J. Parallel Distrib. Comput. 75 (2015) 101–106

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

# Dynamic task scheduling using a directed neural network

Binodini Tripathy[a], Smita Dash[b], Sasmita Kumari Padhy[c,*]

[a] KIIT University, Odisha, India
[b] SOA University Bhubaneswar, Odisha, India
[c] National Institute of Technology, Patna, India

## HIGHLIGHTS

- Development of the learning method for ANN.
- Development of a method for optimization of RBFNN.
- Use of DSO in task scheduling.
- Use of DSO trained ANN in task scheduling.
- Use of DSO trained RBFNN in task scheduling.

ARTICLE INFO

*Article history:*
Received 7 June 2014
Received in revised form
23 September 2014
Accepted 26 September 2014
Available online 8 October 2014

*Keywords:*
Task scheduling
Directed search optimization
Neural network

ABSTRACT

This article is based on the problem of work flow scheduling in grid environment of multi-processors. We, in this paper, introduce three novel approaches for the task scheduling problem using recently proposed Directed Search Optimization (DSO). In the first attempt, task scheduling is framed as an optimization problem and solved by DSO. Next, this paper makes use of DSO as a training algorithm to train (a) a three layer Artificial Neural Network (ANN) and then (b) Radial Basis Function Neural Networks (RBFNN). These DSO trained networks are used for task scheduling and interestingly yield better performance than contemporary algorithms as evidenced by simulation results.

## 1. Introduction

A computational grid is a coordinated phenomenon to share resources. This solves the problem in organization that is virtual, dynamic and multi-institutional. This coordinates resources that are not controlled centrally. The grid uses some standard protocols those are open and of general-purpose. The grid, in fact, is an interfacing tool for delivering important service quality. This virtualizes the resource, provides on-demand, and shares the resource among the organizations. This is consisting of a device to share the power of the computer and at the same time to store the data on the Internet [10,17]. There is a requirement for the management of the grid through an efficient scheduling approach.

Multiprocessor scheduling is an NP-hard problem [14,8,2]. The scheduling problem for tasks either dependent or independent is a well-studied discipline in the literature. In this article, we study the problem in an environment that is heterogeneous. These dynamic scheduling approaches are applicable to any larger set of real-time applications. These applications can be executed deterministically. Some of the conventional approaches provide global optimum, however longer time for execution and limited application for real-world problems are the drawbacks [15], whereas some other conventional approaches used may be deterministic and fast, but however fall into local optima [6].

This led to the research studies for the application of meta-heuristics, because efficacy or application of these algorithms is not limited for a specific problem. Available approaches for multiprocessor scheduling are broadly classified into heuristics and meta heuristics categories. The tasks maintain a queue in heuristic approaches. This queue is a priority queue and the processor processes the tasks on a first come first served basis, while in meta-heuristic approaches, they solve a class of computing problems by hybridizing user-provided procedures. In fact, these are heuristics by themselves, but in an efficient manner. Accordingly, Genetic Algorithms (GA) [14,19], Ant Colony Optimization (ACO) [5], Particle Swarm Optimization (PSO) [22,4,1,13,18], etc., are used for a better solution of the problem. But the results are constrained by efficiency. Hence, this paper proposes DSO [23] for task scheduling.

* Corresponding author.
*E-mail address:* chavisiba@rediffmail.com (S.K. Padhy).

http://dx.doi.org/10.1016/j.jpdc.2014.09.015
0743-7315/© 2014 Elsevier Inc. All rights reserved.

The art of using the artificial neural network (ANN) for task scheduling has been gaining momentum since last three decades. ANN trained with Back Propagation (ANN–BP) once again falls short of providing exact solution to the problem. Hence, this paper proposes DSO [23] as a training algorithm for ANN to be used in task scheduling.

Using neural networks has the limitations of large complexity and also fails because of over-fitting, local optima. On the other hand, RBFNNs, with only one hidden layer, have the ability to find global optima. In addition to less computational complexity, simulations performed in the literature reveal that the RBFNN produces superior performance as compared to other existing ANN-based approaches. Hence the works on task scheduling using RBFNN became an established and an active area of academic research and development [16,7,21,12].

However, there still are some difficulties with building RBFNNs. Main problems with RBFNNs are in determining the number of RBFs, number of cluster centers, etc. The conventional approaches consume longer time in determining the parameters using trial-and-error methods. Selecting the free parameters for the RBFs is also an issue for RBFNN. To get rid of the above-mentioned problems like trial-and-error steps and that of local optimal, Barreto et al. [3] used GA and Feng [9] used PSO to decide the centers of hidden neurons, spread and bias parameters by minimizing the Mean Square Error (MSE) of the desired outputs and actual outputs. In this paper we use DSO [23] for the training of ANN and RBFNN equalizers.

The paper is organized as follows: Section 2 discusses the problem of task scheduling. Sections 3–5 respectively discuss proposed approaches, DSO, DSO-trained ANN, and DSO-trained RBFNN. Performance of proposed approaches is evaluated through simulations explained in Section 6. Finally conclusion of the paper is outlined in Section 7.

## 2. The problem

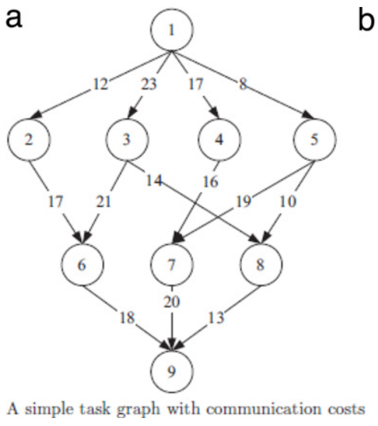
This article considers the problem of assignment of task in the following manner. In the multiprocessor scheduling problem, the schedule is normally reflected as a task graph (TG). A simple task graph, used throughout this article, with communication and computation costs is shown in Fig. 1 and devised in a way used in [20]. The task graph $T = (N, S)$ is a set $N$ of $n$ nodes and a set $S$ of $s$ sides. Here, the nodes correspond to tasks derived from the applied task, and the sides correspond to constrained conditions among the tasks that are dependent. To be more specific, each side $s_{i,j} \in S$ between task $n_i$ and $n_j$ meaning that the objective output of task $n_i$ can transmit to task $n_j$ to make the next task $n_j$ to start execution. With reference to Fig. 1, the task with no predecessor is called as the entry task and the task with no successor is called the exit task.

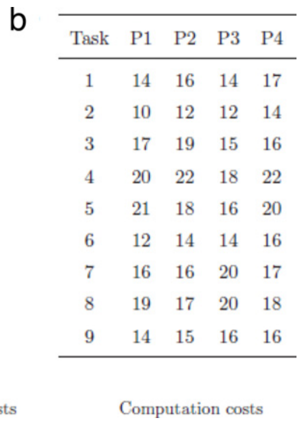
The weights $w_i$ attached to the tasks $n_i \in N$ are positive and termed as computation cost. However, the computing efficiency of the processors in heterogeneous environments is different from each other. The weights $w_{i,j}$ and $\hat{w}_i$ respectively are computation cost of the task $n_i$ on the processor $p_j$ and average computation cost. The computation cost of a task in this paper refers to the task execution time.

The nonnegative weight $c_{i,j}$ associated with side $s_{i,j} \in S$ corresponds to its communication cost between dependent tasks $n_i$ and $n_j$. If the tasks those are dependent are with different processors, then we need to compute the communication cost. Hence, actual communication cost is zero for the tasks with the same processor.

Consider a TG for $T$ tasks and with $P$ processors. For the task $n_i$, its earliest start time $T_{\text{start}}(n_i, p_j)$ on the processor $p_j$ is defined as:

$$T_{\text{start}}\left(n_i, p_j\right) = \max\left\{T_{\text{free}}\left(p_j\right), T_{\text{ready}}\left(n_i, p_j\right)\right\}. \quad (1)$$




b

| Task | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| 1 | 14 | 16 | 14 | 17 |
| 2 | 10 | 12 | 12 | 14 |
| 3 | 17 | 19 | 15 | 16 |
| 4 | 20 | 22 | 18 | 22 |
| 5 | 21 | 18 | 16 | 20 |
| 6 | 12 | 14 | 14 | 16 |
| 7 | 16 | 16 | 20 | 17 |
| 8 | 19 | 17 | 20 | 18 |
| 9 | 14 | 15 | 16 | 16 |

Computation costs

**Fig. 1.** An example of multiprocessor scheduling problem. (a) A simple task graph with communication costs and (b) computation costs [20].

Here, $T_{\text{free}}(p_j)$ is the time when processor $p_j$ is free for task $n_i$ to be executed. Most of the time, $T_{\text{free}}(p_j)$ refers to the completion of the last task by the processor $p_j$. In some cases task $n_i$ may be injected in a time when the processor $p_j$ is idle. In other words, when the length of an idle time gap is more than $w_i$, $T_{\text{free}}(p_j)$ may finish before the finish time of the last task. Here, $T_{\text{ready}}(n_i, p_j)$ represents the time of arrival of the entire data set at the processor $p_j$, and defined as:

$$T_{\text{ready}}\left(n_i, p_j\right) = \max_{n_k \in \text{pred}(n_i)} \left[T_{\text{finish}}\left(n_k\right) + c_{k,i}\right]. \quad (2)$$

Here, $T_{\text{finish}}(n_k)$ is actual finish time for the execution of the task $n_k$ and pred $(n_i)$ is the set of predecessors for the same task $n_i$.

In non-pre-emptive environments, $T_{\text{finish}}(n_i, p_j)$ is the fastest time to finish the task $n_i$ with processor $p_j$ as:

$$T_{\text{finish}}\left(n_i, p_j\right) = T_{\text{start}}\left(n_i, p_j\right) + w_{i,j}. \quad (3)$$

When task $n_i$ is fixed for scheduling with the processor $p_j$, the quickest start and finish time with the processor $p_j$ are also actual start and finish time for the task, respectively (i.e., $T_{\text{start}}(n_i) = T_{\text{start}}(n_i, p_j)$ and $T_{\text{finish}}(n_i) = T_{\text{finish}}(n_i, p_j)$). Assuming that the starting time for the first task is time 0, the total length of schedule is termed as the *makespan* and is the actual largest finish time of the exit task, which is:

$$makespan = \max_i \left[T_{\text{finish}}\left(n_{\text{exit}}\right)\right]. \quad (4)$$

Hence, the objective of the problem is to process the task set with the processor set while minimizing the *makespan* having considered the constraints.

## 3. Task assignment using DSO [23]

This section first outlines basic definitions, parameter ranges and nomenclature used in Section 3.1, followed by algorithm steps of DSO as proposed by Dexuan Zou et al. [23] in Section 3.2.

### 3.1. Terminology

This sub-section outlines the terms those are better explained in Fig. 2.

- $j_g$: global best solution vector.
- $r$ is a random number in the region [0, 1]
- $x_i^j(k)$ and $x_i^j(k+1)$: value of $x_i^j$ in the $k$th iteration, and updated component (determined by $p_\alpha$) respectively.

```
for each particle $j \in [1, PS]$ and $j \neq j_g$ do
    for each dimension $i \in [1, N]$ do
        if $rand() < p_\alpha$
            $x_v = x_i^j(k) + (1+\alpha) \times \{x_i^{j_g}(k) - x_i^j(k)\}$
            If $x_v > x_{iU}$
                $x_v = x_{iU}$
            Else if $x_v < x_{iL}$
                $x_v = x_{iL}$
            End
            $x_i^j(k+1) = x_i^j(k) + r \times (x_v - x_i^j(k))$
        else
            $x_s = x_i^j(k) - \beta \times \{x_i^{j_g}(k) - x_i^j(k)\}$
            if $x_s > x_{iU}$
                $x_s = x_{iU}$
            else if $x_s < x_{iL}$
                $x_s = x_{iL}$
            end
            $x_i^j(k+1) = x_i^j(k) + r \times (x_s - x_i^j(k))$
        end
        if $rand() < p_m$
        $x_i^j(k+1) = x_{iL} + r \times (x_{iU} - x_{iL})$
        end
    end
end
```

**Fig. 2.** Procedure for DSO, step-3.

- $x_i^{j_g}(k)$ : $i$th component of $j_g$ in the $k$th iteration. $x_i^j(k)$ mimics $x_i^{j_g}(k)$.
- $x_{iL}$ and $x_{iU}$: the lower and the upper bounds of the $i$th position component, respectively.
- Forward region: PV, where, $P$ locates $x_i^j(k)$, $Q$ locates $x_i^{j_g}(k)$ and $V$ (on the forward extension line of segment PQ) locates $x_v$. This is the main searching region for DSO.
- Backward region: PS, $S$ (on line PQ, extended backward) locates $x_s$. This is a region that is auxiliary and prevents any premature convergence by slowing down the faster convergence of the DSO.
- Adaptive step: $\text{step}_i^j(k) = \left|x_i^{j_g}(k) - x_i^j(k)\right|$. This step balances among the global and the local search in the DSO.
- Genetic mutation step: improves the diversity of individuals, and thereby improves DSO through prevention of falling into any local minima.
- DSO parameters: (1) population size ($PS$); (2) number of iterations; (3) forward probability ($p_\alpha$); (4) forward coefficient ($\alpha$); (5) backward coefficient ($\beta$) and (6) genetic mutation probability ($p_m$).
- Range: $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, PS$.

### 3.2. DSO steps

DSO follows following four steps.

- Step 1: initialize the DSO parameters.
- Step 2: generate initial population.

Generate the initial population in the ranges $[x_{iL}, x_{iU}]$ as:

$$\text{Pop} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_N^1 \\ x_1^2 & x_2^2 & \cdots & x_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{PS} & x_2^{PS} & \cdots & x_N^{PS} \end{bmatrix}. \tag{5}$$

Here, $x_i^j$ represents the $i$th component of the $j$th candidate solution vector (CSV).

- Step 3: Update.

Solution vectors are to be updated using the procedure shown in Fig. 2.

- Step 4: Stopping criterion.

If the stopping criterion (number of iterations $K$) is satisfied, computation is terminated. Otherwise, Step 3 is repeated.

In developing the DSO-based approaches for task scheduling in this paper, a set of population, $N$ number of CSV, within a bound [0, 1] are selected, each representing the coefficients of filter. Then, DSO starts from the initial random CSVs to proceed repeatedly from generation to generation through three genetic operators. The selection procedure reproduces highly fitted individuals who provide the minimum mean square error (MMSE).

## 4. ANN trained with DSO for task assignment

### 4.1. ANN in task assignment

As mentioned earlier, works on ANN-based approaches are an established field of research, and the main aim of this section is to outline the concept for the ease of the reader.

We in this paper chose a three layer ANN for task assignment. MMSE is taken as criterion for the second layer of ANN to train the weights. In the third layer we use DSO for training and to provide an estimate of the input task. Then this estimated task is fed back to the second layer that performs task assignment. This enhances the performance of ANN for the design of the task schedule. The second layer computes the weights of ANN, while the third layer calculates the one step prediction of the CSV and the estimate of input using the DSO. The feedback of the estimation results is sent to the second layer for the task assignment.

### 4.2. DSO training ANN and task assignment

The training process of ANN using DSO is outlined in Fig. 3. The training process takes the form of teaching in an organization, i.e. administrator, teacher and student. Here, ANN takes two possible roles, that of an administrator and a student, while DSO discharges the function of a teacher.

## 5. RBFNN trained with DSO and task assignment

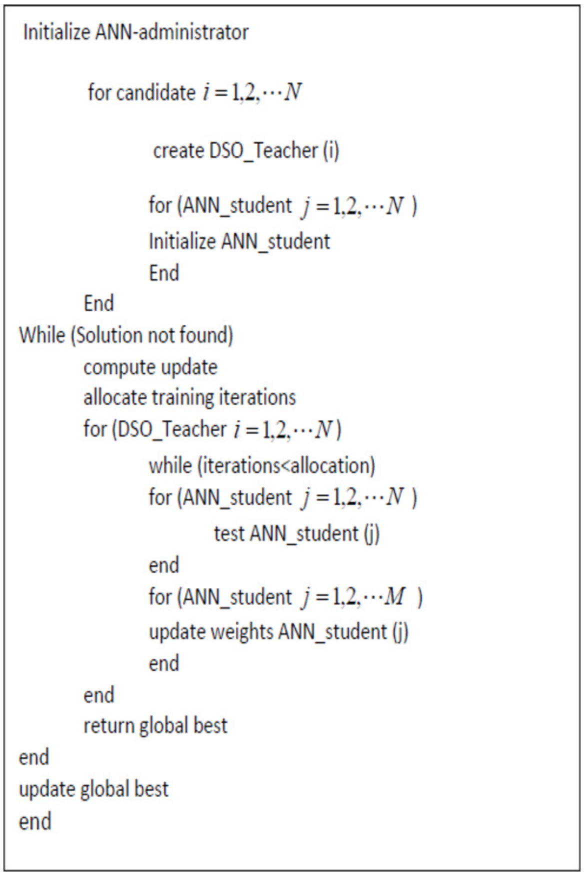
This section provides a brief introduction on RBFNN in task assignment, and then, DSO-based training for the RBFNN approach of task assignment. RBFNN used in this paper is shown in Fig. 4.

In developing the DRBF-based approach in this paper, the schedule is initially chosen from a population of $M$ (=2XN) CSVs. Each schedule constitutes the number of CSVs and each CSV represents one task that is assigned. Objective of this paper is to develop DRBF-based task assignment as shown in Fig. 4. DSO is used here as the training algorithm for RBFNN.

### 5.1. RBFNN in task assignment

As mentioned in Section 1, works on RBFNN in task assignment are an established field of research, and the main aim of this section is to outline the concept for the ease of the reader.

The RBFNN approach of task assignment is constructed by the task graph instead of the schedule, so the complex modeling of task schedule can be avoided. The important problem for design using RBFNN networks is to place the optimal tasks at the desired processors. We can find the relation between the desired



```
Initialize ANN-administrator
        for candidate $i = 1,2,\cdots N$
            create DSO_Teacher (i)
            for (ANN_student $j = 1,2,\cdots N$ )
            Initialize ANN_student
            End
        End
While (Solution not found)
        compute update
        allocate training iterations
        for (DSO_Teacher $i = 1,2,\cdots N$)
            while (iterations<allocation)
            for (ANN_student $j = 1,2,\cdots N$ )
                test ANN_student (j)
            end
            for (ANN_student $j = 1,2,\cdots M$ )
            update weights ANN_student (j)
            end
        end
        return global best
end
update global best
end
```

**Fig. 3.** Training ANN with DSO.

processors. This relation causes the centers can be composed from a data set with smaller size. In fact, this element set is the same as the task schedule. In other words, if the schedule is known, the RBFNN is designed. The object of the problem can be changed to find the optimal data set (output schedule). Task assignment problem becomes function maximization problem where target is maximum Bayesian likelihood and variables are assigned tasks. Because the mathematical relationships between the assigned tasks and the Bayesian likelihood are too complex to be formulated or cannot be formulated when the structure is nonlinear and is unknown, some apply PSO to solve this complex optimal problem with local minima. However use of DSO avoids any such requirements of PSO. Hence, this paper has taken a novel attempt using DSO to train RBFNN that learns the data set instead of centers and uses the pre-known mapping to obtain the centers of RBF network, for task scheduling.

### 5.2. *DSO-based training RBFNN for task assignment*

Method of training RBFNN is the same as that of training ANN explained earlier. However, exceptions are that here DSO is used to find optimal conditions for a number of RBFs, network, centers, etc. For each parameter of RBFNN optimization, we adopt a similar procedure.

Steps for the training algorithm used in this paper can be outlined as follows.

i. Initialize population of CSV. Each CSV denotes a network and centers associated as well as the bandwidths. Maximum of iterations is set as stopping criteria represented by MaxIteration. Set count $= 0$.
ii. Decode each CSV into a network. Compute the connection weights between the hidden layer and the output of the network by the pseudo-inverse method. Compute the fitness of each CSV.
iii. Run DSO to update the position.
iv. Set count $=$ count $+ 1$.
v. If, count $<$ MaxIterations, go to step ii, Otherwise, end.

## 6. Simulation results

The performance of the proposed methods was evaluated with makespan as criteria which is the length of the schedule for different algorithms. We have developed the simulation scenario the same as that used in [11] and used similar simulations as explained therein, but with different parameters for ease in comparison with population-based approaches.

We have not implemented GA. But, we compared with the results provided in [11]. While finding results for GA, we have chosen proportionate results according to the parameter selection. We have compared our algorithms with GA and PSO because of iterative search properties. Here each of these algorithms presents a solution after searching the subsets of possible solutions, and hence the longer time for execution. In a thorough examination for the

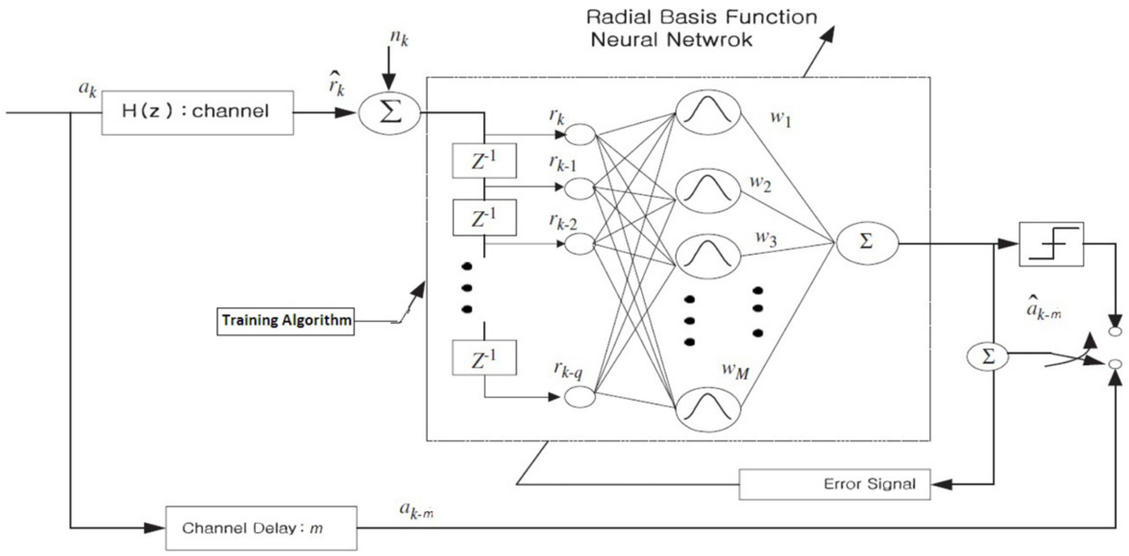


**Fig. 4.** Proposed DRBF.

**Table 1**
Simulation parameters.

| GA | | PSO | | DSO | |
|---|---|---|---|---|---|
| Parameter | Value | Parameter | Value | Parameter | Value |
| Number of iteration | 1000 | Number of iteration | 1000 | Number of iteration | 1000 |
| Number of individuals | 50 | Number of particles | 50 | Population size | 50 |
| Mutation ratio | 0.03 | Coefficient C1 | 0.7 | Forward probability | 0.8 |
| Crossover ratio | 0.9 | Coefficient C2 | 0.7 | Forward coefficient | 1 |
| Mutation type | Uniform | | | Backward coefficient | 10 |
| Crossover type | Single point | | | Genetic mutation probability | 0.01 |

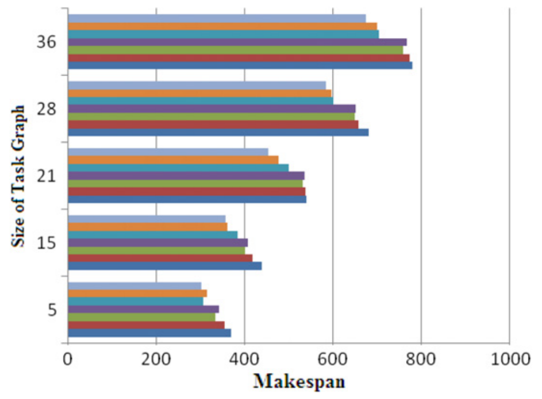


**Fig. 5.** Effect of size of task graph on makespan.

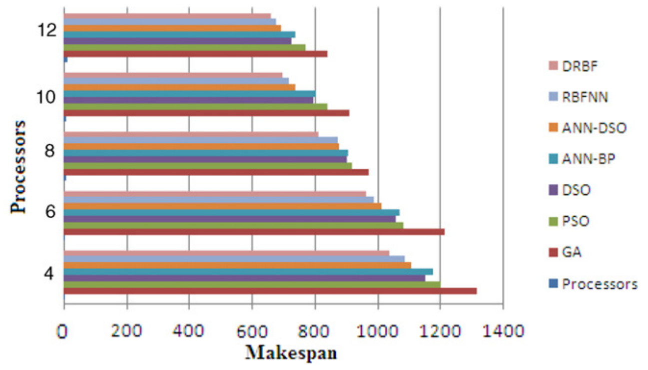


**Fig. 6.** Effect of number of processors on makespan.

effects by the parameters with the study of some random DAG graphs, we conducted three such experiments to evaluate the proposed methods. In experiment 1, the effect of size of task graph was the parameter. The simulations were repeated for 10 times. Since the variations are very less, without considering the actual data distribution information, the average result has been reported. In experiments 2 and 3, the effect of the number of processors and the number of iterations was taken as parameters respectively. Simulation parameters used for GA, PSO, and DSO are outlined in Table 1.

Experiment-1: Effect of size of task graph.

In the heterogeneous computing environments the same task will have a different compute cost on different processing scenarios. However, in the experiments conducted in this paper, we have chosen a single scenario and that is chosen as the best one. Following parameters are finalized for this experiment and used in the process of simulation:

- Size of task graph $(T) = 5, 15, 21, 28, 36$.
- Time for processing $=$ 25 per task.
- Time for communication $=$ 50/side.
- Processor number $(P) = 4$.
- Size of population $=$ 10, 30, 42, 56, 72.
- Max number of iterations $=$ 500.
- $c_1 = c_2 = 2$.

Fig. 5 shows the results of this experiment with the makespan provided by different approaches. All of our proposed strategies outperform the GA in all cases.

The primary merit of the other population-based strategies over the GA is the need for fewer resources. Another advantage is the ease in implementation with less sophisticated operators.

Experiment-2: effect of the number of processors.

We have studied this effect choosing a random TG from the STG graphs of size 200, and the proposed methods ran with a different number of processors for each of the runs. It is observed from the experiments that when the number of processors increases, execution time decreases. We have limited our study to a maximum number of processors at 12, but it can be further increased in real time applications in order to get the desired execution time.

Following parameters are finalized for this experiment and used in the process of simulation:

- Size of task graph $(T) = 200$.
- Time for processing $=$ 25/task.
- Time for communication $=$ 50/side.
- Processor number $(P) = 4, 6, 8, 10, 12$.
- Size of Population $=$ 400.
- Max number of iterations $=$ 500.
- $c_1 = c_2 = 2$.

It is observed from the experiments that when the number of processors increases, execution time decreases. We have limited our study to a maximum number of processors at 12, but it can be further increased in real time applications in order to get the desired execution time.

Fig. 6 depicts makespan obtained by different approaches in this experiment having different numbers of processors. It was observed that the processor number affects the behavior of the algorithms showing an ease in finding the near-optimal solution with the larger number of processors.

Experiment-3: Effect of the number of Iterations.

The effect of the number of iterations in the behavior of the proposed algorithm was evaluated.

Following parameters are finalized for this experiment and used in the process of simulation:

- Size of task graph $(T) = 5$.
- Time for processing $=$ 25/task.
- Time for communication $=$ 50/side.
- Processor number $(P) = 4$.
- Size of Population $=$ 20.
- Max number of iterations $=$ 30, 70, 100, 500.
- $c_1 = c_2 = 2$.

Fig. 7 shows the execution time for each algorithm. It is observed from the figure that the proposed approaches using ANN–DSO and & DRBF achieve significantly better result on all the values of GA.

From above results, we can see that:

- All the proposed approaches result in a reduction of makespan and also quickly.
- DRBF is the fastest in reducing the makespan.

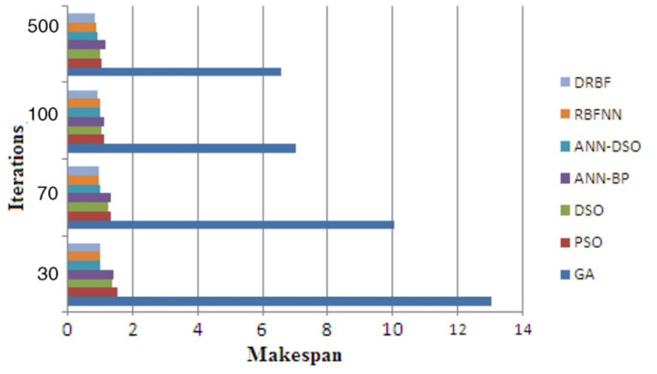


**Fig. 7.** Effect of iterations on execution time (in s).

## 7. Summary and future work

This paper proposed three novel and efficient approaches for work flow scheduling as observed simulation results. Main contributions of the paper can be outlined as:

- Development of learning method for ANN.
- Development of a method for the optimization of RBFNN.
- Use of DSO in task scheduling.
- Use of DSO-trained ANN in task scheduling.
- Use of DSO-trained RBFNN in task scheduling.

In addition to above contributions by this article, this paper also clearly points out a study on DAG dependency, i.e., the effect of DAG on the makespan. Since this paper is limited to one DAG, the study on DAG dependency may appear in some of future works.

## References

[1] T. Bakshi, et al., An evolutionary algorithm for multi-criteria resource constrained project scheduling problem based on PSO, Proc. Technol. 6 (2012) 231–238.
[2] Savaş Balin, Non-identical parallel machine scheduling using genetic algorithm, Expert Syst. Appl. 38 (6) (2011) 6814–6821.
[3] Barreto, et al., Growing compact RBF networks using a genetic algorithm, in: Proceedings of the VII Brazilian Symposium on Neural Networks, 2002, pp. 61–66.
[4] Ruey-Maw Chen, Chuin-Mu Wang, Project scheduling heuristics-based standard PSO for task-resource assignment in heterogeneous grid, Abstr. Appl. Anal. 2011 (2011) Article ID 589862.
[5] Wang Chen, Yan-jun Shi, Hong-fei Teng, Xiao-ping Lan, Li-chen Hu, An efficient hybrid algorithm for resource-constrained project scheduling, Inform. Sci. 180 (6) (2010) 1031–1039.
[6] Tzu-Chiang Chiang, Po-Yin Chang, Yueh-Min Huang, Multi-processor tasks with resource and timing constraints using particle swarm optimization, IJCSNS Int. J. Comput. Sci. Netw. Secur. 6 (4) (2006) 71–77.
[7] Pınar Çivicioğlu, Mustafa Alçı, Erkan BeŞdok, Using an exact radial basis function artificial neural network for impulsive noise suppression from highly distorted image databases, in: Advances in Information Systems, in: Lecture Notes in Computer Science, vol. 3261, 2005, pp. 383–391.
[8] Orhan Engin, Gülşad Ceran, Mustafa K. Yilmaz, An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems, Appl. Soft Comput. 11 (3) (2011) 3056–3065.
[9] H.M. Feng, Self-generating RBFNs using evolutional PSO learning, Neurocomputing 70 (2006) 241–251.
[10] Joon-Min Gil, Sungsuk Kim, JongHyuk Lee, Task scheduling scheme based on resource clustering in desktop grids, Int. J. Commun. Syst. (2013) http://dx.doi.org/10.1002/dac.2701. online article.
[11] S. Jin, et al., A Performance Study of Multiprocessor Task Scheduling Algorithms, Vol. 43, Springer, 2008, pp. 77–97.
[12] Harpreet Kaur, Balwinder Dhaliwa, Design of low pass FIR filter using artificial neural network, Int. J. Inf. Electron. Eng. 3 (2) (2013) 204–207.
[13] K.K. Mandal, et al., Daily combined economic emission scheduling of hydrothermal systems with cascaded reservoirs using self organizing hierarchical particle swarm optimization technique, Expert Syst. Appl. 39 (3) (2012) 3438–3445.
[14] Fatma A. Omara, Mona M. Arafa, Genetic algorithms for task scheduling problem, J. Parallel Distrib. Comput. 70 (1) (2010) 13–22.
[15] Dar-Tzen Peng, Kang G. Shin, Tarek F. Abdelzaher, Assignment and scheduling communicating periodic tasks in distributed real-time systems, IEEE Trans. Softw. Eng. 23 (12) (1997) 745–758.
[16] Robert J. Schilling, et al., Approximation of nonlinear systems with radial basis function neural networks, IEEE Trans. Neural Netw. 12 (1) (2001) 1–15.
[17] Uwe Schwiegelshohn, et al., Perspectives on grid computing, Future Gener. Comput. Syst. 26 (2010) 1104–1115.
[18] Q. Tao, et al., A rotary chaotic PSO algorithm for trustworthy scheduling of a grid workflow, Comput. Oper. Res. 38 (5) (2011) 824–836.
[19] Yun Wen, Hua Xu, Jiadong Yang, A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system, Inform. Sci. 181 (3) (2011) 567–581.
[20] J. Yang, et al., Task scheduling using Bayesian optimization algorithm for heterogeneous computing environments, Appl. Soft Comput. 11 (2011) 3297–3310.
[21] O. Yavuz, T. Yildirim, Design of digital filters with bilinear transform using neural networks, in: 16th IEEE Conference on Signal Processing, Communication and Applications, 2008, pp. 1–4.
[22] L. Zhang, et al., A task scheduling algorithm based on PSO for grid computing, Int. J. Comput. Intell. Res. 4 (1) (2008) 37–43.
[23] Dexuan Zou, et al., Directed searching optimization algorithm for constrained optimization problems, Expert Syst. Appl. 38 (2011) 8716–8723.

**Binodini Tripathy** received her graduate and post-graduate degrees in Electronics and Communication Engineering in 2000 and 2009 from IE (I), Kolkata, India, and ITER, SOA University, Bhubaneswar, Odisha, India, respectively. Presently she is pursuing her Ph.D. research under the guidance of Dr. Padhy, and also she is with the Department of ECE at KIIT University, Odisha, India. Her present research interests include multiprocessor scheduling, soft and evolutionary computing.

**Smita Dash** received an M.Tech degree from SOA University in 2014 and presently working for her Ph.D. degree.

**Sasmita Kumari Padhy** received her MCA from BPUT, Odisha, India, M.Tech from Berhampur University Odisha, India, and Ph.D. from Utkal University Odisha, India, respectively in 2003, 2008 and 2011. Dr. Padhy presently working with National Institute of Technology, Patna, Bihar, India. Her research interests include signal processing, soft and evolutionary computing and multiprocessor scheduling.