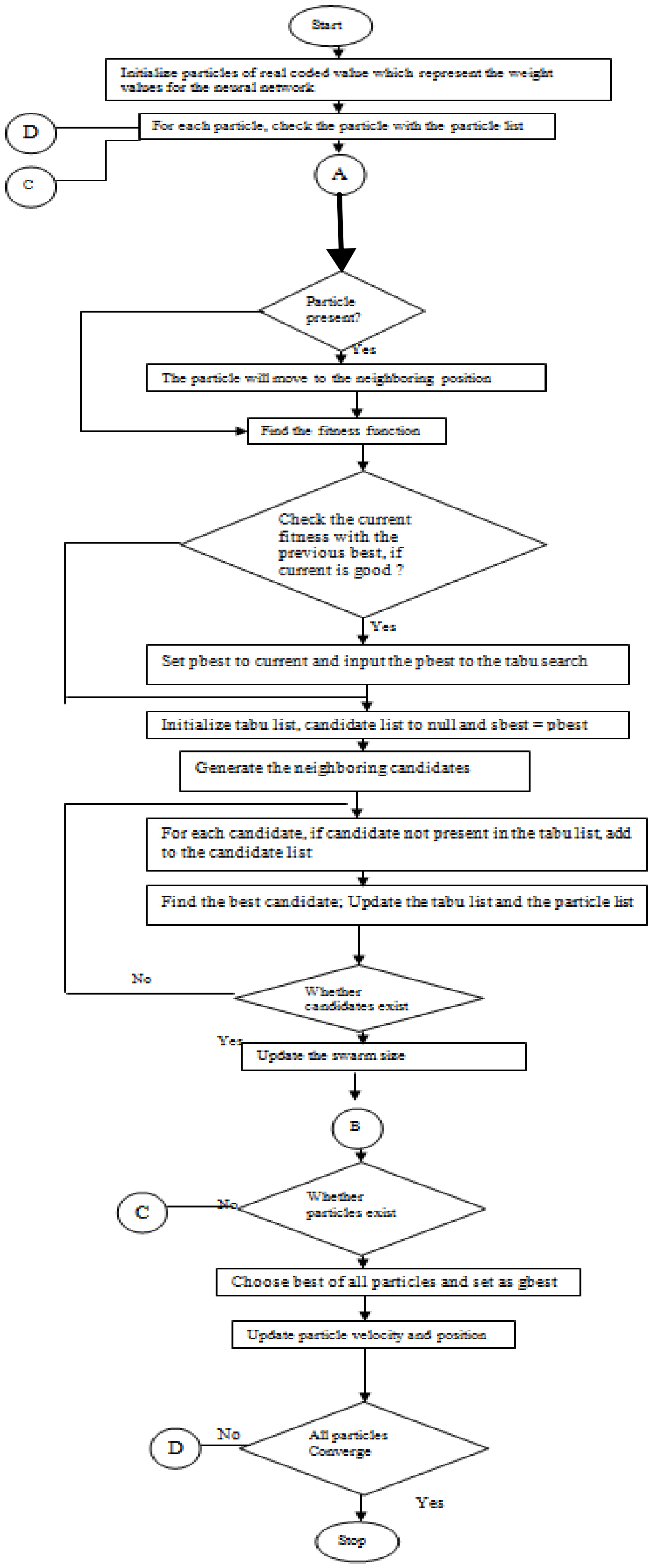
Start
Initialize particles of real coded value which represent the weight values for the neural network
D
For each particle, check the particle with the particle list
C
A
Particle present?
Yes
The particle will move to the neighboring position
Find the fitness function
Check the current fitness with the previous best, if current is good ?
Yes
Set pbest to current and input the pbest to the tabu search
Initialize tabu list, candidate list to null and sbest = pbest
Generate the neighboring candidates
For each candidate, if candidate not present in the tabu list, add to the candidate list
Find the best candidate; Update the tabu list and the particle list
No
Whether candidates exist
Yes
Update the swarm size
B
C
No
Whether particles exist
Choose best of all particles and set as gbest
Update particle velocity and position
D
No
All particles Converge
Yes
Stop

Figure 13. Flowchart of hybrid PSO and the tabu search optimization for weight determination.

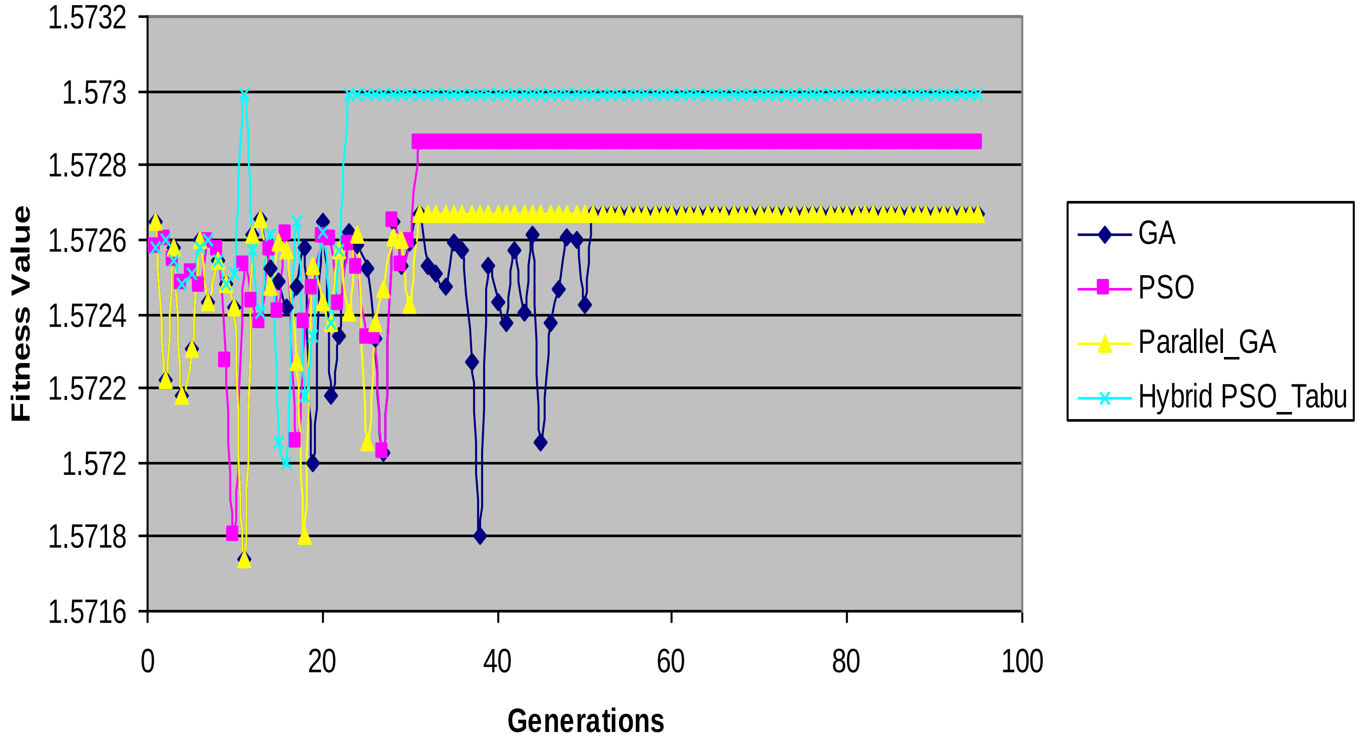

Figure 14. Comparative optimization results of genetic algorithm and particle swarm optimization with tabusearch.

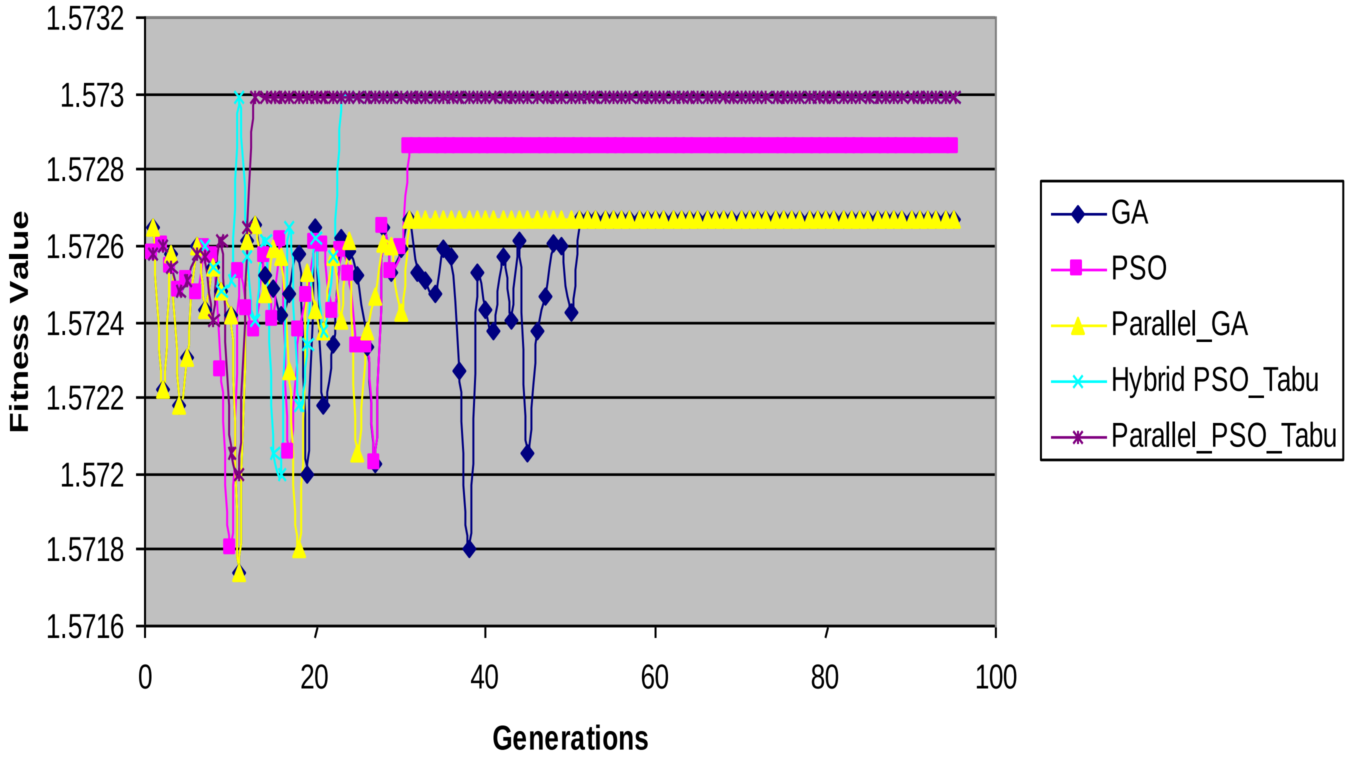

Figure 15. Comparative optimization results of genetic algorithm and particle swarm optimization with tabu search (parallel).

Table 17. Fitness evaluation of the optimization techniques with error calculation.

Input	Output	Output(GA)	Output (Parallel GA)	Output (PSO)	Output (Hybrid PSO_Tabu)	Output parallel PSO_Tabu	Error (GA)	Error (Parallel GA)	Error (PSO)	Error (Hybrid PSO_Tabu)	Error parallel PSO_Tabu
(0.9,0.793)	0.72	0.72	0.71999	0.71999	0.7211	0.7211	0.000833	0.00083	0.00139	0.15278	0.15278
(0.75,0.8)	0.54	0.54	0.54	0.53999	0.5401	0.5401	0.000370	0.00037	0.00185	0.01852	0.01852
(1,0)	1	0.9883	0.98828	0.98999	0.99999	0.99999	1.172000	1.18590	1.00100	0.00100	0.00100
(0.16,0)	0.16	0.1789	0.17894	0.15998	0.16023	0.16023	11.835625	10.58305	0.01250	0.14375	0.14375
(0.356,0.98)	0.655	0.655	0.655	0.654998	0.65501	0.65501	0.000305	0.00031	0.00031	0.00153	0.00153
(0.657,0.645)	0.651	0.651	0.651	0.65099	0.651001	0.651001	0.000307	0.00031	0.00154	0.00015	0.00015
(0.65,0)	0.65	0.6568	0.65676	0.6521	0.65002	0.65002	1.040462	1.02975	0.32308	0.00308	0.00308
(0.956,0.965)	0.971	0.9383	0.93834	0.9711	0.9710008	0.9710008	3.363543	3.48061	0.01030	0.00008	0.00008
(0.955,0.956)	0.956	0.9308	0.93083	0.9559	0.956003	0.956003	2.633159	2.70437	0.01046	0.00031	0.00031
(0.974,0)	0.974	0.9863	0.98626	0.9743	0.97399	0.97399	1.258932	1.24328	0.03080	0.00103	0.00103
(1,0.997)	0.997	0.949	0.94905	0.9982	0.9977	0.9977	4.809529	5.05253	0.12036	0.07021	0.07021
(1,1)	1	0.9501	0.95013	0.9845	0.9999	0.9999	4.987400	5.24920	1.55000	0.01000	0.01000
(0.998,0)	0.998	0.9881	0.98814	0.99822	0.99801	0.99801	0.987876	0.99773	0.02204	0.00100	0.00100

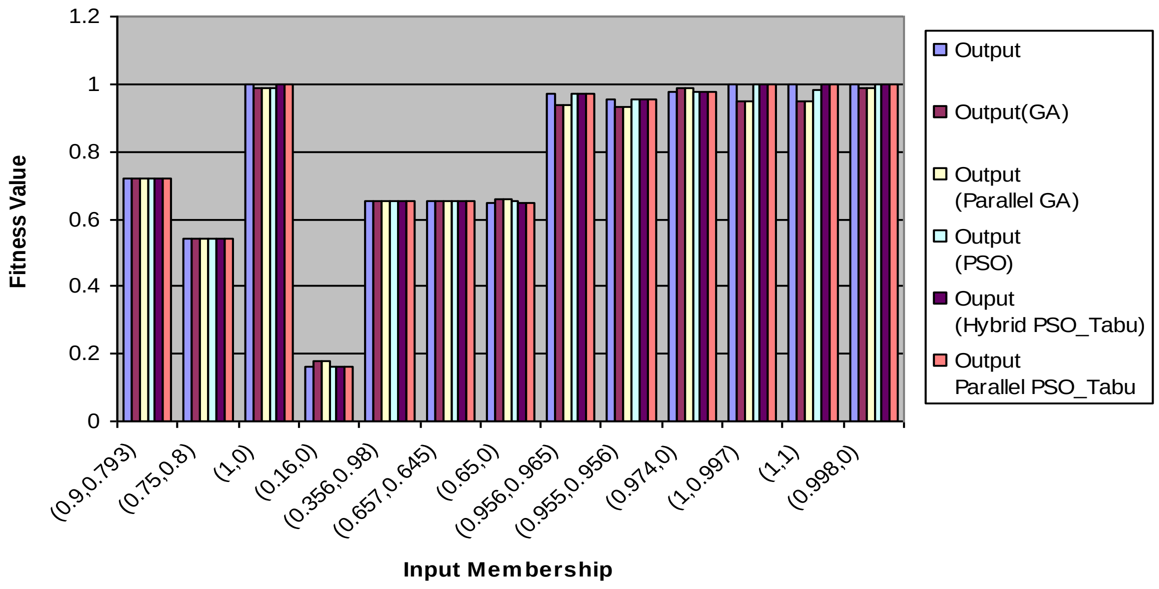

Figure 16. Comparative analysis of optimization results.

Table 18. Computational time analysis of all the optimization techniques.

S/N	Input (in membership values)	Output (in membership values)	Serial GA (ms)	Parallel GA (ms)	PSO (ms)	Hybrid PSO_Tabu (ms)	Parallel implementation PSO_Tabu (ms)
1	(0.9, 0.793)	0.72	9275.6	6456.7	9457.8	7426.5	5023.4
2	(0.75, 0.8)	0.54	8682.5	6072.4	8925.1	6993.2	4624.5
3	(1,0)	1	4060.3	3233.4	4311.4	3991.4	3214.5
4	(0.16,0)	0.16	6393.7	4910.4	6410.3	6012.1	5912.6
5	(0.356, 0.98)	0.655	9876.9	6993.5	9942.1	8624.8	7982.1
6	(0.657, 0.645)	0.651	11492.3	7547.9	12134.6	10342	9108
7	(0.65,0)	0.65	6739.9	4223.4	6825.6	5231.4	4921.1
8	(0.956, 0.965)	0.971	12285	7512.2	13212	12122	10123.4
9	(0.955, 0.956)	0.956	11064.6	6295.8	12116.1	10034.1	9346.9
10	(0.974, 0)	0.974	8143.2	5192	8242.1	8042.1	6643.8
11	(1, 0.997)	0.997	6778.3	5952.9	6991.2	6445.2	5892.6
12	(1,1)	1	2629	2496.5	2878.1	2512	1967
13	(0.998, 0)	0.998	6025.5	4898.5	6123.1	6004.3	5638.1
	Time (ms)		103446.8	71785.6	107569.5	93781.1	80398
	Time (sec)		103.4468	71.7856	107.5695	93.7811	80.398
	Time (min)		1 min 43 s	1 min 1 s	1 min 47 s	1 min 34 s	1 min 20 s

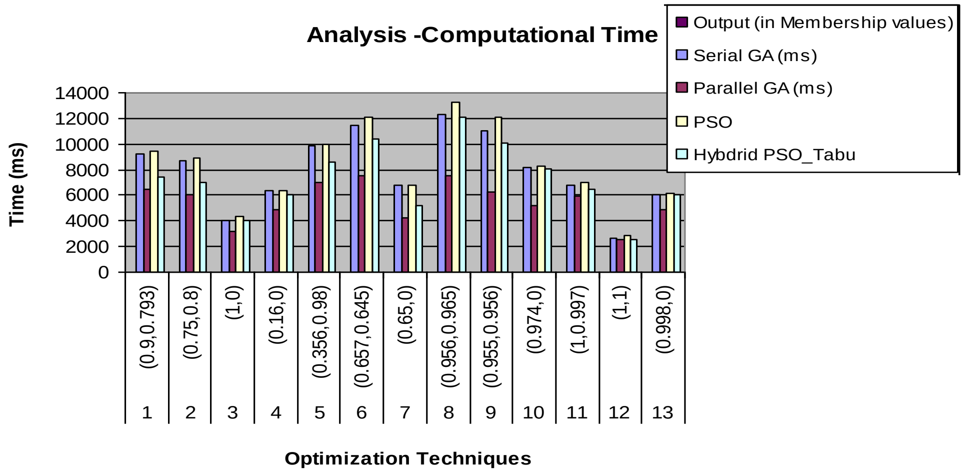

Figure 17. Computational time analysis of all the optimization techniques.

Table 19. Improvement of the computational time for the various optimization techniques.

Input (in membership values)	Output (in membership values)	Improvement of GA over PSO	Improvement of Parallel GA over GA	Improvements of PSO_Tabu over PSO	Improvement of PSO_Tabu over GA	Improvement of Parallel PSO_Tabu over GA	Improvement of Parallel PSO_Tabu over PSO	Improvement of Parallel PSO_Tabu over serial PSO_Tabu
(0.9,0.793)	0.72	1.92645224	30.3904869	21.4775106	19.93509854	45.84285653	46.88616803	32.3584461
(0.75,0.8)	0.54	2.71817683	30.0616182	21.6456958	19.45637777	46.73769076	48.1854545	33.8714751
(1,0)	1	5.824094262	20.3654902	7.42218305	1.696918947	20.83097308	25.44185183	19.4643483
(0.16,0)	0.16	0.258958239	23.1993994	6.21187776	5.968375119	23.16499054	23.36396113	18.288119
(0.356,0.98)	0.655	0.655797065	29.1933704	13.2497159	12.67705454	19.18415697	19.7141449	7.451767
(0.657,0.645)	0.651	5.293128739	34.3221113	14.7726336	10.00931058	37.27974383	40.59960773	30.3036163
(0.65,0)	0.65	1.255567276	37.3373492	23.3561885	22.38163771	26.98556358	27.90230895	5.93149061
(0.956,0.965)	0.971	7.016348774	38.8506309	8.25007569	1.326821327	30.61945462	35.48743566	29.6865204
(0.955,0.956)	0.956	8.678535172	43.099615	17.1837472	9.313486253	42.63778175	47.61598204	36.7466938
(0.974,0)	0.974	1.199936909	36.2412811	2.42656605	1.241526673	39.28922291	40.01771393	38.5260069
(1,0.997)	0.997	3.045256894	12.1770946	7.80981806	4.914211528	13.06669814	15.71404051	8.57382238
(1,1)	1	8.655015462	5.03993914	12.7201974	4.450361354	25.18067706	31.65630103	21.6958599
(0.998,0)	0.998	1.593963842	18.703842	1.94019369	0.351838022	18.04663513	19.35294214	17.757274

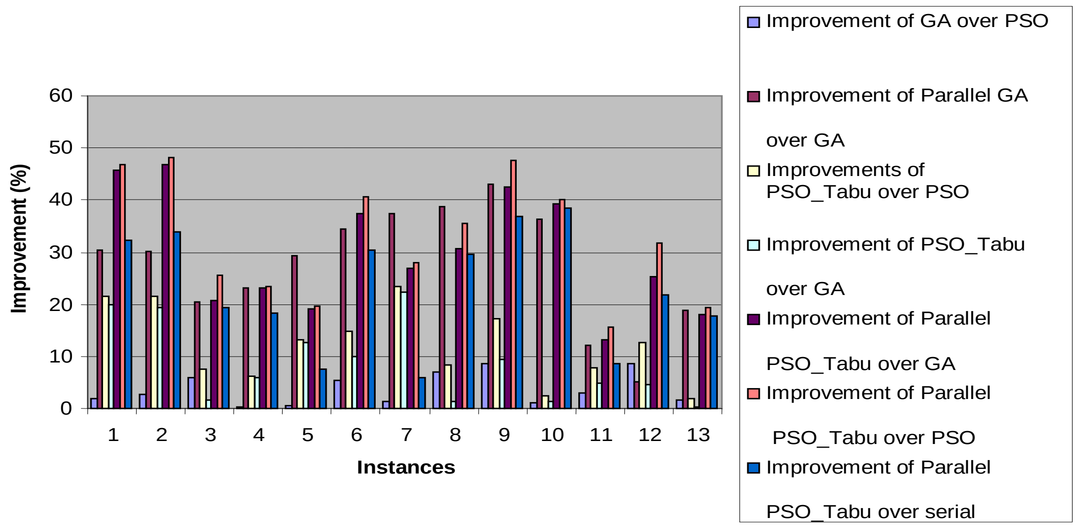

Figure 18. Improvements in computational time for the various optimization techniques.

logic controller used for the parallel job scheduling. The hybrid approach produces an average of 25 to 30% improvement in the computational time of the algorithm.

In this paper, an attempt is made to improve the performance of the fuzzy logic controller for the learning step of parallel job scheduling using the neural network. The fuzzy logic controller was used due to its robustness and can be easily modified, but the controller fails to give the correct defuzzification results, thus the structure of the neuro fuzzy approach is used to improve the performance of the fuzzy logic controller. The neural network's performance depends mainly on the weight determination of its layers and the paper concentrates on the determination of 108 weights. The weight determination is made with the optimization algorithms like genetic algorithm, parallel implementation of the genetic algorithm, Particle swarm optimization and the hybrid approach of PSO with the tabu search and with its parallel implementation. All these types of analysis are done to speed up training of the neural network. The results shows that the hybrid approach and its parallel implementation of the PSO with the tabu search produces good results in terms of the weight determination and also in terms of the converge.

REFERENCES

Avi N, Dror GF (2007). Probabilistic Backfilling 13th Workshop on Job Scheduling Strategies for Parallel Processing In Conjuction with 21st ACM Int. Conf. Super Comput. pp. 102-115.

Dan T, Yoav E, Dror GF (2007). Backfilling Using System –Generated Predictions Rather than User Runtime Estimates. IEEE Trans. Parallel Distrib. Syst. 18(6):789-803.

Dutot P, Pascual F, RZadca K, Trystram D (2011). Approximation Algorithms for the multiorganization scheduling problem. Parall. Distrib. Syst. IEEE Trans. 99(1).

Eitan F, Dror GF, Fabrizio P, Juan F (2005). Adaptive Parallel Job Scheduling with Flexible Coscheduling. IEEE Trans. Parallel Distrib. Syst. 16(11):1066-1077.

Ghedjati F (2010). Heuristic and a hybrid meta-heuristic for a generalized job shop scheduling problem. Evolutionary computing. Proc. IEEE Conf. pp. 1-8

Jintao M, Jun Y, Xiaoxu L (2010). Parallel- batching scheduling problem with family jobs for minimizing makespan, Industrial and Information systems. Proceed. Int. Conf. IEEE Explore, pp. 159-162.

Minh TN, Wolters L (2010). Using Historical Data to Predict Application Runtimes on Backfilling Parallel Systems, Parallel, Distr. Network based Process. 18th Eur. micro Int. Conf. pp. 246-252.

Moratori P, Petrovic S, Vazquez-Rodringuez JA (2010). Fuzzy Approach for Robust Job Shop Rescheduling. Proc. Fuzzy Syst. IEEE Int. Conf. pp. 1-7.

Sun H, Cao Y, Hsu WJ (2011). Effective adaptive scheduling of multiprocessor with stable parallelism feedback. Parall. Distrib. Syst. IEEE Trans. 22(4):594-607.

Thanushkodi K, Sudha SV (2009). A comparative evaluation of medium and coarse grain parallel jobs in job scheduling. IJSCI, ISSN, 21:0973-4864.

Thanushkodi K, Sudha SV (2009). An Approach for Parallel Job Scheduling Using Nimble Algorithm. IEEE Catalog Number: CFP0811G, ISBN: 978-1-4244-3595-1, Library of Congress: 2008911559.1-9.

Thanushkodi K, Sudha SV (2009). Synchronization granularity and processes based scheduling of parallel jobs with agile algorithm. AMSE France 30(2):1.