CHAPTER 2

PROPOSED IMPROVED PARTICLE SWARM OPTIMIZATION

The present chapter proposes an IPSO approach for multiprocessor task scheduling problem with two classifications, namely, static independent tasks and dynamic tasks with and without load balancing to minimize the makespan of the schedule.

2.1 INTRODUCTION

PSO is a popular swarm based optimization technique that mimics the bird’s flocking behavior. Though PSO shows better performance, premature convergence and local optima are the two major problems faced by the PSO approach for scheduling problem.

To overcome this difficulty, an IPSO is proposed, in which a split–up is made in the cognitive behavior. That is the particle is made to remember its worst position also. This modification helps in exploring the search space very effectively to identify the promising solution region. The introduction of the worst particle called bad experience component in the cognitive component yields a significant improvement in the results, when applied to the task scheduling problem. The inclusion of the worst particle plays a major role in the achievement of a better solution than with only using the good experience component called best particle in the velocity equation.
In the present chapter an IPSO approach to obtain better schedule with minimum convergence time is presented. The effectiveness of the proposed IPSO approach is demonstrated with datasets and is given in Tables 2.1 to 2.6.

2.2 PROPOSED IMPROVED PARTICLE SWARM OPTIMIZATION APPROACH

In the proposed IPSO algorithm, the velocity updation equation is slightly modified by splitting the cognitive component of the general PSO into two different components to obtain better results for the multiprocessor task scheduling problem. The first component is called as good experience component. This means that the particle has a memory about its previously visited best position. This is similar to the standard PSO method. The second component is given the name, bad experience component. The bad experience component helps the particle to remember its previously visited worst position. To calculate the new velocity, the bad experience of the particle is also taken into consideration. On including the characteristics of P_best and P_worst in the velocity updation process along with the difference between the present best particle and current particle respectively, the convergence towards the solution is found to be faster and an optimal solution is reached in comparison with conventional PSO approaches.

The new velocity update equation is given by,

\[ V_i = w \times V_i + C_{1g} \times r_1 \times (P_{best_i} - S_i) \times P_{best_i} + C_{1b} \times r_2 \times (S_i - P_{worst_i}) \times P_{worst_i} + C_2 \times r_3 \times (G_{best_i} - S_i) \]

(2.1)
The position update equation is given by,

\[
S_{i+1} = S_i + V_i
\]  

(2.2)

where,

- \(C_{1g}\) : self-confidence factor, which accelerates the particle towards its best position;
- \(C_{1b}\) : self-confidence factor, which accelerates the particle away from its worst position;
- \(C_2\) : Swarm confidence factor,
- \(P_{\text{worst } i}\) : worst position of the particle \(i\),
- \(r_1, r_2, r_3\) : uniformly distributed random numbers in the range [0 to 1],
- \(S_i\) : current position
- \(S_{i+1}\) : modified position,
- \(V_i\) : current velocity and
- \(V_{i+1}\) : modified velocity

The positions are updated using Equation (2.2). The inclusion of the worst experience component in the behaviour of the particle gives the additional exploration capacity to the swarm. By using the bad experience component, the particle can bypass its previous worst position and try to occupy the better position. Figure 2.1 shows the concept of IPSO searching points.
2.2.1 The Proposed IPSO Algorithm

**Step 1:** Select the number of particles, generations, tuning accelerating coefficients $C_{1g}$, $C_{1b}$, and $C_2$ and random numbers $r_1$, $r_2$, $r_3$ to start the optimal solution searching.

**Step 2:** Initialize the particle position and velocity.

**Step 3:** Select particles individual best value for each generation

**Step 4:** Select the particles global best value, which means particle near to the target among all the particles is obtained by comparing all the individual best values.

**Step 5:** Select the particles individual worst value.

**Step 6:** Update particle individual best (Pbest), global best (Gbest), particle worst (Pworst) in the velocity Equation (2.1) and obtain the new velocity.
Step 7: Update new velocity value in the Equation (2.2) and obtain the new position of the particle.

Step 8: Find the optimal solution with minimum ‘F’ by the updated new velocity and position.

The flowchart for the proposed model formulation scheme is shown in Figure 2.2

![Flowchart for the proposed IPSO](image)

**Figure 2.2** Flowchart for the proposed IPSO
For each particle
Current value = new p best
Choose the minimum ‘F’ of all particles as the g best
Calculate particle velocity using (2.1)
Calculate particle position using (2.2)
Update memory of each particle
End
End
Return by using IPSO
Stop

Figure 2.2 (Continued)
2.3 BASIC CONCEPTS OF TASK SCHEDULING

Scheduling is the allocation of ideal jobs to resources at particular times which minimizes the total length of the schedule. The intention of scheduling is to distribute the tasks among various processors in such a way that the total execution time or the entire schedule length or makespan is minimized. The execution time depends on the allocation of tasks and the scheduling policy applied to tasks. Total Finishing Time (TFT) and Average Waiting Time (AWT) are two computable criteria to minimize the makespan in multiprocessor architecture. Total finishing time is defined as the maximum of each processor’s finishing time which is the time that the processor finishes its job. Waiting Time is defined as the average of time that each job waits in a ready queue. Simultaneously minimizing these two criteria is a Multi Objective Optimization (MOO) Problem. The scheme of job scheduling is shown in Figure 2.3.

![Figure 2.3 Schematic of job scheduling](image)

A Task/job scheduling policy uses the information associated with requests to decide which request should be serviced next. All requests waiting to be serviced are kept in a list of pending requests. Whenever scheduling is to be performed, the scheduler examines the pending requests and selects one for servicing. This request is handled over to the server. A request leaves the server, when it completes or when it is pre-empted by the scheduler, in which
case it is put back into the list of pending requests. In either situation, the scheduler performs scheduling to select the next request to be serviced. The scheduler records the information concerning each job in its data structure and maintains it all through the life of the request in the system.

2.4 BASIC CONCEPTS OF MULTI OBJECTIVE OPTIMIZATION

The main objective of Multi Objective Optimization (MOO) algorithms is to find a set of solutions which optimally balances the trade-offs among the objectives of a Multi Objective Problem (MOP). The task is to find a set of non-dominated solutions, referred to as the Pareto-optimal set. In the following, domination and Pareto-optimal, set concepts will be described, as follows, (Moattar et al 2008).

Let \( P \subseteq Q^m \) denote the \( m \times \) –dimensional search space. The search space, \( P \) is also referred to as the decision space and \( F \subseteq P \) the feasible space. With no constraints, the feasible space is the same as the search space.

Let \( X = (x_1, x_2, ..., x_m) \) referred to as a decision vector. A single objective function, \( f_n(X) \) is defined as \( f_n : Q^m \rightarrow Q \). Let \( f(X) = (f_1(X), f_2(X), ..., f_m(X)) \in O \subseteq Q^m \) be an objective vector containing \( m_n \) objective function evaluations, \( O \) is referred as the objective space.

**Domination:** A decision vector, \( X_1 \) dominates a decision vector, \( X_2 \) (denoted by \( X_1 \prec X_2 \)) if and only if,

\[ f_n(X_1) \leq f_n(X_2) \forall n = 1, ..., m_n \] and

\[ X_1 \text{ is not worse than } X_2 \text{ in all objectives, i.e.} \]

\( f_n(X_1) \leq f_n(X_2) \forall n = 1, ..., m_n \) and
- $X_1$ is strictly better than $X_2$ in at least one objective, i.e.
  \[ \exists n = 1, \ldots, m_n : f_n(X_1) < f_n(X_2) \]

Similarly, an objective vector, $f_1$, dominates another objective vector $f_2$, if $f_1$ is not worse than $f_2$ in all objective values and $f_1$ is better than $f_2$ in at least one of the objective values. Objective vector dominance is denoted by $f_1 \preceq f_2$.

**Pareto-optimal:** A decision vector $X^* \in F$ is pareto optimal if there does not exist a decision vector, $X \neq X^* \in F$ that dominates it. That is $\nexists n : f_n(X) < f_n(X^*)$. An objective vector $f^*(X)$ is pareto optimal, if $X$ is pareto-optimal.

**Pareto-optimal set:** The set of all pareto optimal decision vectors form the pareto-optimal set $P^*$ that is
\[ P^* = \{ X^* \in F | \nexists X \in F : X \prec X^* \} \]

One of the approaches to deal with multi-objective problems is to define an aggregate objective function as a weighted sum of objectives and is redefined as,

\[
\text{Minimize } \sum_{n=1}^{m_n} \omega_n f_n(x) \quad (2.3)
\]

Subject to,
\[
g_t(X) \leq 0, \quad t = 1, \ldots, m_g
\]
\[
h_t(X) = 0, \quad t = m_g + 1, \ldots, m_g + m_h
\]
\[
X = [X_{\min}^m, X_{\max}^m]
\]
\[
\omega_n \geq 0, \quad n = 1, \ldots, m_n
\]
where,

\[ g_t \text{ and } h_t \text{ are the inequality and equality constraints and} \]

\[ [X_{\min}, X_{\max}] \text{ represents the boundary constraints} \]

2.5 SIMULATION PROCEDURE

The details of the simulation carried out for the multiprocessor task scheduling problem with the datasets are as follows,

Benchmark datasets are taken from EricTailard’s site for dynamic task scheduling. Two datasets are taken for simulation. Dataset 1 involves 50 tasks and 20 processors. Dataset 2 involves 100 tasks with 20 processors. The data for the static scheduling are randomly generated such as, 2 processors with 20 tasks, 3 processors with 20 tasks, 3 processors with 40 tasks, 4 processors with 30 tasks, 4 processors with 50 tasks, 5 processors with 45 tasks and 5 processors with 60 tasks.

To demonstrate the effectiveness of the proposed approach, the approach is run with 30 independent trials with different values of random seeds and control parameters.

The optimal result is obtained for following parameter settings.

**Proposed IPSO:**

- Max. Iteration : 500
- Swarm size : Twice the number of tasks (Salman et al 2002)
- \( \omega_{\min} - \omega_{\max} \) : 0.5
- \( C_{lb}, C_{lg} \text{ and } C2 \) : 2
The proposed approach IPSO is developed using MATLAB R2009 and executed in a PC with Intel core i3 processor with 3 GB RAM and 2.13 GHz speed.

2.6 STATIC TASK SCHEDULING

Let \( T = \{1, 2...n\} \) be a given set of independent tasks, and \( P = \{1, 2...m\} \) represents the set of identical processors. All tasks are mutually independent and each task can be scheduled on any processor. All the tasks should be scheduled in such a way that one processor executes one task at a time.

Static task scheduling requires prior information about the number of tasks, number of processors, execution time of tasks, and scheduling done before the execution. The objective of scheduling is to simultaneously minimize the total finishing time and the average waiting time of the schedule. Minimization of these two objectives simultaneously makes the scheduling problem a multi objective optimization problem.

The considered scheduling problem is formulated by,

\[
fin(P_j) = \sum_{i=1}^{n} exe_i
\]

(2.4)

Min \( TFT_{schedule} = \max fin(P_j) \) \( 1 \leq j \leq m \)

(2.5)

\( WT_i = Arrival_i + fin_{i-1} \)

(2.6)

\[
AWT_{schedule} = \frac{\sum_{j=1}^{m} \sum_{i=1}^{n} WT_i}{n}
\]

(2.7)

where ‘n’ is the number of tasks, and ‘m’ is the number of processors, the variable ‘i’ refers tasks from 1 to n and variable ‘j’ refers processors from 1 to m.
\( \text{fin}(P_j) \) refers the finishing time of processor \( j \)

\( \text{exe}_i \) represents execution time of task \( i \)

\( \text{TFT}_{\text{schedule}} \) denotes the Total Finishing Time i.e., maximum of each processor’s finishing time

\( WT_i \) refers the waiting time of task \( i \) and

\( AWT_i \) is the average waiting time of ‘\( n \)’ tasks

The considered Multi objective problem is unified by using weighted sum of objectives converted into single objective optimization problem (Moattar et al 2008) and is represented by,

\[
\text{Min } f = \text{TFT}_{\text{schedule}} + \beta \text{AWT}_{\text{schedule}}
\]  

(2.8)

where \( \beta \) is the weight coefficient, which can be adjusted to compromise between two minimization objectives. The first term in Equation (2.8) is minimizing the Total finishing time, that is the makespan of the entire schedule and the second term is related to minimization of the waiting time.

The global minimization problem is changed into global maximization problem. Through the transforming Equation (2.8) the proper fitness function for the multiobjective problem is represented by,

\[
F = \begin{cases} 
V - \text{TFT}_{\text{schedule}} - \beta \text{AWT}_{\text{schedule}} & f < V \\
0 & f \geq V
\end{cases}
\]  

(2.9)

where \( V \) should select an appropriate positive number to ensure that the fitness of all good individuals are positive in the feasible solution space.
2.6.1 Results and Discussion

The proposed IPSO approach is applied to the static independent tasks. The obtained results are depicted in Table 2.1. The results reveal that the proposed approach IPSO simultaneously reduces the total finishing time and the average waiting time.

Table 2.1 Total finishing time and average waiting time using IPSO

<table>
<thead>
<tr>
<th>No of Processors</th>
<th>No of jobs</th>
<th>Proposed IPSO</th>
<th>Total Finishing Time (TFT)</th>
<th>Average Waiting Time (AWT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td></td>
<td>57.34</td>
<td>29.12</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
<td>54.01</td>
<td>45.00</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td></td>
<td>69.04</td>
<td>41.03</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td></td>
<td>70.97</td>
<td>29.74</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td></td>
<td>70.62</td>
<td>30.06</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
<td></td>
<td>68.04</td>
<td>33.65</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td></td>
<td>72.31</td>
<td>36.56</td>
</tr>
</tbody>
</table>

The proposed approach IPSO produces total finishing time as 69.04s and average waiting time as 41.03s for 3 processors and 40 tasks. For the dataset 4 processors with 30 tasks, the achieved result for total finishing time is 70.97s and average waiting time is 29.74s. For the dataset 5 processors with 60 tasks, the result achieved for total finishing time is 72.31s and average waiting time is 36.56s.

2.6.2 Performance Comparison

In order to validate the performance of the proposed algorithm IPSO, comparisons have been made with the existing algorithms such as LPT, SPT GA and standard PSO for the same datasets and are reported in Table 2.2.
Table 2.2  Comparison of total finishing time and average waiting time of jobs using LPT, SPT, GA, PSO and IPSO

<table>
<thead>
<tr>
<th>No. of Processors</th>
<th>No. of jobs</th>
<th>LPT AWT</th>
<th>LPT TFT</th>
<th>SPT AWT</th>
<th>SPT TFT</th>
<th>GA AWT</th>
<th>GA TFT</th>
<th>PSO AWT</th>
<th>PSO TFT</th>
<th>Proposed IPSO AWT</th>
<th>Proposed IPSO TFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>52.4</td>
<td>60.9</td>
<td>30.21</td>
<td>70.41</td>
<td>31.38</td>
<td>61.80</td>
<td>30.10</td>
<td>60.52</td>
<td>29.12</td>
<td>57.34</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>47.1</td>
<td>56.7</td>
<td>38.31</td>
<td>69.56</td>
<td>47.01</td>
<td>57.23</td>
<td>45.92</td>
<td>56.49</td>
<td>45.00</td>
<td>54.01</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>56.5</td>
<td>70.9</td>
<td>44.96</td>
<td>80.21</td>
<td>44.31</td>
<td>70.21</td>
<td>42.09</td>
<td>70.01</td>
<td>41.03</td>
<td>69.04</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>48.2</td>
<td>62.7</td>
<td>32.64</td>
<td>75.36</td>
<td>32.91</td>
<td>74.26</td>
<td>30.65</td>
<td>72.18</td>
<td>29.74</td>
<td>70.97</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>50.7</td>
<td>66.2</td>
<td>38.91</td>
<td>79.27</td>
<td>35.72</td>
<td>76.21</td>
<td>32.79</td>
<td>71.20</td>
<td>30.06</td>
<td>70.62</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
<td>46.12</td>
<td>61.72</td>
<td>36.27</td>
<td>80.72</td>
<td>38.03</td>
<td>72.65</td>
<td>34.91</td>
<td>70.09</td>
<td>33.65</td>
<td>68.04</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>51.87</td>
<td>75.96</td>
<td>40.72</td>
<td>84.76</td>
<td>42.93</td>
<td>77.29</td>
<td>39.61</td>
<td>75.42</td>
<td>36.56</td>
<td>72.31</td>
</tr>
</tbody>
</table>

For the dataset 4 processors with 30 tasks, the Longest Processing Time algorithm (LPT) produces the total finishing Time and average waiting time as 62.7s and 48.2s. Shortest Processing Time (SPT) algorithm produces 75.36s as the total finishing time and 32.64s as average waiting time for the same dataset of 4 processors and 30 tasks. Genetic Algorithm (GA) produces 32.91s and 74.26s as average waiting time and total finishing time respectively for the same dataset. Standard PSO produces total finishing time as 72.18s and the average waiting time as 30.65s. The proposed approach IPSO produces the average waiting time as 29.74s and the total finishing time 70.97s.

LPT reduces the total finishing time whereas the average waiting time is drastically high and in SPT the total finishing time is drastically high and reduces the average waiting time. Genetic Algorithm reduces the total finishing time and average waiting time simultaneously when compared with LPT and SPT algorithms. Standard PSO also simultaneously reduces the total finishing time and the average waiting time than GA. Further, the introduction of the proposed approach IPSO reduces both the total finishing time and average waiting time simultaneously. These results prove that the proposed approach IPSO produces comparatively better results.
The variations in total finishing time and waiting time using conventional methodologies LPT, SPT and evolutionary approaches, GA, PSO and IPSO are shown in Figures 2.4 to 2.10.

**Figure 2.4**  Total finishing time and average waiting time for 2 processors with 20 jobs using LPT, SPT, GA, PSO and IPSO

**Figure 2.5**  Total finishing time and average waiting time for 3 processors with 20 jobs using LPT, SPT, GA, PSO and IPSO
Figure 2.6 Total finishing time and average waiting time for 3 processors with 40 jobs using LPT, SPT, GA, PSO and IPSO

Figure 2.7 Total finishing time and average waiting time for 4 processors with 30 using LPT, SPT, GA, PSO and IPSO
Figure 2.8  Total finishing time and average waiting time for 4 processors with 50 jobs using LPT, SPT, GA, PSO and IPSO.

Figure 2.9  Total finishing time and average waiting time for 5 processors with 45 jobs Using LPT, SPT, GA,PSO and IPSO.
Thus, it is observed from the results that the proposed algorithm IPSO is better than the conventional methodologies LPT, SPT and the other optimization techniques, such as GA, standard PSO.

### 2.7 DYNAMIC TASK SCHEDULING

The present research examines Dynamic Task Scheduling with the following scenario. The processors in the system are distributed computing environment and the tasks are scheduled as when they arrive in the queue. Load balancing of dynamic tasks is particularly useful in a system, where processor utilization is the major issue, instead of minimizing execution time of the applications.

The objective is to minimize the total execution cost encountered by the assignment of all tasks allocated to the processors. A particle is evaluated by calculating its fitness function. Fitness function indicates the goodness of the schedule. The cost function in case of optimality determination computes
the best possible solution. This cost function (for ex. best cost, worst cost) will be the solution for the objective function considered. As a result, it does not infer to the regular cost value, which has unit Rupees or dollars. Since the terminology used here refers to optimal best cost, worst cost and average cost, it does not possess an unit of its own.

Dynamic task scheduling can be done under two cases namely,

i. Considering only the arrival time of the tasks (without load balancing)

ii. Considering both the time of arrival of each task and efficient processor utilization. (with load balancing)

Fitness function to minimise the total execution time, that means schedule length varies for the above said two cases, hence is dealt individually in the following sub-sections.

2.7.1 Dynamic Task Scheduling without Load Balancing

Dynamic Task Scheduling is adopted to schedule the tasks, in the way in which they are arriving. Dynamic load balancing does not require the prior knowledge of the tasks as like static. That is, it need not be aware of the run-time behaviour of the applications before execution. In Dynamic Scheduling, tasks are redistributed among processors during the execution time. This redistribution is done by transferring tasks from heavily loaded processors to the lightly loaded processors (called load balancing), to improve the processor utilization and performance.

To minimise the makespan of the schedule, the equation is formulated and represented as,
\[ TFT(P_j) = arrival_i + \sum_{i=1}^{n} fin_i \]  \hspace{1cm} (2.10)

\[
fin_i = \begin{cases} 
exe_i & \text{if } arrival_i \leq fin_{i-1} \\
arrival_i + exe_i & \text{otherwise}
\end{cases}
\hspace{1cm} (2.11)

Fitness function is formulated to minimise the average of Total Finishing Time.

\[
F(X) = \min \left[ \frac{\sum_{j=1}^{m} TFT(P_j)}{m} \right] \hspace{1cm} (2.12)
\]

where \( 'n' \) is the number of tasks

\( 'm' \) is the number of processors

\( TFT(P_j) \) = Total Finishing Time of Processor \( j \)

\( arrival_i \) = arrival time of task \( i \)

\( fin_i \) = finishing time of task \( i \) and

\( exe_i \) = execution time of task \( i \)

2.7.1.1 Results and Discussion

The results obtained for the proposed algorithm IPSO have been tabulated and is shown in Table 2.3.

The Table 2.3 given below shows the details of the obtained best, worst and average cost. GA produces the best cost as 3018 for 50 tasks for the
dataset 1 and it is 5928 for dataset 2. PSO obtains 2972 as best cost for dataset 1 and it is 5552 for dataset 2. The proposed IPSO approach achieves best cost for dataset 1, as 2374 and 4527 for dataset 2.

From the results, the best cost, worst and average costs are better in the proposed IPSO when compared to the other approaches. The convergence time for the proposed methodology for dataset 1 is 4.0521s and 5.7112s for dataset 2.

Table 2.3  Comparisons of best cost, worst cost, average cost and convergence time using GA, PSO, and proposed IPSO for dynamic task Scheduling without load balancing

<table>
<thead>
<tr>
<th>Methods</th>
<th>GA</th>
<th>PSO</th>
<th>Proposed IPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of tasks</td>
<td>50</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Best cost</td>
<td>3018</td>
<td>5928</td>
<td>2972</td>
</tr>
<tr>
<td>Worst cost</td>
<td>4186</td>
<td>6582</td>
<td>3724</td>
</tr>
<tr>
<td>Average cost</td>
<td>3492.8</td>
<td>6393.6</td>
<td>3187.5</td>
</tr>
<tr>
<td>Convergence time in seconds</td>
<td>6.8412</td>
<td>9.99226</td>
<td>3.9774</td>
</tr>
</tbody>
</table>

The results from Table 2.3 infer that the IPSO performs better than the other algorithms.

The best cost comparison for dataset 1 and 2 using GA, standard PSO and IPSO are shown in Figures 2.11 and 2.12.
2.7.1.2 Performance Comparison

The performance of the proposed IPSO approach is compared with the previously proposed (Visalakshi and Sivanandam 2009) variant PSO methods namely, PSO with Fixed (PSO-FI) and PSO with Variable Inertia (PSO-VI) for multiprocessor dynamic task scheduling with same datasets.
Table 2.4 Performance comparison of various PSO based approaches

<table>
<thead>
<tr>
<th>Methods</th>
<th>PSO-FI (Visalakshi and Sivanandam 2009)</th>
<th>PSO-VI (Visalakshi and Sivanandam 2009)</th>
<th>Proposed IPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tasks</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Best cost</td>
<td>2814</td>
<td>2592</td>
<td>2374</td>
</tr>
<tr>
<td></td>
<td>5448</td>
<td>4893</td>
<td>4527</td>
</tr>
<tr>
<td>Worst cost</td>
<td>3586</td>
<td>3428</td>
<td>3136</td>
</tr>
<tr>
<td></td>
<td>5739</td>
<td>5867</td>
<td>5213</td>
</tr>
<tr>
<td>Average cost</td>
<td>3200</td>
<td>3010</td>
<td>2755</td>
</tr>
<tr>
<td></td>
<td>5593.5</td>
<td>5380</td>
<td>4870</td>
</tr>
<tr>
<td>Convergence time in seconds</td>
<td>4.0721</td>
<td>4.3162</td>
<td>4.0521</td>
</tr>
<tr>
<td></td>
<td>5.8221</td>
<td>5.9828</td>
<td>5.7112</td>
</tr>
</tbody>
</table>

The proposed IPSO approach gives the best cost for dataset 1 as 2374 and for dataset 2, it is 4527. PSO-FI produces the best cost for dataset 1 as 2814 and for dataset 2, it is 5448. PSO-VI produces 2592 as the best cost for dataset 1 and it is 4893 for dataset 2. The convergence time for the proposed approach is 4.0521s for dataset 1 and 5.7112 for dataset 2.

The above mentioned comparison reveals that the proposed method IPSO yields better results with faster (0.2 to 0.3s) convergence compared with PSO-FI and PSO-VI. The result infers that the inclusion of the bad experience particles plays a major role for the improvement of the achievement.

2.7.2 Dynamic Task Scheduling with Load Balancing

The drawback in dynamic scheduling without load balancing is that some of the processors may be lightly loaded and others are heavily loaded. To overcome the drawback and also to improve the processor utilization and performance, load balancing is considered. To achieve load balancing in dynamic task scheduling, tasks have to be allocated in such a way that they
are arriving. Our main goal is to minimise the makespan of the entire schedule and to improve the processor utilization.

To achieve better load balancing, tasks have to be assigned to the processors in such a way to reduce the makespan and to increase the utilization. Hence, equations are formulated and represented as follows, (Zomaya et al 1999, Zomaya and The 2001, (Visalakshi and Sivanandam 2009).

\[
TAQ(P_j) = arrival(t_1) + \left(\frac{1}{\text{max span}}\right) \times \text{Avg.Utilization} \tag{2.13}
\]

\[
\text{Utilization}(P_j) = \frac{CT(P_j)}{\text{max span}} \tag{2.14}
\]

\[
\text{Avg.Utilization} = \frac{\sum_{j=1}^{m} \text{Utilization}(P_j)}{m} \tag{2.15}
\]

where \(TAQ(P_j)\) is used to evaluate the Quality of task assignment in a processor.

\(arrival(t_1)\) is the arrival time of task 1

\(\text{max span}\) is the total finishing time of the schedule

\(CT(P_j)\) is the Completion Time of processor j

\(\text{Avg.Utilization}\) is the sum of all processors utilization divided by the total number of processors and

\(m\) is the number the processors

Effective utilization of processors support the concept of load balancing. If all the processors are used to their maximum, the loads, which
are the measures of idleness of processors are effectively reduced. Hence, the fitness function calculates the average of the total execution time of the set of tasks allocated to the processors as,

$$F(X) = \max \left\{ \frac{\sum_{j=1}^{m} T_{AQ}(P_j)}{m} \right\}$$  \hspace{1cm} (2.16)$$

2.7.2.1 Results and Discussion

The results obtained using the proposed method IPSO have been compared with GA and standard PSO and shown in Table 2.5.

Table 2.5 Comparisons of best cost, worst cost, average cost and convergence time using GA, PSO, and proposed IPSO for dynamic task scheduling with load balancing

<table>
<thead>
<tr>
<th>Method</th>
<th>GA</th>
<th>PSO</th>
<th>Proposed IPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of tasks</strong></td>
<td>50</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td><strong>Worst Cost</strong></td>
<td>8.8782</td>
<td>17.4521</td>
<td>9.0012</td>
</tr>
<tr>
<td><strong>Average Cost</strong></td>
<td>9.1326</td>
<td>17.0484</td>
<td>9.9116</td>
</tr>
<tr>
<td><strong>ConvergenceTime</strong></td>
<td>8.643</td>
<td>11.8425</td>
<td>5.0603</td>
</tr>
</tbody>
</table>

The achieved results best cost, worst and average cost values using the proposed approach IPSO are compared with GA and standard PSO. The best cost is the best fitness value achieved. The proposed approach IPSO produces the best cost as 12.0042 for dataset 1 and it is 21.4291 for dataset 2. The average cost produced by IPSO is 11.4931 for dataset 1 and 20.3197 for dataset 2.
Thus the result infers that the proposed IPSO produces the best results when compared with GA and standard PSO. The convergence time is slightly increased (0.02 to 0.05s) using IPSO than with the standard PSO.

The best cost comparison for dataset 1 and for dataset 2 are shown in Figures 2.13 and 2.14 using GA, PSO and IPSO.

**Figure 2.13** Best cost for 50 tasks and 20 processors using GA, PSO and IPSO

**Figure 2.14** Best cost 100 tasks and 20 processors using GA, PSO and IPSO
2.7.2.2 Performance Comparison

The performance of the proposed IPSO approach is compared with that of the previously proposed (Visalakshi and Sivanandam 2009) variant PSO methods namely PSO with Fixed (PSO-FI) and Variable Inertia (PSO-VI) for the same datasets and for the same dynamic task scheduling problem.

Table 2.6 Performance comparisons of various PSO based approaches

<table>
<thead>
<tr>
<th>Method</th>
<th>PSO-FI (Visalakshi and Sivanandam 2009)</th>
<th>PSO-VI (Visalakshi and Sivanandam 2009)</th>
<th>Proposed IPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tasks</td>
<td>50</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Best cost</td>
<td>11.224</td>
<td>11.584</td>
<td>12.0042</td>
</tr>
<tr>
<td>Worst cost</td>
<td>9.652</td>
<td>10.112</td>
<td>10.9820</td>
</tr>
<tr>
<td>Average cost</td>
<td>10.438</td>
<td>10.848</td>
<td>11.4931</td>
</tr>
<tr>
<td>Convergence time in seconds</td>
<td>5.1274</td>
<td>5.1382</td>
<td>5.1176</td>
</tr>
</tbody>
</table>

The proposed approach IPSO produces the best cost for dataset 1 as 12.0042 and for dataset 2, it is 21.4291. PSO-FI produces the best cost for dataset 1 as 11.224 and for dataset 2, it is 19.926. PSO-VI produces the best cost for dataset 1 as 11.584 and for dataset 2, it is 20.728. The convergence time for the proposed method for dataset 1, is 5.1176s and for dataset 2 is 6.9064 s. The convergence time for dataset 1and dataset 2 using PSO-FI are 5.1274s and 6.9016s respectively. The convergence time for dataset 1and dataset 2 using PSO-VI are 5.1382s and 6.9101s respectively. This comparison reveals that the proposed approach IPSO achieves better results faster (0.003 to 0.02 times) than PSO-FI and PSO-VI. Thus, the result infers that the inclusion of the bad experience component plays a major role in the
improvement and achievement of a near optimal solution when applied to the task assignment problem with dynamic tasks.

2.8 CONCLUSION

The chapter two brought out a detailed study of the proposed IPSO. The concept of the proposed IPSO is presented to solve the multiprocessor task scheduling with two cases, namely, Static Scheduling with independent tasks and Dynamic Task Scheduling with and without load balancing, to reduce the makespan of the schedule.

The proposed method IPSO is validated, for Static Scheduling by comparing with the traditional algorithms such as Longest Processing Time algorithm (LPT), Shortest Processing Time (SPT), Genetic Algorithm (GA) and standard PSO. For the dataset, 4 processors with 30 tasks, LPT produces total finishing Time and average waiting time as 62.7s and 48.2s respectively. SPT produces 32.64s as average waiting time and 75.36s as total finishing time. GA produces 32.91s and 74.26s as average waiting time and total finishing time respectively. Standard PSO produces the total finishing time as 72.18s and average waiting time as 30.65s, and the proposed approach IPSO produces total finishing time as 70.97s and average waiting time as 29.74s.

The proposed method IPSO is validated for Dynamic Task Scheduling by comparing with the previously proposed variant PSO methods namely PSO-FI and PSO-VI for the same datasets. The proposed IPSO approach for dynamic task scheduling without load balancing gives the best cost for dataset 1, as 2374 and for dataset 2, it is 4527. PSO-FI produces best cost for dataset 1, as 2814 and for dataset 2, it is 5448. PSO-VI produces 2592 as best cost for dataset 1 and 4893 for dataset 2. The convergence time for the proposed approach is 4.0521s for dataset 1 and 5.7112s for dataset 2 and the proposed approach IPSO is 0.2s faster than the PSO-VI.
For Dynamic Task Scheduling with load balancing, the proposed approach IPSO produces the best cost as 12.0042 for dataset 1 and for dataset 2, it is 21.4291. PSO-FI produces the best cost for dataset 1 as 11.224 and for dataset 2, it is 19.926. PSO-VI produces the best cost for dataset 1 as 11.584 and for dataset 2, it is 20.728. The convergence time for the proposed method IPSO for dataset 1 is 5.1176s and for dataset 2 is 6.9064s. The convergence time for dataset 1 and dataset 2 using PSO-FI are 5.1274s and 6.9016s respectively. The convergence time for dataset 1 and dataset 2 using PSO-VI are 5.1382s and 6.9101s respectively. The proposed approach IPSO is 0.02 times faster than PSO-VI.

Thus, the simulation results show that the proposed IPSO works well and produces better results for static and dynamic task scheduling problem in a multiprocessor system. Further, to improve the performance of IPSO approach, a hybrid approach IPSO with SA is proposed in the next chapter.