# 7 PROPOSED PSO BASED SCHEDULING ALGORITHM

Different static (e.g., RM, SJF) and dynamic (e.g., EDF, LST) scheduling algorithms have been evaluated and compared for Soft Real-Time Systems. The performance of Static and Dynamic scheduling algorithms vary based on the CPU load. It has been observed that EDF and LST perform well in the underload scenario where CPU load is less than or equal to one. Even EDF is one of the optimal scheduling algorithms, and it makes sure that in the underload scenario, all task will meet their deadline. Static algorithms like RM and SJF also perform well in the underload scenario. Still, in some instances, it has been observed that there is possible to schedule all task by EDF, but static algorithms fail to schedule those tasks, and few tasks missed their deadline. In a slightly overload situation, when CPU load is higher than one at that time, the performance of the dynamic scheduling algorithm degrades very fast. In contrast, static scheduling algorithms can still schedule a few tasks and meet their deadline. Thus, the Dynamic Priority scheduling algorithm performs well in the underload scenario, and the Static Priority scheduling algorithm performs decently in the overload scenario [20].

The Dynamic Priority scheduling algorithms are more responsive to the average cases, but their worst-case real-time performance may be more unsatisfactory than the Static Priority scheduling algorithm. Still, there is no single priority scheduling algorithm exist which perform well in underload and overload scenario. Hybrid priority driven scheduling algorithms have been developed, which use characteristics of both types of algorithms, like D_EDF and S_LST, which is using features of dynamic scheduling algorithm during underload scenarios and static scheduling algorithms during overload scenarios [24][53][60]. The problem with this kind of hybrid algorithm is scheduling algorithm needs to keep checking with the status of CPU load, and based on that; it will assign the priority to the task. It has been proposed a technique for scheduling tasks using Artificial Intelligence and Swarm techniques. Swarm Intelligence is the study of computational systems inspired by collective intelligence. Collective Intelligence emerges through the cooperation of large numbers of homogeneous agents in the environment. Examples include schools of fish, flocks of birds, and colonies of ants. Such intelligence is decentralized, self-organizing, and distributed throughout an environment. In nature, such systems are commonly used to solve problems such as effective foraging for food, prey evading, or colony re-location. The information is typically stored throughout the participating homogeneous agents or stored or communicated in the environment itself, such as through pheromones in ants, dancing in bees, and proximity in fish and birds. Using Swarm Intelligence, it is possible to find optimal solutions for problems like scheduling of the task [22][23]. Previously, ACO-based scheduling algorithm has been proposed, and it has been shown that the swarm intelligence-based scheduling algorithm performs equally well like a dynamic scheduling algorithm. Moreover, it gives better performance in an overload scenario as well [33][52].

A study of various swarm intelligence techniques have been done and it has been identified that Particle Swarm Optimization is a strong candidate solution for scheduling in Soft Real-Time systems. Particle Swarm Optimization investigates probabilistic algorithms inspired by flocking, schooling, and herding. Like evolutionary computation, swarm intelligence algorithms or strategies are considered adaptive strategies and are typically applied to search and optimization domains [41]. This research has used PSO (Particle Swarm Optimization) based swarm technique to design the new scheduling approach. It considered each task as a particle and applied a modified PSO technique to identify the most critical task to execute. The efficiency of the newly proposed method has been compared with existing EDF and ACO based scheduling algorithms considering two significant parameters, the SR (Success Ratio) and the ECU (Effective CPU Utilization). All three algorithms have been tested on the simulator with a Soft Real-time periodic task set on 500 timelines. It has been observed that during the underload scenario, the proposed algorithm performs equally to EDF and ACO based algorithms. However, during overload and highly overload situations, the proposed algorithm performs better compared to EDF and ACO based algorithms.

## 7.1 Swarm Intelligence Techniques

Different Swarm Intelligence Techniques have been studied to identify the best fitted for problem statements. It is an emerging field of biologically-inspired artificial intelligence based on the behavioral models of social insects such as ants, bees, wasps, termites, etc. The following are the few well-applied Swarm Techniques [61].

1) **PSO -** Particle Swarm Optimization belongs to Swarm Intelligence and Collective Intelligence and is a sub-field of Computational Intelligence. PSO is inspired by the social foraging behavior of some animals, such as the flocking behavior of birds and the schooling behavior of fish. Particles in the swarm fly through an environment, following the fitter members of the swarm and generally biasing their movement toward historically good areas of their environment. The algorithm's goal is to have all the particles locate the optima in a multi-dimensional hyper-volume [62].

2) **ACO -** It is inspired by the pheromone communication of the blind ants regarding the right path between the colony and the food source in an environment. The probability of the ant following a particular route is not only a function of pheromone intensity but also a function of distance to that city, the function known as visibility. The objective of the strategy is to exploit historical, i.e., pheromone-based and heuristic information to construct candidate solutions each in a probabilistic step-wise manner and fold the information learned from constructing solutions into the history. The probability of selecting a component is determined by the heuristic contribution of the component to the overall cost of the solution, and the quality of solution and history is updated proportionally to the quality of the best-known solution [56].

3) **Bees -** It is inspired by the foraging behavior of honey bees. The hive sends out the Scout bees when locating nectar (a sugary fluid secreted within flowers), return to the hive, and communicate the other bees the fitness, quality, distance, and direction of the food source via waggle dance. The objective of the algorithm is to locate and explore good sites within a problem search space. Many scout bees are sent out; each iteration is always in search of additional good sites that are continually exploited in the local search application [63].

4) **GSA** - Gravitational search algorithm (GSA) is a newly developed stochastic search algorithm based on the Newtonian gravity- "Every particle in the universe attracts every other particle with a force that is directly proportional to the product of their masses and

inversely proportional to the square of the distance between them" and the mass interactions. In this approach, the search agents are a collection of masses that interact with each other based on the Newtonian gravity and the laws of motion in which all of the objects attract each other by the gravity force, while this force causes a global movement of all objects towards the objects with heavier masses. Thus, the heavy masses correspond to good solutions to the problem [64].

During the literature study, it has been observed that Particle Swarm Optimization has been widely used in scheduling for the Cloud Computing environment. A. S. Ajeena Beegom and M. S. Rajasree proposed the Integer-PSO algorithm for task scheduling in a cloud computing system in 2019 [35]. A two-level particle swarm optimization algorithm was created for the flexible job-shop scheduling problem [36], and PSO based scheduling was also applied in workflow applications in cloud computing environments [37]. However, the PSO-based scheduling approach has not been explored with the Real-Time Operating system. In this research, PSO-based Swarm Intelligence Techniques has been considered to resolve the problem statement. This approach has been selected because it is highly recommended for scheduling problems and it is the right approach when the problem size is between 20 to 40 [40][41].

## 7.2 Particle Swarm Optimization Technique

The Particle Swarm Optimization algorithm comprises a collection of particles that move around the search space influenced by their own best past location and the best past location of the whole swarm or a close neighbour. Each iteration, a particle's velocity is updated using following equation 12 [41][65][66][67].

$$v_{i,d}(t+1) = v_{i,d}(t) + \left(c_1 \times r_1 \times \left(p_{i,d}^{best} - p_{i,d}(t)\right)\right) + \left(c_2 \times r_2 \times \left(p_{gbest,d} - p_{i,d}(t)\right)\right) \quad (12)$$

$$p_{i,d}(t+1) = p_{i,d}(t) + v_{i,d}(t+1) \qquad (13)$$

where,

- $v_{i,d}(t+1)$ and $v_{i,d}(t)$ represent the current and previous velocity in the $d^{th}$ dimension of particle $i$, respectively.
- $c_1$ and $c_2$ are acceleration coefficient for the personal best and global best positions, respectively.
- $p_{i,d}(t+1)$ and $p_{i,d}(t)$ are the current and previous position of particle $i$.
- $p_{i,d}^{best}$ and $p_{gbest,d}$ are the best position found by particle $i$ so far and the best position found by the whole swarm so far, respectively
- $r_1$ $and$ $r_2$ are the randomly generated numbers in the range of [0, 1].
- $d \in D$ is the dimension $d$ in the search space.

Variants on this update equation consider the best positions within a particle's local neighbourhood at time t. A particle's position is updated using equation 13 [41].

Figure 7.1 shows the graphical representation of the Particle Swarm Optimization. After each iteration, the particle moves in a new direction, and most of the time, it is optimal, and that decision will be based on the personal best position and global best position.

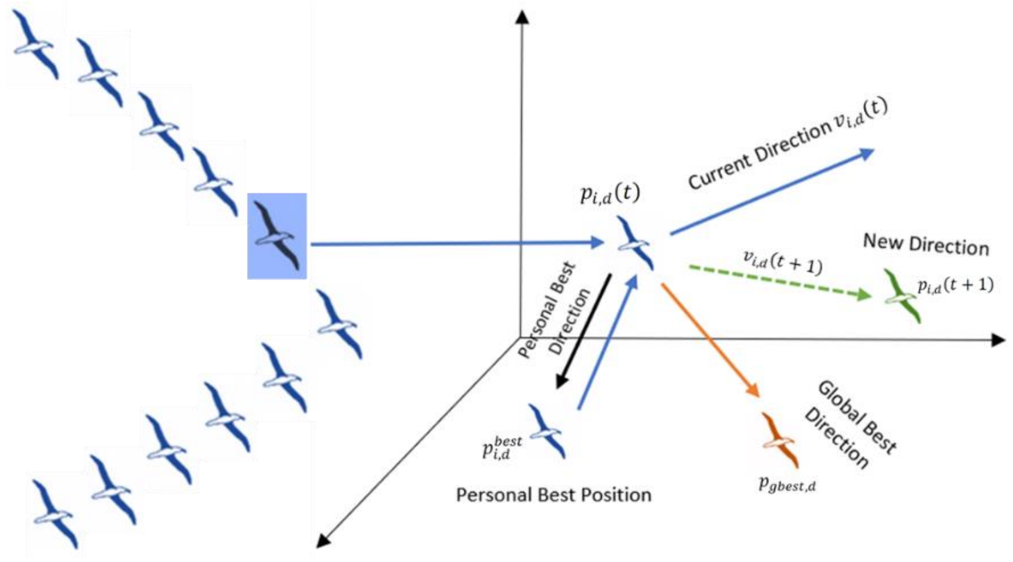

Figure 7.1 – Graphical representation of PSO

Heuristics for this approach are [40][41][65]:

- The number of particles should be low, around 20-40,
- The speed a particle can move should be bounded,
- The learning factors should be between 0 and 4, typically 2.0,
- Particles may leave the problem space's boundary and maybe penalise, be reflected in the domain, or be biased to return toward a position in the problem domain. Alternatively, a wrapping strategy may be used at the edge of the domain, creating a loop or related geometrical structures at the chosen dimensionality.
- An inertia or momentum coefficient can be introduced to limit the change in velocity.

## 7.3 PSO Based Scheduling Algorithm

Selecting the scheduling algorithm for Soft Real-Time System is a crucial decision, as discussed previously. This research introduced the scheduling algorithm, which is based on PSO techniques. The algorithm considering each given task as a particle, and all tasks which are eligible for scheduling are viewed as a set of particles. The ultimate goal of the scheduling algorithm is to choose a task at a given point in time in such a way that the task can meet its deadline [37]. In the Soft Real-Time system, it is intended to ensure that all tasks will meet their deadline in the underload condition, and the maximum tasks will meet their deadline in the overload scenario.

The scheduling algorithm executes when a new task arrives, or the currently performing task is completed. When more than one task is ready to run, the scheduling algorithm needs to select the task effectively. This thesis has proposed the PSO based scheduling algorithm, which has the following significant steps.

Step 1: Initialization of Task as a Particle

Step 2: Compute the velocity and position of each task

Step 3: Analyze the position and velocity of each task

Step 4: Selection of Task for execution

### 7.3.1 Initialization of Task as a Particle

At given point of time, all schedulable task is considered as a set of $N = \{T_1, T_2, T_3, \ldots T_n\}$. Each task (particle) $T_i \in N$ , needs to initialize with its initial position and velocity. Each Periodic task $T_i$ in task set, $N$ has essential characteristics associated with it, like execution time of task ($E_i$), deadline of the task ($D_i$) and rate (period) of the task ($R_i$). These characteristics are already known in Soft Real-Time System before the scheduling algorithm is going to select the task for scheduling. Each task (particle) $T_i \in N$, needs to initialize with its initial position ($P_i$) and initial velocity ($V_i$). $P$ is the set of the initial position of each task and $P = \{P_1, P_2, P_3, \ldots P_n\}$. $V$ is set of the initial velocity of each task and $V = \{V_1, V_2, V_3, \ldots V_n\}$. Initial value of $v_i$ and $p_i$ is going to calculate for each task $T_i \in N$ based on the following equation 14 and equation 15.

$$v_i = T_{i\,(Deadline)} \quad (14)$$

$$p_i = T_{i\,(Execution\,Time)} + T_{i\,(Period)} - T_{i\,(Elapsed\,Time)} \quad (15)$$

It is also necessary to initialize individual task best position $p_i^{best}$ and global best position $p_{gbest}$. Initially, for each task $p_i^{best} = p_i$ and initial value of $p_{gbest}$ for the whole task set is chosen from a minimum of the set $P$. Figure 7.2 represents the task set with its parameters like Execution Time, Deadline, and Rate. Then, for each task, the algorithm initializes its position, velocity, and best position using Equation 14 and Equation 15, as described above.

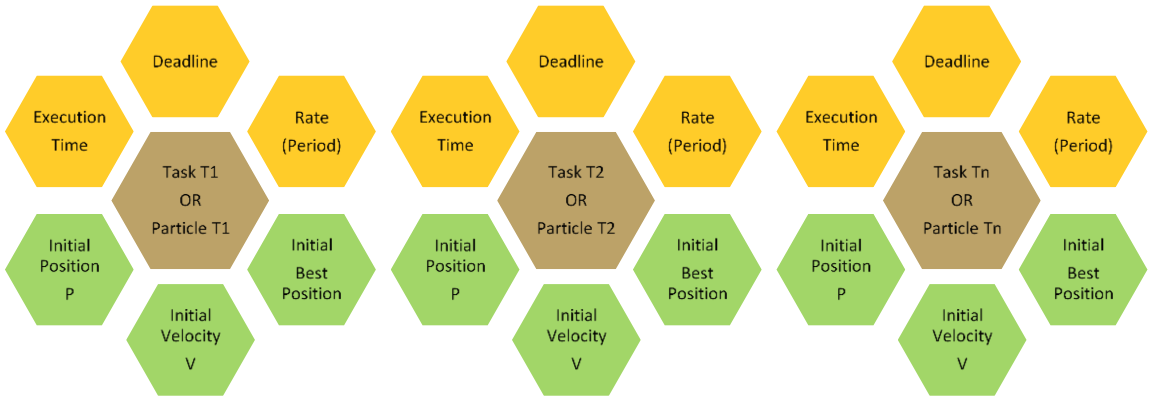

Figure 7.2 – Task Set for PSO Algorithm with its parameters and initial values

### 7.3.2 Compute the velocity and position for each task

The algorithm calculates the velocity $(v)$ for each task which is ready to execute and part of task set $N$. To calculate the velocity $(v)$ value for each task, this algorithm has considered Equation 12 as a base equation and proposed Equation 16, and it is an optimal equation for scheduling problem of Soft Real-Time System.

$$v_{i,d}(t+1) = v_{i,d}(t) + \left(c_1 r_1 \left(p_{i,d}^{best} - p_{i,d}(t)\right)\right) + \left(c_2 r_2 \left(p_{gbest,d} - p_{i,d}(t)\right)\right) \qquad (16)$$

where,

- $v_{i,d}(t+1)$ is the new velocity of task $T_i$ in the $d^{th}$dimension,
- $v_{i,d}(t)$ is the current velocity of task $T_i$ in the $d^{th}$ dimension,
- $c_1 = (T_{i\,(Execution\,Time)})^{-1}$, where $T_{i\,(Execution\,Time)}$ is the execution time of the task $T_i$, which is required on the processor to complete the task,
- $c_2 = (T_{i\,(Deadline)})^{-1}$, where $T_{i\,(Deadline)}$ is the deadline of the task $T_i$

- $r_1$ and $r_2$ are generated uniformly between 0 and 1,
- $p_{i,d}(t)$ is the $T_i$ task's position at time t in the $d^{th}$ dimension,
- $p_{i,d}^{best}$ is the $T_i$ task's best-known position in the $d^{th}$ dimension,
- $p_{gbest,d}$is the best position known to the entire task set in the $d^{th}$ dimension,
- $d \in D$ is the dimension $d$ in the search space.

The algorithm also needs to calculate the new position ($p$) for each task and to calculate it; it is using equation 13, mention in section 7.2.

### 7.3.3 Analyze the position and velocity of each task

The goal of the algorithm is to have all the tasks locate the optima in a multi-dimensional hypervolume. It can be achieved by assigning initial velocity and position to each task as per 7.3.1. The algorithm is executed and, in each iteration, it calculates the new position of each task based on equation 13 and updates its velocity based on equation 16. The evolution of velocity and position is carried out for the specified number of iterations, and the number of iterations depends on the problem size. Over the period, through a combination of exploration and exploitation of known right positions in the search space, the task set cluster or converge together around an optimal task. If any task leaves the boundary of the problem space, then it will be penalized and reflected in the domain by changing its velocity [68].

### 7.3.4 Selection of task for execution

The algorithm calculates the new velocity and the new position of the task in each iteration. The algorithm also changes the value of $p_{gbest}$ in every iteration. $p_{gbest}$ value will be set as the smallest $p_i^{best}$ value. The task which has $p_{gbest} = p_i^{best}$ will be considered and will get the chance to execute on the processor.

### 7.3.5 The Algorithm

**Algorithm: PSO based scheduling approach for Soft Real-Time System**

**Input** : Task Set $N$ = $\{T_1, T_2, T_3, \dots T_n\}$

**Output** : $T_{most\_imp\_task\_to\_execute}$

```
1. $T_{most\_imp\_task\_to\_execute} = -1;$
   /* Initializing each task $T_i$ ($T_i \in N$)with its initial velocity($v_i$), position ($p_i$) and local
   best position ($p_i^{best}$) value. İt also initializing global best ($p_{gbest}$) value. */
2. for each task $T_i$ in set N do
3. |      $v_i = T_{i\,(Deadline)}$ ;
4. |      $p_i = T_{i\,(Execution\,Time)} + T_{i\,(Period)} - T_{i\,(Elapsed\,Time)}$;
5. |      $p_i^{best} = p_i$ ;
6. |      if $p_i^{best} < p_{gbest}$ then
7. |      |    $p_{gbest} = p_i^{best}$;
8. |      |    $T_{most\_imp\_task\_to\_execute} = T_i$;
9. |      end if
```

```
10. end for
    /* Updating velocity($v_i$), position ($p_i$) and local best position ($p_i^{best}$) values of each task
    $T_i$ ($T_i \in N$). İt also update global best ($p_{gbest}$) values after each iteration if the condition
    is satisfied. At the end of stopping condition $T_{most\_imp\_task\_to\_execute}$ will contain the
    most important task which needs to execute by the processor. */
11. while stopingcondition() do
12. |       for each $T_i \in N$ do
13. |       |     $v_i(t+1) = v_i(t) + \left(c_1 r_1 \left(p_i^{best} - p_i(t)\right)\right) + \left(c_2 r_2 \left(p_{gbest} - p_i(t)\right)\right)$;
14. |       |     if $v_i(t+1) < 0.0$ do
15. |       |     |     $v_i(t+1) = 0.0$;
16. |       |     end if
17. |       |     $p_i(t+1) = p_i(t) + v_i(t+1)$;
18. |       |     if $p_i(t+1) < p_i^{best}$ do
19. |       |     |     $p_i^{best} = p_i(t+1)$;
20. |       |     |     if $p_i(t+1) < p_{gbest}$ do
21. |       |     |     |     $p_{gbest} = p_i(t+1)$ ;
22. |       |     |     |     $T_{most\_imp\_task\_to\_execute} = T_i$;
23. |       |     |     end if
24. |       |     end if
25. |       end for
26. end while
27. return $T_{most\_imp\_task\_to\_execute}$ ;
```

### 7.3.6 Convergence Analysis and Parameter Selection

The algorithm proposed in the research is using the following equations to update the value of velocity ($v_i$) and position ($p_i$) of the task during every iteration,

$$v_i(t+1) = v_i(t) + \left(c_1 r_1 \left(p_i^{best} - p_i(t)\right)\right) + \left(c_2 r_2 \left(p_{gbest} - p_i(t)\right)\right)$$

And

$$p_i(t+1) = p_i(t) + v_i(t+1)$$

where,

$c_1 = (T_{i\,(Execution\,Time)})^{-1}$,

$c_2 = (T_{i\,(Deadline)})^{-1}$ and

$0 < c_1, c_2 \leq 1$

Here, it has been assumed that for $r_1 and\ r_2$ are random values between 0 and 1. Also, assumed that $\emptyset_1 = c_1 r_1$ and $\emptyset_2 = c_2 r_2$ and $\emptyset = \emptyset_1 + \emptyset_2$ such that $0 < \emptyset_1 \leq 1$, $0 < \emptyset_2 \leq 1$ and $0 < \emptyset \leq 2$.

Also, this research has considered that when one task is taken into consideration, the other tasks are frizzed in their positions. This assumption gives us the flexibility to omit the suffix $i$ and rewrite the equations as

$$v(t+1) = v(t) + \emptyset_1(p^{best} - p(t)) + \emptyset_2\left(p_{gbest} - p(t)\right) \quad (17)$$

And

$$p(t+1) = p(t) + v(t+1) \qquad (18)$$

This research has used a similar approach adopted by zheng (2003) in their research to check the behaviour and convergence of this algorithm [69]. The equation (17) and (18) can be combined and written in a compact matrix-vector form as follows [68].

$$X(t+1) = AX(t) + B \qquad (19)$$

where,

$X(t+1) = \begin{bmatrix} v(t+1) \\ p(t+1) \end{bmatrix}$ and $X(t) = \begin{bmatrix} v(t) \\ p(t) \end{bmatrix}$

$A = \begin{bmatrix} 1 & -\emptyset \\ 1 & 1-\emptyset \end{bmatrix}$ and $B = \begin{bmatrix} P^{best}\emptyset_1 + P_{gbest}\emptyset_2 \\ P^{best}\emptyset_1 + P_{gbest}\emptyset_2 \end{bmatrix}$

Here, $X(t)$ is the “task state” which is defined by its current position $p(t)$ and velocity $v(t)$. ‘A’ is the dynamic matrix which will use to determine the convergence of Algorithm. The characteristic equation of matrix A is given by

$$\lambda^2 - (trace\ of\ A)\lambda + \det(A) = 0$$

where the trace of $A = (1) + (1 - \emptyset) = 2 - \emptyset$ and $\det(A) = 1$

$$\lambda^2 - (2 - \emptyset)\lambda + 1 = 0$$

To find the roots of this equation,

$$\Delta = (\emptyset - 2)^2 - 4(1)(1) = (\emptyset^2 - 4\emptyset + 4) - 4$$

$$\Delta = (\emptyset^2 - 4\emptyset), 0 < \emptyset \leq 2$$

The definition of $\emptyset$ suggests that

$$\Delta \leq 0 \text{ as we have } 0 < \emptyset \leq 2$$

The experimental data suggests that the value of $\Delta$ remains negative for all the processes. For $\Delta < 0$, the eigenvalues are given by

$$\lambda_1 = \alpha + \beta i \text{ and } \lambda_2 = \alpha - \beta i$$

where

$\alpha = \frac{(2-\emptyset)}{2} \; and \; \beta = \frac{1}{2}\sqrt{-\Delta}$

Here the product of eigenvalues is 1 and the L2 norm of $\lambda_1$and $\lambda_2$are $||\lambda_1|| = ||\lambda_2|| = 1$. To get the optimum results, we need particles to converge at the optimum location i.e., $\lim_{t \to \infty} P_t = P^*$. When a particle finds the equilibrium state than $X^{eq}(t+1) = X^{eq}(t)$ for any t then Equation (19) lead us to

$$X^{eq} = \begin{bmatrix} P^* \\ 0 \end{bmatrix}, P^{eq} = P^*, V^{eq} = 0 \quad (20)$$

This is evident because, at the equilibrium state, particle velocity becomes zero. The eigenvalues of the dynamic matrix have $L_2$ norm equal to 1, and both eigenvalues are complex numbers. In this case, the particle trajectories show harmonic oscillations around the equilibrium point and converge to the optimum value. The discussion clarifies that the algorithm used in the research can get the optimum results, and the system remains stable.

## 7.4 Case Study for Instance of Task Set

The proposed algorithm in section 7.3 has been tested with a set of the periodic task set. This section demonstrates how it operates with one case study shown in Table 7.1. It shows one task set with its arrival time, its deadline, and its required execution time. As described in section 7.3, each task will be initialized with its initial position ($p_i$) using equation 14 and equation 15, and its initial value has been shown in Table 7.2. To get an optimal position for each task $p_i$ will be calculated for N number of times. After that task set will be evaluated, and identify the most important task which needs to be executed by the system. In the above task set (shown in Table 7.1), T5 is the most crucial task, and the scheduling algorithm will select it for execution, so it will meet the deadline, as shown in Table 7.2.

Table 7.1 – An instant of Task Set for Case Study

| Task | Arrival Time | Absolute Deadline | Execution Time |
|---|---|---|---|
| **T1** | 0 | 12 | 1 |
| **T2** | 0 | 12 | 2 |
| **T3** | 0 | 3 | 1 |
| **T4** | 0 | 12 | 2 |
| **T5** | 0 | 2 | 1 |

Table 7.2 – PSO Algorithm Calculation for Instance of Task Set

| Task | Initial Values of $p_i$ | After N iteration Values of $p_i$ | Selection for Execution of Task by PSO Algorithm at t=0 is |
|---|---|---|---|
| **T1** | 13.00 | 28.25 | T5 |
| **T2** | 14.00 | 32.70 | |
| **T3** | 04.00 | 07.24 | |
| **T4** | 14.00 | 32.70 | |
| **T5** | 03.00 | 05.25 | |

## 7.5 Performance Analysis and Result Comparison

PSO based scheduling algorithm has been implemented using the simulator, which was developed in C language and compared with EDF and ACO based scheduling algorithms. The algorithm executes when a new task arrives, or the current task completes its execution. All algorithms are tested with the periodic task set (Data Set) described in Section 3.2 to evaluate their performance. Load of the system ($U_p$) is calculated based on equation 3 (described in section 3.2). All algorithms have been assessed with three major categories of CPU load ($U_p$) values which consider as underload, overload, and highly overload scenarios. If the value of

$\mathrm{U_p} \leq 1$, it is considered as underload scenario, if it is $1.0 < \mathrm{U_p} \leq 1.5$, it considered an overload scenario, and if it is $1.5 < \mathrm{U_p} \leq 5.0$ is considered a highly overload scenario. The value of $\mathrm{U_p}$ varies between 0.5 to 5 for the entire Data Set. All algorithms have been tested on 500-time units to prove their effectiveness [70]. Performance of all algorithms has been measured and evaluated concerning SR and ECU parameters which are explained in Section 3.1. Detailed performance analysis and result comparison has been given in this section.

### 7.5.1 Underload Scenario

PSO, ACO and EDF based scheduling algorithms have been tested in the underload scenario. The scenario is considered an underload when the utilization factor for the task set is less than or equals one. Table 7.3 represents the scenario where the task set contains 1 to 9 tasks, and the utilization factor (CPU load) is less than 1 or equal to 1 ($\mathrm{U_p} \leq 1$). Results show that ECU values remain nearly the same for all these algorithms, whereas for PSO based scheduling algorithm, it is slightly less. Results also indicate SR is not 100% in the very specific case of a task set for the PSO based scheduling algorithm. When the Load of the CPU is less than one, it means that the task set is schedulable and the scheduling algorithm can schedule all tasks, and all tasks can meet their deadline. PSO-based scheduling algorithm equally performs compared to the ACO and EDF-based scheduling algorithm, but still, there is a specific task set where EDF and ACO can schedule it, but PSO missed few task deadlines. Figure 7.3 and 7.4 provides the performance comparison of PSO, ACO and EDF scheduling algorithms in underload scenario concerning ECU and SR parameters.

Table 7.3 – EDF, ACO, and PSO based scheduling algorithms performance in Underload Scenario

| **CPU Load** | **ECU%** | | | **SR%** | | |
|---|---|---|---|---|---|---|
| | **EDF** | **ACO** | **PSO** | **EDF** | **ACO** | **PSO** |
| **0.50** | 49.49 | 49.98 | 49.49 | 100.00 | 100.00 | 100.00 |
| **0.55** | 54.66 | 55.04 | 54.40 | 100.00 | 100.00 | 100.00 |
| **0.60** | 59.39 | 59.88 | 59.39 | 100.00 | 100.00 | 100.00 |
| **0.65** | 64.35 | 65.00 | 64.35 | 100.00 | 100.00 | 100.00 |
| **0.70** | 69.35 | 69.93 | 69.35 | 100.00 | 100.00 | 100.00 |
| **0.75** | 74.31 | 74.88 | 74.31 | 100.00 | 100.00 | 100.00 |
| **0.80** | 79.22 | 79.83 | 79.22 | 100.00 | 100.00 | 100.00 |
| **0.85** | 84.16 | 84.72 | 84.16 | 100.00 | 100.00 | 100.00 |
| **0.90** | 89.16 | 89.62 | 89.15 | 100.00 | 100.00 | 99.99 |
| **0.95** | 94.17 | 94.54 | 94.08 | 100.00 | 100.00 | 99.94 |
| **1.00** | 99.10 | 99.37 | 97.99 | 100.00 | 100.00 | 99.26 |

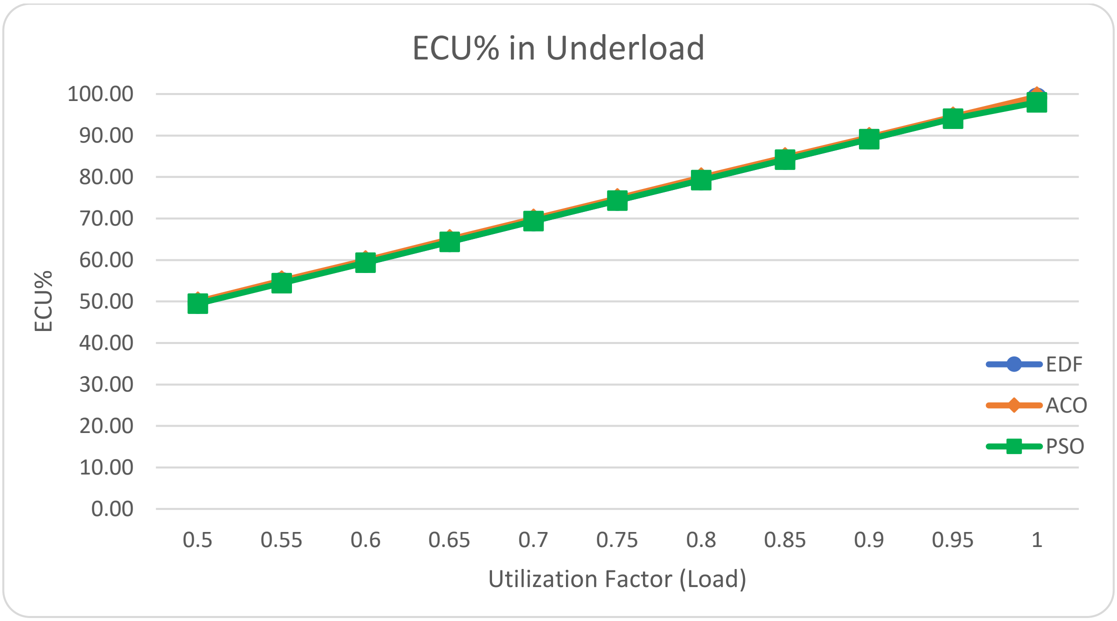

Figure 7.3 – ECU% Vs CPU Load for EDF, ACO and PSO based Scheduling Algorithms in Underload Scenario

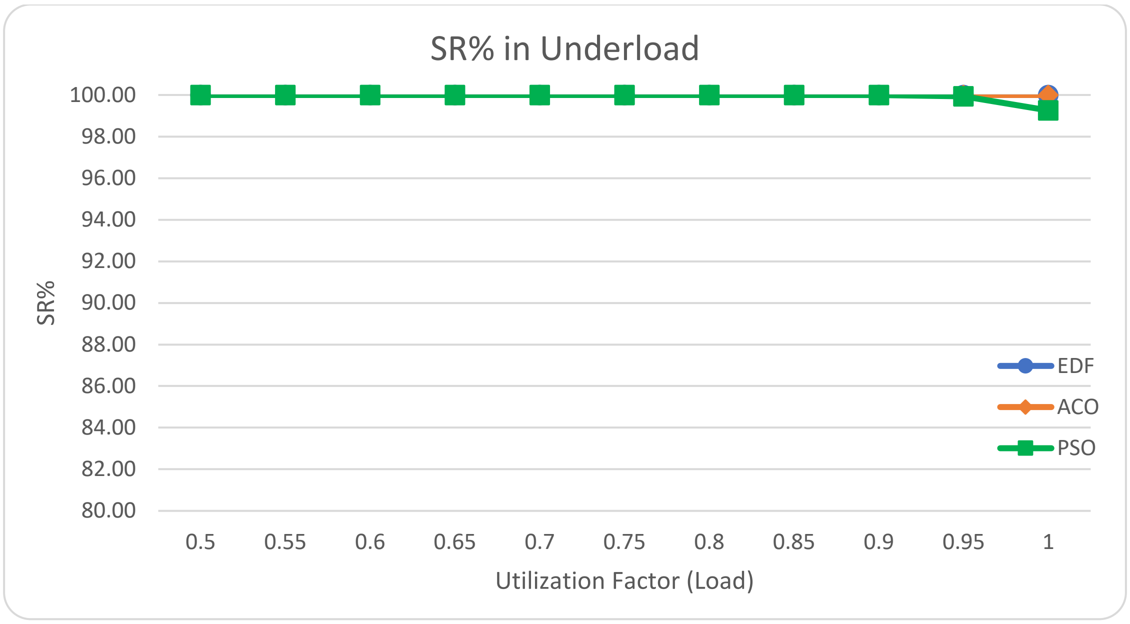

Figure 7.4 – SR% Vs CPU Load for EDF, ACO and PSO based Scheduling Algorithms in Underload Scenario

### 7.5.2 Overload Scenario

PSO, ACO and EDF algorithms have been tested in the overload scenario. The scenario is considered as an overload when the utilization factor for the task set is $1.0 < U_p \leq 1.5$. Table 7.4 represents the scenario where the task set contains 1 to 9 tasks, and utilization factor (CPU Load) vary between $1.0 < U_p \leq 1.5$. Results show a significant difference in ECU and SR values for PSO, ACO and EDF based scheduling algorithms. When the CPU load is greater than 1, the task set is not schedulable, and few tasks will miss their deadline. Table 7.4 observations reflect that in a slightly overload situation, EDF performance degrades very poorly, whereas ACO and PSO based scheduling algorithms are still able to meet most of the deadlines of the given task set. It means in an overload situation, PSO and ACO based scheduling algorithms give better performance compared to the EDF. Even the PSO based scheduling algorithm performs more batter than the ACO based scheduling algorithm. Figure 7.5 and 7.6 provides the performance comparison of EDF, ACO and PSO based scheduling algorithms in overload scenario concerning ECU and SR parameters.

Table 7.4 – EDF, ACO, and PSO based scheduling algorithms performance in Overload Scenario

| CPU Load | ECU% | | | SR% | | |
|---|---|---|---|---|---|---|
| | EDF | ACO | PSO | EDF | ACO | PSO |
| **1.05** | 17.45 | 63.69 | 65.91 | 18.27 | 67.01 | 78.24 |
| **1.10** | 09.21 | 54.22 | 70.60 | 09.31 | 55.01 | 80.53 |
| **1.15** | 06.29 | 51.86 | 67.90 | 06.19 | 50.87 | 75.91 |
| **1.20** | 04.62 | 46.61 | 80.39 | 04.22 | 45.33 | 80.03 |
| **1.25** | 04.06 | 45.15 | 77.35 | 03.67 | 36.23 | 78.97 |
| **1.30** | 03.63 | 38.78 | 74.73 | 03.19 | 35.90 | 76.02 |
| **1.35** | 03.12 | 39.03 | 74.06 | 02.65 | 37.14 | 72.65 |
| **1.40** | 02.66 | 38.05 | 81.39 | 02.24 | 33.91 | 76.49 |
| **1.45** | 02.50 | 34.11 | 78.15 | 02.00 | 30.65 | 70.66 |
| **1.50** | 02.21 | 33.08 | 86.15 | 01.71 | 27.91 | 73.35 |

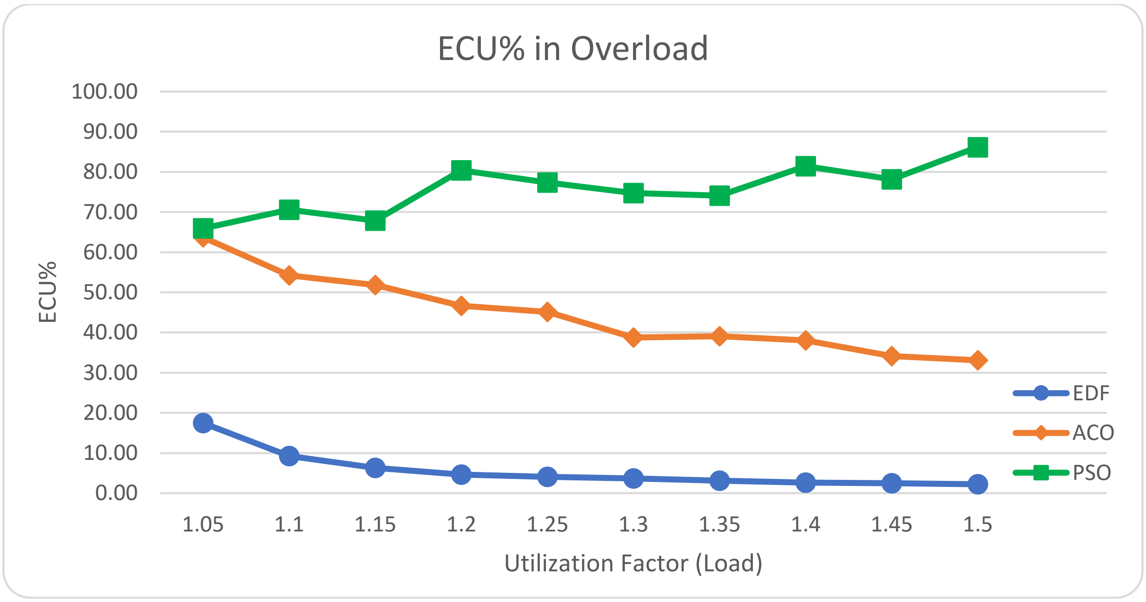

Figure 7.5 – ECU% Vs CPU Load for EDF, ACO and PSO based Scheduling Algorithms in Overload Scenario

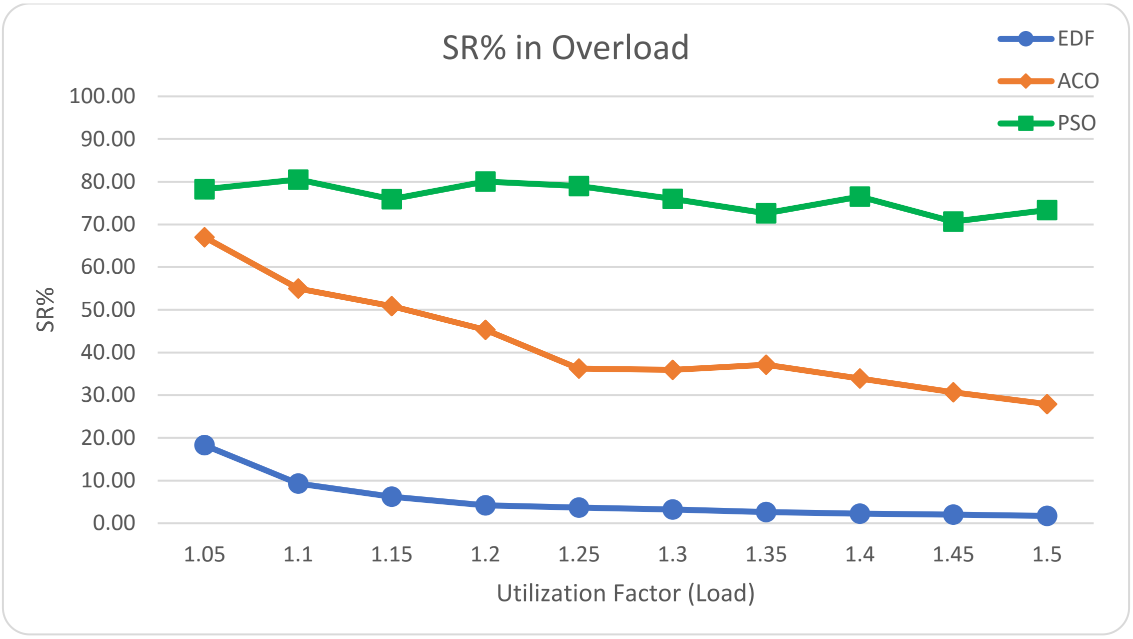

Figure 7.6 – SR% Vs CPU Load for EDF, ACO and PSO based Scheduling Algorithms in Overload Scenario

### 7.5.3 Highly Overload Scenario

PSO, ACO and EDF based scheduling algorithms were also tested in the highly overload scenario. The scenario is considered as a Highly Overload when the utilization factor for the task set is more than 1.5. Table 7.5 represents the scenario where the task set contains 1 to 9 tasks, and the utilization factor (CPU Load) is between 1.5 and 5 ($1.5 < \mathrm{U_p} \leq 5.0$). Results show that ECU and SR values are very low for the EDF scheduling algorithm because the most task in task sets missed their deadlines. When the CPU load is more than one, it means that the task set is not schedulable, and there is no scheduling algorithm exist that can schedule all tasks. There will be tasks in the task set that will miss their deadline, but ACO and PSO based algorithms can schedule some of the tasks that meet their deadline. Even PSO based scheduling algorithm performs better than ACO based scheduling algorithm in highly overload situation. Figure 7.7

and 7.8 provides the performance comparison of EDF, ACO and PSO scheduling algorithms in highly overload scenario concerning ECU and SR parameters.

Table 7.5 – EDF, ACO, and PSO based scheduling algorithms performance in Highly Overload Scenario

| **CPU Load** | **ECU%** | | | **SR%** | | |
|---|---|---|---|---|---|---|
| | **EDF** | **ACO** | **PSO** | **EDF** | **ACO** | **PSO** |
| **1.60** | 2.17 | 45.98 | 85.02 | 1.61 | 37.25 | 71.25 |
| **1.70** | 2.03 | 40.45 | 86.52 | 1.42 | 30.24 | 68.70 |
| **1.80** | 1.93 | 35.52 | 86.34 | 1.30 | 26.39 | 68.27 |
| **1.90** | 1.90 | 33.56 | 85.17 | 1.29 | 25.35 | 65.52 |
| **2.00** | 1.84 | 29.56 | 86.90 | 1.20 | 21.45 | 65.23 |
| **2.25** | 1.76 | 32.51 | 85.89 | 1.04 | 21.24 | 59.81 |
| **2.50** | 1.55 | 25.54 | 87.86 | 0.89 | 15.39 | 56.23 |
| **2.75** | 1.46 | 18.31 | 88.82 | 0.78 | 10.16 | 53.75 |
| **3.00** | 1.32 | 14.66 | 94.25 | 0.63 | 07.11 | 49.21 |
| **3.50** | 1.27 | 15.80 | 94.46 | 0.57 | 07.69 | 45.52 |
| **4.00** | 1.11 | 09.67 | 96.20 | 0.43 | 03.79 | 40.47 |
| **4.50** | 1.08 | 09.86 | 97.69 | 0.38 | 03.37 | 36.99 |
| **5.00** | 0.97 | 08.74 | 97.88 | 0.31 | 02.41 | 32.15 |

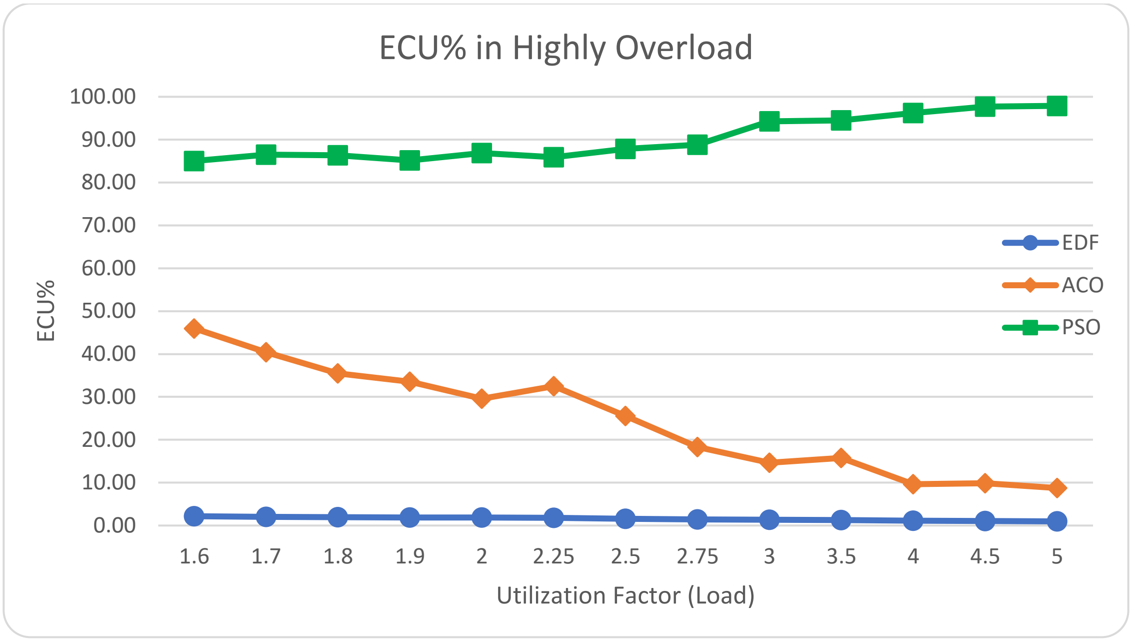

Figure 7.7 – ECU% Vs CPU Load for EDF, ACO and PSO based Scheduling Algorithms in Highly Overload Scenario

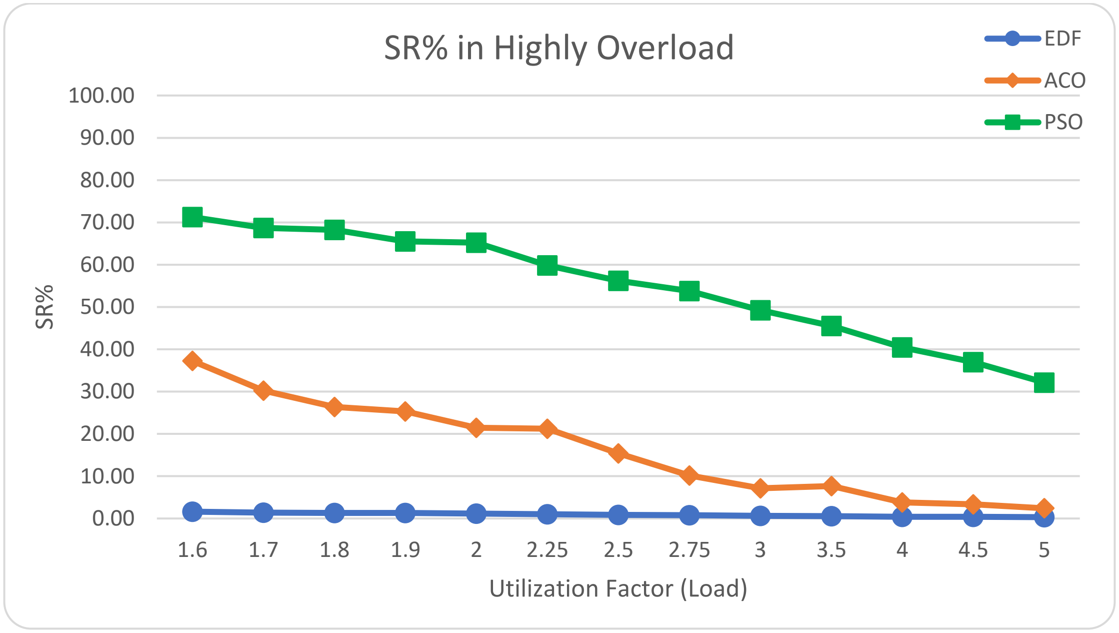

Figure 7.8 – SR% Vs CPU Load for EDF, ACO and PSO based Scheduling Algorithms in Highly Overload Scenario

## 7.5.4 Time Complexity Analysis

This section compares the time complexity of EDF, ACO, and PSO based scheduling algorithms. Practical performance analysis of these algorithms has been done in sections 7.5.1, 7.5.2 and 7.5.3 by implementing these algorithms on the simulator. The experiment set-up has been prepared for the periodic task set so, the time complexity comparison is for a periodic task only. At a given point of time, all schedulable task is considered as a set of $N$ = $\{T_1, T_2, T_3, \ldots T_n\}$. EDF is a dynamic scheduling algorithm and identifies the most crucial task to execute based on the absolute deadline. When the scheduling algorithm is executed to select the most critical task, EDF will have $O(N)$ time complexity [10][71]. ACO based scheduling algorithm use concept of traversing the different path to identify the optimal route and then select the most crucial task for execution. Due to its traversing techniques, when scheduling algorithm will be executed to select the most crucial task ACO based scheduling algorithm will have $O(N^2)$ time complexity to select the most crucial task. The algorithm which proposed with this paper also calculate Velocity and Position of each task for $N$ iteration to identify optimal positions in given task set and because of the time complexity of PSO based scheduling algorithm is also $O(N^2)$ to select the most crucial task. It is true that PSO based scheduling algorithm time complexity is higher than EDF but as discuss in section 7.5.2 and 7.5.3 it gives an excellent performance in overload scenario and even in the modern evolution of electronics devices Real-Time system able to perform faster and able to schedule a task using any algorithm by ignoring its time complexity overhead.

## 7.6 Conclusion

The proposed PSO based scheduling algorithm has been compared with EDF and ACO based scheduling algorithms under the Soft Real-Time periodic task set. The performance parameters SR and ECU have been calculated for each algorithm for a large dataset, and a comparison has been made. It has been observed that during the underload scenario ($\mathrm{U_p} \leq 1$) proposed scheduling algorithm performs similar to the EDF and ACO based algorithms. In a slightly overload situation when $1.00 \leq \mathrm{U_p} \leq 1.5$, EDF performance gets degraded sharply. The proposed algorithm and ACO based scheduling algorithm perform better compare to EDF, and even the proposed approach delivers better than the ACO based scheduling algorithm. During highly overload scenarios ($1.50 \leq \mathrm{U_p} \leq 5.00$) EDF and ACO based algorithms perform poorly, whereas the PSO based scheduling algorithm is still able to meet a specific deadline. So, instead of static or dynamic priority, the proposed approach works well during underload, overload, and highly overload scenarios. The PSO based scheduling algorithm uses the equation n. 16 to take decision on which task should be executed first. This equation gives importance to personal task ($p_{i,d}^{best}$) information as well as whole task set ($p_{gbest,d}$) information. By considering both of this information at the time of selection of task, proposed PSO based scheduling algorithm performs batter compare to the traditional algorithms.