Received: 7 April 2020 | Revised: 16 July 2020 | Accepted: 2 August 2020

DOI: 10.1002/spe.2890



RESEARCH ARTICLE

WILEY

# HEART: Unrelated parallel machines problem with precedence constraints for task scheduling in cloud computing using heuristic and meta-heuristic algorithms

**Amit Kumar Bhardwaj**[1,2] | **Yuvraj Gajpal**[2] | **Chirag Surti**[3] | **Sukhpal Singh Gill**[4]

[1]L.M. Thapar School of Management, Thapar Institute of Engineering and Technology, Dera Bassi Campus, Mohali, India

[2]Supply Chain Management, Asper School of Business, University of Manitoba, Winnipeg, Manitoba, Canada

[3]Department of Information System, Analytics and Supply Chain Management, Rider University, Lawrenceville, New Jersey,

[4]School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK

**Correspondence**
Sukhpal Singh Gill, School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK.
Email: s.s.gill@qmul.ac.uk

**Funding information**
Natural Sciences and Engineering Research Council of Canada, Grant/Award Number: 318689

**Summary**

Cloud computing is becoming a profitable technology because of it offers cost-effective IT solutions globally. A well-designed task scheduling algorithm ensures the optimal utilization of clouds resources and reducing execution time dynamically. This research article deals with the task scheduling of inter-dependent subtasks on unrelated parallel computing machines in a cloud computing environment. This article considers two variants of the problem-based on two different objective function values. The first variant considers the minimization of the total completion time objective function while the second variant considers the minimization of the makespan objective function. Heuristic and meta-heuristic (HEART) based algorithms are proposed to solve the task scheduling problems. These algorithms utilize the property of list scheduling algorithm of unrelated parallel machine scheduling problem. A mixed integer linear programming (MILP) formulation has been provided for the two variants of the problem. The optimal solution is obtained by solving MILP formulation using *A Mathematical Programming Language* (AMPL) software. Extensive numerical experiments have been performed to evaluate the performance of proposed algorithms. The solutions obtained by the proposed algorithms are found to out-perform the existing algorithms. The proposed algorithms can be used by cloud computing service providers (CCSPs) for enhancing their resources utilization to reduce their operating cost.

**KEYWORDS**

cloud computing, heuristics, metaheuristic, resource optimization, scheduling

## 1 | INTRODUCTION

Cloud computing is gaining popularity due to its ability for delivering cost-effective cloud services which can bring a win-win situation for the end users as well as for the service providers.[1]. In the cloud computing environment, the remote users can obtain computational resources over the Internet in scalable fashion based on their requirement.[2] Cloud computing has become a backbone for the IT infrastructure of different industries and organizations, viz., education, weather

  

forecasting, hedge funding, e-commerce, and big data solution.[3,4] Kochan et al[5] used a cloud computing based framework in the domain of hospital supply chain to enhance the performance of demand and supply of healthcare items.[3,6,7]

According to the enterprise cloud computing survey, the top four cloud service providers are Microsoft Azure with 23% share, Amazon Web Services (AWS) with 22% share, Google Cloud with 21% share, and IBM Cloud with 17% share.[8]. This survey forecast that about 90% of enterprises will increase their annual spending on cloud computing. Also, SaaS-based applications are expected to grow by 18%, and Infrastructure/Platform as a service is expected to grow by 27% annually. The survey further indicates the increases in the efficiency of the enterprises through the use of cloud computing solutions. According to the Forbes magazine report, the total turnover of the cloud computing industry was $67 billion in 2015, which is expected to grow by $162 billion in 2020.[9]. The report further reveals that cloud computing will impact the business considerably in near future. The cloud computing services providers (CCSPs) are competing with each other to capture the market share[10]. Hence, the success of a CCSP depends upon the cost-effective offerings to their clients.[11]

In a cloud computing environment, a cloud is considered as a cluster of many distributed computers.[12,13] A physical computer can have more than one virtual machines (VMs) residing on it for a parallel execution of different tasks.[14] Thus, the cloud computing system can be considered as a system of parallel VMs.[15] An end user can use the cloud resources in the form of a lease from the cloud service providers.[16] The VMs can be leased at the price of 10 cents per hour.[17]. In a cloud computing environment, millions of users submit their billions of tasks for processing on unrelated virtual parallel machines environment.[18] In unrelated parallel machines environment, the execution time of a task is considered to be different for different machines.[19]. An unequal execution time of tasks over different machines arises because each machine has different processing speed, memory or complexity.[20]. This article deals with the design of efficient algorithms to achieve the cost-effective solution for cloud computing task scheduling problem using precedence constraints.

### 1.1 | Motivation and our contributions

The main motivation behind this research work is to consider a scheduling problem where unrelated VMs are responsible for processing independent tasks. A task consists of a series of subtasks.[21] A successor subtask can be started only after the completion of its predecessor subtask. The subtasks can be executed independently in the same machine or in a different machine. The execution time of each machine is known in advance. Each machine has known release time (ie, the available time for execution of existing tasks) due to the ongoing process of other jobs. In this article, we have considered two problem variants for two different objective functions. The first variant of the problem considers the minimization of the total completion time objective, while the second variant of the problem considers the minimization of the makespan objective. A mixed integer linear programming (MILP) formulation has been provided for the proposed variants of the problem. The MILP formulation is solved to obtain an optimal solution for small problem instances. We proposed ***HE****uristics and met****A****-heu****R****is****T****ic* ***(HEART)*** algorithms to solve the bigger problem instances. The proposed algorithms are compared with the similar algorithms available in the literature. The benchmark problem instances are introduced to perform the numerical experiments. These problem instances can be used in future research to compare the performance of different algorithms.

### 1.2 | Article organization

The rest of the article is organized as follows. Section 2 provides a literature review of related research work on task scheduling in cloud computing domain along with the literature review of parallel machines scheduling problems. Section 3 provides the problem description and formulation of the proposed problem, whereas Section 4 describes the proposed algorithms. In Section 5, the results of the proposed algorithms are compared with the existing algorithms. Finally, Section 6 provides the conclusion of this research article.

## 2 | RELATED WORK

The related work section is divided into two subsections: (a) task scheduling and (b) unrelated parallel machines.

## 2.1 | Task scheduling in cloud computing

In the last decade, application of scheduling algorithms in the cloud computing environment took the momentum with the aim of improving the resource utilization.[22] Gill and Buyya[23] discussed the resource provisioning for workloads in the parallel computing environment of clouds. Fang et al[24] discussed the load balancing problem in cloud computing environment to meet the quality of service (QoS) requirements. Many researchers considered the genetic algorithm (GA) to solve QoS-oriented scheduling problems in cloud computing.[25-28] Pandey et al[29] used particle swarm optimization (PSO) for job scheduling to minimize the computation and data transmission costs. Tsai et al[21] considered parallel cloud computing services with different processing capacity to perform subtasks. They included processing and receiving cost in their model with the aim to minimize the cost and makespan simultaneously. These research papers indicate the popularity of scheduling issues in cloud computing environment due to its ability to improve resource utilization. Also, there are many algorithms available in the literature to solve the scheduling problem in the production environment. However, the above-mentioned paper did not utilize the existing algorithms/properties form a production scheduling problem to solve the cloud computing scheduling problem. One of the aims of this article is to relate the existing scheduling model from the production environment to the scheduling model in the cloud computing environment.

The problem considered in this article resembles with the parallel machine scheduling problem. The parallel machine scheduling problem has been proved to be NP-hard.[30,31] The parallel machine problem can be classified under three categories on the basis of machine configuration: uniform machine configuration, heterogeneous machine configuration, and unrelated machine configuration. In uniform machine configuration, the processing time of a task is the same for all machines. In heterogeneous machine configuration, the processing time of a task depends on machine speed. In unrelated machine configuration, the processing time of a task is different for different machines. The problem considered in this paper resembles with unrelated parallel machine problems; therefore, the remainder of the literature review discusses only unrelated parallel machine scheduling problems.

## 2.2 | Unrelated parallel machine scheduling

Luis and Ruiz[32] and Lin et al[33] proposed metaheuristics to solve the unrelated parallel machine problems to minimize the makespan objective. Luis and Ruiz[32] proposed iterative greedy local search based metaheuristic to solve the problem. Lin et al[33] proposed an artificial immune system, which combines the feature of the artificial immune system and simulated annealing. Lin et al[34] extended their work for the multi-objective problem by proposing a GA to find the nondominated solutions to minimize the makespan, the total weighted completion time, and the total weighted tardiness objectives. Other research papers in literature[18-20] considered variants of unrelated parallel machine problems such as setup time, batch processing, additional resources, and so on.

Jose et al[35] proposed job scheduling technique (JST) considered non-identical job sizes and unequal ready times over the unrelated parallel batch processing machines to minimize makespan objective. They developed many scheduling heuristics based on first-fit, best-fit, and earliest job ready time rules to solve the problem. Shahvari and Logendran[36] proposed an enhanced Tabu search algorithm (TSA) to solve the problem and considered batch processing in unrelated parallel machines to minimize a linear combination of total weighted completion time and the total weighted tardiness objectives. Joo and Kim[37] considered setup time and production-availability in unrelated parallel machines to minimize the total completion time objective. They proposed a hybrid GA to solve the problem. Cheng and Huang[38] addressed an unrelated parallel machine scheduling problem for jobs with distinct due dates and dedicated machines to minimize the total earliness and tardiness objectives. They developed a modified GA with a distributed release time control mechanism to solve the problem. Diana et al[39] considered unrelated parallel machines with sequence dependent setup times to minimize the makespan. They developed a variable neighborhood descent (VND) metaheuristic to solve the problem. Luis et al[40] considered unrelated parallel machine problem with additional resources to minimize the makespan objective. They proposed a mathematical model based on linear programming formulation to solve the problem. Furthermore, they combined metaheuristic strategies with a linear programming model to solve the bigger problem instances. Oleh and Lars[41] considered scheduling problems in flexible job shops in an unrelated parallel machines environment to minimize the total weighted tardiness objective. They proposed an iterative local search to solve the problem.

Wang et al[42] developed an optimal charging scheduling (OCS) technique for electric vehicles considering the impact of renewable energy sources, which uses MILP to optimize execution time. Deng et al[43] proposed an MILP based two-stage load (TSL) scheduling approach for building load's peak-to-average ratio reduction and improves execution time. These

prior works such as Wang et al[42] and Deng et al[43] use MILP with a limited perspective. None of them considered precedence constraints in their problem. The unrelated parallel machine problem, with precedence constraints, is considered by Herrmann et al,[44] Liu and Yang,[45] Afzalirad and Rezaeian,[46] and Gacias et al.[47] The problem presented in this article considers that many independent tasks consist of numerous inter-dependent subtasks. Each subtask can be processed independently but it can be started only after processing of predecessor subtask. Thus, the problem discussed in this article can be considered as a special case of unrelated parallel machine scheduling problem with precedence constraints. Gacias et al[47] considered scheduling problem with precedence constraints and sequence-dependent setup times to minimize the total completion time and maximum lateness objectives independently. They proposed a branch-and-bound based exact algorithm and limited discrepancy based heuristic method to solve the problem. Afzalirad and Rezaeian[46] considered resource constrained unrelated parallel machine scheduling problem with sequence-dependent setup time, precedence constraints, and machine eligibility restrictions to minimize the makespan objective. They developed a GA and artificial immune system (AIS) to solve the problem.

Herrmann et al[44] considered unrelated parallel machine scheduling problem with precedence constraints to minimize makespan objective. Herrmann et al[44] highlighted the problem as an application of office scheduling problem where workers perform different interdependent tasks with different skill sets for each subtask. They proposed a look-ahead based HH heuristic to solve the problem. The HH heuristic is based on scheduling a task in each iteration, which could lead to a late schedule of some tasks in the future. Liu and Yang[45] proposed a serial schedule (SS) heuristic to solve unrelated machine problem with precedence constraints for minimizing the makespan objective. The SS heuristic assigns a task to the earliest available machine iteratively. They compared SS algorithm with HH algorithm of Herrmann et al.[44] Their numerical experiment showed the better performance of SS heuristic as compared to the HH heuristic.

The algorithm proposed by Gacias et al[47] and Afzalirad and Rezaeian[46] cannot be used to solve the problem considered in this article because they considered additional constraints. However, the algorithm proposed by Herrmann et al[44] and Liu and Yang[45] can be used to solve the problem considered in this article. We propose a *simple heuristic and ant colony based metaheuristic* to solve the problem. In the numerical experiment, we compare the performance of proposed algorithms with the SS heuristic of Liu and Yang[45] and HH heuristic of Herrmann et al.[44]

## 2.3 | Critical analysis

As discussed above, none of the above-mentioned papers considered precedence constraints in their unrelated parallel machines problem. Existing works are considering tasks independently, which leads to poor scheduling decisions, as shown in the evaluation section. Wang et al[42] and Deng et al[43] use MILP without considering. Hence, the precedence constraints-based MILP and heuristic/meta-heuristic approaches out-perform the baseline models. The current work shows a proof-of-concept of the novel approach and establishes that the proposed approach out-performs in a fundamental computing platform. This research work performs task scheduling of inter-dependent subtasks on unrelated parallel computing machines in a cloud computing environment using heuristic and meta-heuristic algorithms. We have performed the evaluation of our research work by considering all the possible performance parameters. Table 1 shows the comparison of proposed work (HEART) with existing techniques based on important parameters.

**TABLE 1** Comparison of our proposed work (HEART) with related works

| Work | Heuristic Optimization | Multi-Scale System | Dataset | Unrelated Parallel Machines | Precedence Task Constraints | Heuristic and Meta-Heuristic Algorithms | Cloud Computing | Evaluation Parameters | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | MKS | TCT | RPD | CPU | PG | ST |
| HH Heuristic[44] | √ | √ | Small | | √ | | | √ | | | | | |
| SS Heuristic[45] | √ | | Small | | √ | | √ | √ | | | | | |
| JST[35] | √ | | Small | √ | | | | √ | | | | | |
| TSA[36] | | | Small | √ | | | | √ | | | | | |
| OCL[42] | | √ | Small | | | | | √ | | | | | |
| TSL[43] | | √ | Small | | | | | √ | | | | | |
| **HEART (this work)** | √ | √ | Small and Large | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |

Abbreviations: MKS: Makespan value produced by an algorithm, TCT: Total completion time value produced by an algorithm, RPD: Relative percentage deviation of an algorithm from the best solution, CPU: CPU time consumed by an algorithm, PG: Percentage gap of an algorithm form the optimal solution and ST: Scheduling Time.

## 3 | SYSTEM MODEL AND PROBLEM FORMULATION

The section discusses the cloud model and problem formulation.

### 3.1 | Cloud model

The proposed algorithms in this article can be deployed in real cloud platforms to allow efficient task scheduling in unrelated machines. This is quite evident in modern systems due to variation in compute performance, bandwidth availability, and dynamic resource consumption statistics. Thus, we describe a large-scale distributed cloud platform model suitable for the proposed heuristic and meta-heuristic algorithms. The system model is shown in Figure 1. The design strategy as illustrated in the figure is as follows: The data is acquired from the *Data Acquisition layer* which consists of API gateways, IoT devices including sensors and point of scale (PoS) systems (AVAC[48]). These data are encapsulated in a task using gateways to be sent to *Computing and Communication layer*. Herein, the cloud nodes communicate using light-weight message passing approaches like MQTT to share task data and computational meta-data. MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol.[49] The *Analytics layer* resides in one of the cloud nodes which handles the complete system and is responsible for monitoring and scheduling of tasks and cloud machines. The final task results are sent to the end user from the analytics layer using alerts, web-portals, or gateway applications (Mancini et al).[50]

### 3.2 | Problem formulation

The problem considered in this article deals with a situation in which a cloud computing scheduler receives $n$ independent tasks for executing at $m$ resources. Each task consists of a series of subtasks and is acceptable for processing on any resources. The cloud computing scheduler wants to assign $nt$ inter-dependent subtasks from set $T = (T_1, T_2, T_3, \ldots T_i, \ldots T_{nt})$, to $m$ available resources $R = (R_1, R_2, \ldots, R_m)$. A task $i$ consists of $n^i$ inter-dependent series of subtasks denoted by set $T^i = (T_l, \ldots T_{l+n^i})$, where $T_l$ and $T_{l+n^i}$ are the first and last subtask for the task $l$. There is a temporal relationship among subtasks of task $i$, viz., all subtasks are performed in a sequential series. The subtask $T_{l+1}$ can be

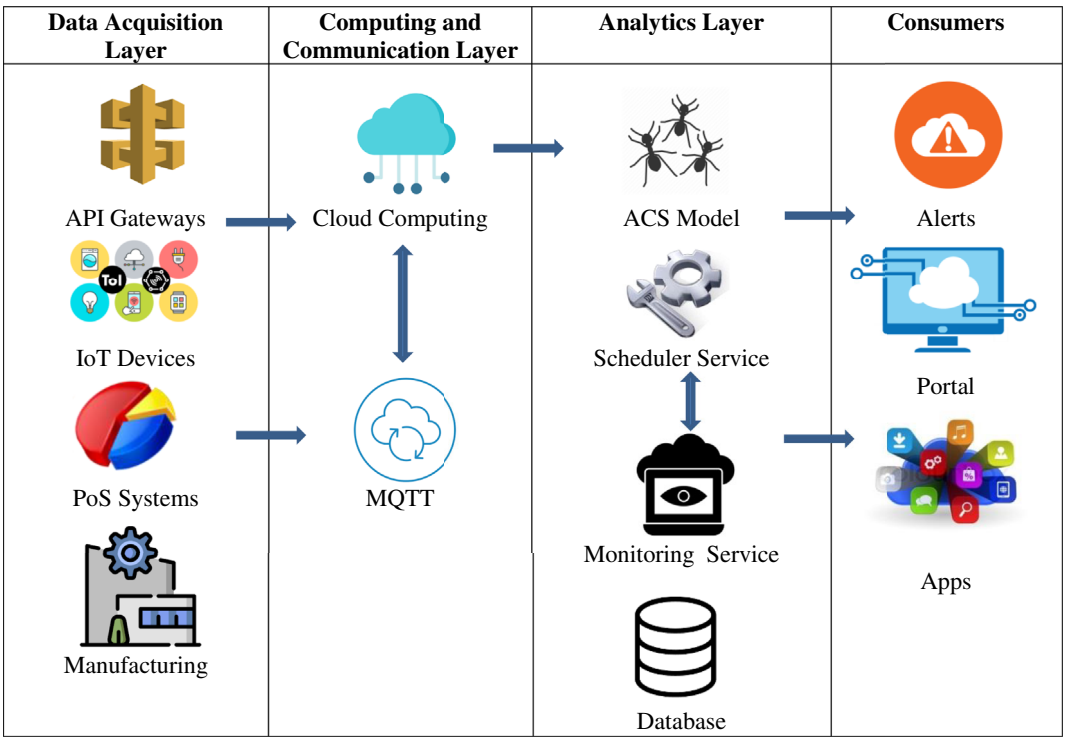


**FIGURE 1** System model [Color figure can be viewed at wileyonlinelibrary.com]

started only after completion of the sub-task $T_l$; the subtask $T_{l+2}$ can be started only after completion of the subtask $T_{l+1}$ and so on. Each subtask has just one predecessor except the first subtask of a task. Let $pred(T_i)$ denote the predecessor of task $T_i$, which can be defined as follows.

$$pred(T_i) = \begin{cases} T_{i-1} & if subtasks\ T_{j-1}\ and\ T_j\ belongs\ to\ the\ same\ job \\ T_0 & otherwise \end{cases}$$

Here $T_0$ is an imaginary subtask with zero process time. We assume that this subtask has already been completed. A subtask $T_i$ can be executed to any cloud resources $R_k \subseteq R$; however, it can be started only after the completion of predecessor subtask $pred(T_i)$.

The process time $P_{ik}$ for executing the subtask $T_i$ on resource $R_k$ is known in advance. A resource $R_k$ is available for processing any subtask only after time $t_k$ because the resources are assumed to be, at the time, executing previously assigned subtask. A subtask-preemption is not allowed and resources are not allowed to process more than one subtask at a time. A subtask is executed on a single resource at a time and the given resources are available continuously. The problem involves assigning subtasks to appropriate resources in such a way that the specified objective function is minimized. A MILP formulation is provided for the two variants of the problem. The first variant of the problem considers the minimization of the total completion time objective function and the second variant of the problem considers the minimization of the Makespan objective. The first problem is denoted as Problem 1 and the second problem is denoted as Problem 2. Two variants of the problem are formulated using the following decision variables.

***Decision variables*** :

$F_{jk}$ Completion time on resource $j$ for a subtask scheduled at position $k$,

$X_{ijk}$ Binary variable taking value 1 if subtask $i$ is assigned to resource $j$ at position $k$; 0 otherwise,

$C_i$ Completion time of task $i$,

$C_{\max}$ Makespan of the optimal solution.

***Problem 1*** : ***Minimization of total completion time***

$$Min\, Z = \sum_{i=1}^{n} C_i \tag{1}$$

*Subject to*

$$\sum_{j=1}^{m}\sum_{k=1}^{n} X_{ijk} = 1 \quad \forall i \in T \tag{2}$$

$$\sum_{i=1}^{n} X_{ijk} \leq 1 \ \ \forall j \in R, k \in L \tag{3}$$

$$F_{j1} >= t_j + \sum_{i=1}^{n} X_{ij1} \times P_{ij} \forall j \in R; \tag{4}$$

$$F_{jk} \geq F_{j\,k-1} + \sum_{i=1}^{n} X_{ij1} \times P_{ij} \forall j \in R, k \geq 2 \tag{5}$$

$$C_i \geq F_{jk} - M(1 - X_{ijk}) \ \ \forall i \in T, \forall j \in R, k \in L \tag{6}$$

$$C_i \geq C_{pred(T_i)} + \sum_{j=1}^{m}\sum_{k=1}^{n} p_{ij} X_{ijk} \ \ \forall i \in T \tag{7}$$

$$C_0 = 0 \tag{8}$$

$$X_{ijk} \in \{0, 1\}, \ \ \forall i \in T, j \in R, k \in L \tag{9}$$

Equation (1) provides the expression for the minimization of the total completion time. Equation (2) ensures that a task $T_i$ is scheduled on one resource and one position only. Equation (3) ensures that a maximum one subtask can be assigned for a given resource at a given position. Equation (3) also implies the possibility of not assigning any subtask in a

given resource at a given position. Equation (4) calculates the completion time for the task scheduled at first position on resource $j$, while the Equation (5) calculates the completion time for other positions. Equation (6) provides the completion time calculation for task $i$. Equations (7) and (8) preserve the predecessor constraints among different subtasks. Finally, Equation (9) enforces binary condition for decision variable $X_{ijk}$.

***Problem 2*** : ***Minimization of Makespan***

$$MinZ = C_{\max} \tag{10}$$

*Subject to*

$$\sum_{j=1}^{m}\sum_{k=1}^{n} X_{ijk} = 1 \quad \forall i \in T \tag{11}$$

$$\sum_{i=1}^{n} X_{ijk} \leq 1 \;\; \forall j \in R, k \in L \tag{12}$$

$$F_{j1} \geq t_j + \sum_{i=1}^{n} X_{ij1} \times P_{ij} \forall j \in R \tag{13}$$

$$F_{jk} \geq F_{j\,k-1} + \sum_{i=1}^{n} X_{ij1} \times P_{ij} \forall j \in R, k \geq 2 \tag{14}$$

$$C_i \geq F_{jk} - M(1 - X_{ijk}) \;\; \forall i \in T, \forall j \in R, k \in L \tag{15}$$

$$C_i \geq C_{pred(T_i)} + P_{ij} \;\; \forall i \in T \tag{16}$$

$$C_0 = 0 \tag{17}$$

$$C_{\max} \geq C_i \quad \forall i \in T \tag{18}$$

$$X_{ijk} \in \{0, 1\}, \;\; \forall i \in T, j \in R, k \in L \tag{19}$$

Equation (10) provides expression for the minimization of Makespan objective. Equations (11) to (17) are same as Equations (2) to (8). The additional constraints (18) provide the calculation of Makespan. Finally, Equation (19) enforces the binary constraints for decision variable $X_{ijk}$.

## 4 | PROPOSED ALGORITHMS

In this article, an optimal solution is generated for the total completion time and the Makespan objective using the MILP formulation provided in the previous section. "A Mathematical Programming Language (AMPL)" software is used to solve the MILP formulation. The MILP formulation can only be solved for small problem instances because the CPU time increases exponentially with increase in problem size. Furthermore, a heuristic and a meta-heuristic are proposed to solve the cloud resource allocation problem for the large problem instances.

The proposed algorithms utilize the list-scheduling dominant property of the unrelated scheduling problem with precedence constraints. A list-scheduling algorithm is an assignment rule that finds a feasible schedule for a given order of tasks. The assignment rule considers the tasks one by one from the given list of tasks for assigning them to the machine on the basis of the partial schedule given by the previous scheduled tasks. In the list-scheduling algorithm, decision of scheduled tasks is not changed in future. If a list schedule algorithm evaluates all the feasible schedules related to all possible order of tasks and one of these feasible schedules is guaranteed to find an optimal solution for the problem, then it is said that the list algorithm produces dominant set of solution.[51] In literature, it is said that a list scheduling algorithm finds an optimal solution if it produces a dominant set of solutions. The parallel machine scheduling literature has mainly two list scheduling algorithm: (a) list scheduling algorithm with earliest available machine-assignment-rule and (b) list scheduling algorithm with earliest completion time (EST) of task-assignment-rule. In the earliest available machine assignment rule based list-scheduling algorithm, next task from the given order of tasks is scheduled on a first

available machine. In the earliest completion-time of task-assignment-rule based list-scheduling-algorithm, next task from the given order of tasks is scheduled on a machine where the task completes earliest.

It has been proven that both list scheduling algorithms find an optimal solution for uniform parallel machine scheduling problem with precedence constraints and makespan objective $P||C_{max}$.[47,51] However, literature is silent about the list-scheduling algorithm for the total completion-time objective. To the best of our knowledge, optimal list scheduling algorithm for total completion time objective is not available in literature. It can be easily seen that both list scheduling algorithms will produce the same feasible schedule for a given order of tasks in uniform parallel machine problem. However, they will produce different feasible solutions for the unrelated parallel machine scheduling problem with precedence constraints.

Consider an instance with three tasks and two machines with precedence constraint $J_1 < J_3$ (ie, task $J_1$ precedes task $J_3$). The process time of tasks on two machines is given in Table 2.

The optimal solution for this instance is obtained by scheduling task $J_1$ in $M_2$ and tasks $J_2$ and $J_3$ in $M_2$. The Gantt chart representing optimal solution with Makespan 8 is shown in Figure 2.

There can be only three possible lists for the problem satisfying precedence constraints; $\{J_1\text{-}J_2\text{-}J_3\}$, $\{J_1\text{-}J_3\text{-}J_2\}$, and $\{J_2\text{-}J_1\text{-}J_3\}$. The schedule obtained by list scheduling algorithm with earliest available machine assignment rule is given in Figure 3.

It is clear from Figure 3 that list scheduling algorithm with earliest available machine assignment rule does not produce a dominant set of solution. The schedule obtained by list scheduling algorithm with the EST of task assignment rule is shown in Figure 4. It appears that the list scheduling algorithm with the EST of task assignment rule produces a dominant solution. However, we could not provide a formal proof of this property. The formal proof is an open-ended research question for future research. We use this property in our proposed algorithms to solve the problem.

This problem instances can also be used for completion time objective. It can be easily shown that the earliest available machine assignment rule is non-dominant for total completion time objective as well.

**TABLE 2** Process time of jobs on two machines

| Task /Machine | $M_1$ | $M_2$ |
|---|---|---|
| $J_1$ | 10 | 4 |
| $J_2$ | 6 | 10 |
| $J_3$ | 2 | 10 |

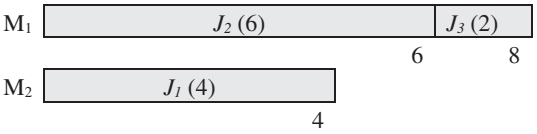


**FIGURE 2** Gantt chart of optimal solution

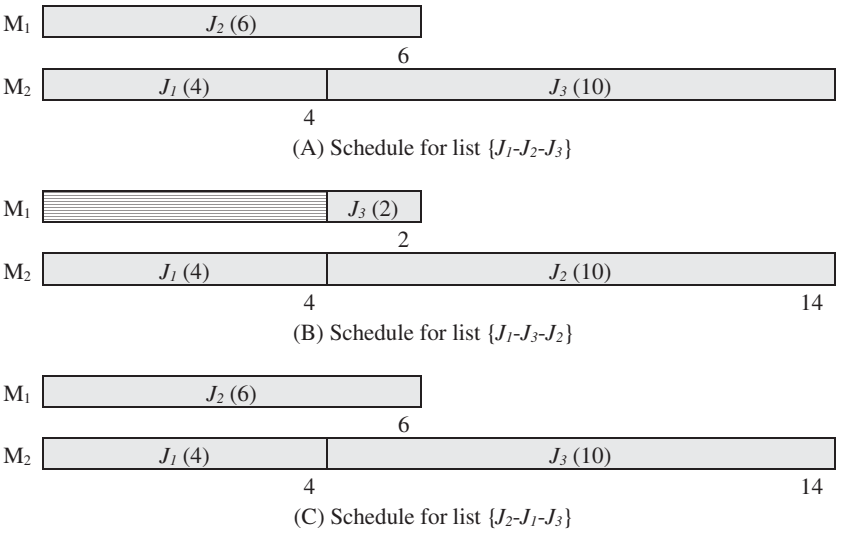


**FIGURE 3** Schedules for list scheduling algorithm with earliest available machine assignment rule

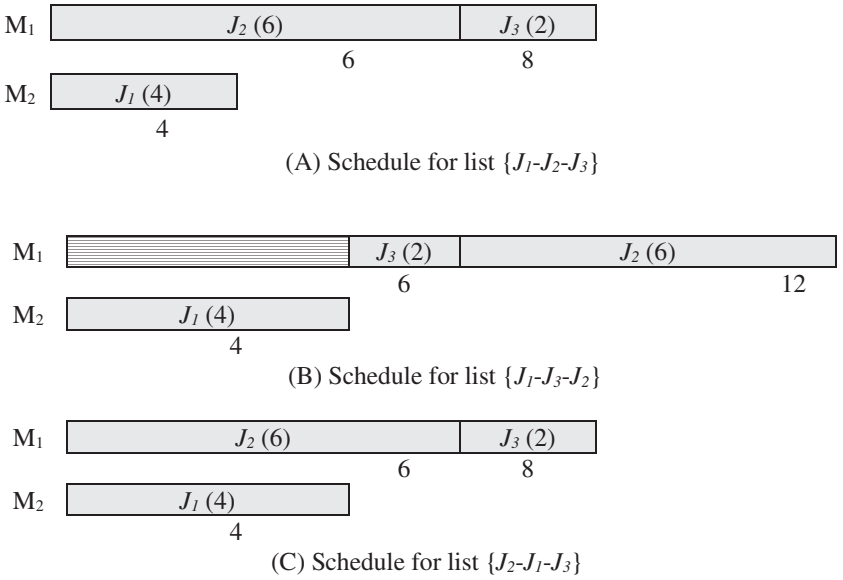


**FIGURE 4** Schedule for list scheduling algorithm with earliest completion time of task assignment rule

## 4.1 | Earliest completion time based heuristic

A heuristic solution method is developed to solve the unrelated parallel machine scheduling problem with precedence constraint. The heuristic partially utilizes the list-scheduling algorithm based on EST of tasks assignment rule. The proposed heuristic assigns tasks iteratively until all the subtasks are scheduled. In each iteration, all unassigned feasible subtasks, that is, subtasks with assigned predecessor, are evaluated on all the resources. An unassigned subtask with the lowest completion time is selected for the assignment. The detailed description of the heuristic algorithm (Algorithm 1) is provided below.

**Algorithm 1.** Earliest Completion Time (EST) based Heuristic Algorithm

**Step 0:** Initialize the available time of all the resources as their ready time (ie, $A_k = t_k; k = 1, \ldots, m$) and completion time of all subtasks to be infinite (ie, $C_i = \infty; i = 1, \ldots, nt$). Set the completion time of $0^{th}$ subtask to be zero (ie, $C_0 = 0$). Initialize unscheduled subtasks set $S = \{T_1, T_2, \ldots, T_{nt}\}$.

**Step 1:** Build the feasible set of subtasks $\Omega$ from the unscheduled subtasks set $S$ whose predecessor subtask has already been scheduled.

**Step 2:** Evaluate the earliest completion time $E_i$ for subtask $T_i$ from set $\Omega$.

$$E_i = \min_{k \in M} \{\max(F_{pred(T_i)}, A_k) + P_{ik}\} \tag{20}$$

**Step 3:** Select the task for scheduling that has minimum earliest completion time $E_i$. Determine the resource in which this task can be scheduled for minimum completion time. Assume that subtask $T_i$ provides earliest completion time at resource Rk.

**Step 4:** Schedule the subtask $T_i$ at resource $R_k$ for processing at time $t = \max(CT_{\Pr ed(T_i)}, A_k)$. Update the available time $A_k$ of resource $R_k$ and the completion time $C_i$ of task $T_i$as follows.

$$A_k = t + P_{ik} \tag{21}$$

$$C_i = t + P_{ik} \tag{22}$$

**Step 5:** Remove task $T_i$ from set $S$.

**Step 6:** Go back to *Step 1* if there is unscheduled task, otherwise stop.

## 4.2 | Ant colony system algorithm

This paper uses the ant colony system (ACS) algorithm to solve the problems under consideration. The ACS algorithm is used in many parallel machine scheduling problems. Arnaout et al[52] proposed an ACS based algorithm to minimize the Makespan in the parallel machine environment. Behnamian et al[53] proposed a hybrid meta-heuristic for a Makespan minimization scheduling problem. They considered the ACS and variable neighborhood search (VNS) algorithms to solve the parallel machine problem. Gao et al[54] designed a multi-objective ACS algorithm for the VM placement in the cloud computing environment for the purpose of improving server utilization and power efficiency. Hua et al[55] proposed an ACS algorithm for optimizing computing resources allocation problem. Gajpal and Rajendran[56] used ACS for minimizing the completion-time variance of jobs in flowshops. Zhang et al[57] used ACS in electric vehicle routing problem with recharging stations for minimizing energy consumption. Thiruvady et al[58] used ACS in mining domain for a shared resource constrained scheduling problem. Ting and Chen[59] used a multiple ant colony optimization (MACO) algorithm to solve the location-routing problem with capacity constraints on depots and routes. ACS is also used by Thepphakorn et al[60] for an academic time tabling problem. Hong et al[61] used an ACS based heuristic an efficent algorithm for a two-stage supply chain problem with fixed costs. The successful application of ant colony algorithm on solving different combinatorial optimization problem motivated us to use ACS for solving unrelatted parallel machine problem with precedence constraints.

In the ACS algorithm, artificial ants are created to find better solutions to a particular problem by using the information from the solutions of previous iterations. At the end of each iteration, the solutions are stored in the trail intensity of each path. Finally, the ant solutions are generated by using current trail intensity. Detailed explanations and descriptions of the application of ACS can be found in Stützle and Hoos.[62] The fundamental procedure of ACS is shown in Algorithm 2:

**Algorithm 2.** Ant Colony System (ACS) Algorithm

***Step* 1:** Initialize the trail intensities and parameters
***Step 2*:** While (termination condition is not met) do the following:

- Generate an ant solution for each ant using the trail intensities.
- Improve ant solution using local search.
- Update trail intensities using elitist ants.

***Step 3*:** Return the best solution found so far.

The trail intensities is denoted as$\tau_{ik}$, which determines the intensity of assigning task $T_i$ to resource $R_k$. We initialize the trail intensity $\tau_{ik} = 0.01, \forall\, i \in T,\, k \in R$.

### 4.2.1 | Generate an ant solution

In classical, ant colony algorithm a task is selected for scheduling on the basis of trail intensity $\tau_{ik}$ for assigning task $T_i$ to resource $k$. The EST-based list scheduling property seems to provide the optimal solution for the problem considered in this paper. Hence, the proposed ACS utilizes this property for building the ant solution along with the trail intensity of ant algorithm. An ant solution is generated in a similar way of heuristic algorithm is described in Section 4.1. All the steps of generating an ant solution in ACS are similar to the heuristic algorithm except in *step 3*. In step 3 of heuristic algorithm, subtask $T_i$ is chosen from set $\Omega$ for assignment to resource $R_k$. The subtask $T_i$ is chosen on the basis of EST rule. In the proposed ACS, a subtask is chosen using the combination of EST rule and trial intensity rule. The rules are selected randomly with 90% probability of EST rule and 10% probability on trial intensity rule. The EST rule is described in step 2 and step 3 of the heuristic algorithm. The trail intensity rule of ant algorithm uses the following probability for selecting task $T_i$.

$$\mathrm{P}_{ik} = \frac{\tau_{ik}}{\sum_{l \in \Omega} \tau_{lk}} \tag{23}$$

### 4.2.2 | Local search

Once the solution is constructed by the ant, the ant solution is improved by local search. In the proposed local search scheme, a randomly selected subtask is removed from its original position and re-inserted in all other feasible position. If the best insertion position improves the current solution, then the move is accepted for future evaluation. All the subtasks are evaluated for possible improvement through insertion. The process is repeated if at least one subtask is relocated with improved solution. The process is stopped when none of the subtask insertion is able to improve the solution. The local search uses a speed up mechanism to reduce CPU time by avoiding the evaluation of infeasible insertions. This article uses two simple properties to identify infeasible insertion places. The first property states that the insertion of a subtask is not feasible anywhere before its immediate predecessor's subtask. The second property states that the insertion of a subtask is not feasible anywhere after its successor's subtask (not only the immediate successor task but all the successor's subtasks).

### 4.2.3 | Updating trail intensity

After all ants have constructed their solutions, the trail intensities are updated using the solution of $\gamma$ elitist ants. The elitist ants are defined as the $\gamma$ best ant solution found so far. Elitist ants are updated by comparing the present elitist ant solutions with the current ant solutions. While updating elitist ants, the algorithm ensures that the solutions of elitist ants are distinct from each other. The trail intensity of assigning task $T_i$ to resource $R_k$ is updated using elitist ants as follows:

$$\tau_{ik}^{new} = \rho \times \tau_{ik}^{old} + \sum_{\mu=1}^{\gamma} \Delta\tau_{ik}^{\mu}, i = 1, 2, \ldots, nt; k = 1, 2, \ldots, m. \tag{24}$$

Here $\rho$ is the called evaporation factor, taking value between 0 and 1. The first term in Equation (24) represents evaporations of existing trail intensity. The second term represents the deposition of pheromone by $\gamma$ elitist ant where

$$\Delta\tau_{ik}^{\mu} = \begin{cases} 1/L^{\mu} & \text{if task } T_i \text{ is assigned to resouce } R_k \text{ in the } \mu^{th} \text{ eltist ant} \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

Here $L^{\mu}$ is the objective function value for the $\mu^{th}$ elitist ant solution. Our ACS algorithm has a computation complexity of O(n^2), much better than the prior work baselines with complexity of O(n^2 * log(n)).

## 5 | PERFORMANCE EVALUATION

This section presents numerical experiment and evaluation of proposed algorithms. The optimal solution for small problem instances is obtained by solving MILP formulation. AMPL software with CPLEX solver is used to solve the MILP formulation. The software can solve the problem size of 10 subtasks within 15 minutes. In this section, we provide the experiment for small instances as well as the large problem instances. All the algorithms (ie, heuristics and meta-heuristic) are implemented in same simulation environment, coded in *C Language*, and run on an AMD Opteron 2.6 GHz PC with 16 GB memory on Unix OS. The HH and SS heuristic are coded on the basis of pseudo code available in those papers. We used the following notations for reporting results.

HH: HH algorithm of Herrmann et al.[44]
SS: SS algorithm of Liu and Yang.[45]
EST: EST based proposed heuristic algorithm.
ACS: Ant colony algorithm based solution.
MKS: Makespan value produced by an algorithm.
TCT: Total completion time value produced by an algorithm.
RPD: Relative percentage deviation of an algorithm from the best solution.
CPU: CPU time (in seconds) consumed by an algorithm.
PG: Percentage gap of an algorithm form the optimal solution.

n: Independent tasks received by a CCSP at a particular time.

m: number of virtual machine/commodity computers available to process above n at a particular time.

nt: total number of inter-dependent subtasks of n tasks.

The performance of proposed solution method is evaluated through PG and RPD value. The formula used to calculate the PG and RPD is given below.

$$\text{PG} = \{(AS - Opt) * 100\}/Opt \tag{26}$$

$$\text{RPD} = \{(AS - Best) * 100\}/Best \tag{27}$$

Where, *AS* represents the solution of the algorithm, *Opt* represents the optimal solution, and *Best* represents the best solution among all the solutions used for evaluation.

## 5.1 | Experiments on small instances

The small instances are generated to find the optimal solution. In small instances, the number of tasks considered is 2, 3, 4, and 5 and the numbers of resources considered are 2, 3, and 4. Thus, a total of 12 groups of problem instances are generated. These groups are represented by AY1 to AY12. We generated 10 problem instances for each group and thus the total of 120 small problem instances is generated. The number of subtasks for each task are generated from a uniform distribution in the range of [2, 3]. The process time of tasks are generated from a uniform distribution in the range of [10, 25].

### 5.1.1 | Experiments on small instances for total completion time objective

The results of HH heuristic, SS heuristics, EST heuristics, and Ant colony algorithm for small instances are reported in Table 3 for the total completion time objective. The average total completion time obtained for HH heuristic, SS heuristics, EST heuristics, and ACS meta-heuristic are shown in Table 3 along with their Percentage Gap (PG) value. The average PG for 12 groups of problem instances are reported in the bottom of Table 3. The average PG of HH and SS heuristic from optimal solution is noted as 10.7% and 4.26%, respectively. The results show that the solution is far away from the

**TABLE 3** Experimental results of small instances for total completion time objective

| Instance No. | *n* | *m* | *nt* | Optimal Solution | | HH Heuristic | | | SS Heuristic | | | EST Heuristic | | | ACS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TCT | CPU | TCT | CPU | PG | TCT | CPU | PG | TCT | CPU | PG | TCT | CPU | PG |
| AY1 | 2 | 2 | 6 | 319.2 | 0.2 | 337.5 | < 1 | 5.34 | 328.80 | < 1 | 2.95 | 322.40 | < 1 | 0.86 | 320.40 | < 1 | 0.41 |
| AY 2 | 2 | 3 | 5 | 212.30 | 0.18 | 222.30 | < 1 | 5.76 | 217.80 | < 1 | 2.44 | 217.90 | < 1 | 2.39 | 213.30 | < 1 | 0.43 |
| AY 3 | 2 | 4 | 5 | 138.60 | 0.21 | 141.20 | < 1 | 1.80 | 148.80 | < 1 | 7.43 | 141.10 | < 1 | 1.29 | 139.50 | < 1 | 0.97 |
| AY4 | 3 | 2 | 8 | 431.00 | 4.47 | 474.9 | < 1 | 10.96 | 454.4 | < 1 | 6.46 | 450.1 | < 1 | 4.77 | 432 | < 1 | 0.35 |
| AY5 | 3 | 3 | 8 | 330.30 | 6.01 | 376.20 | < 1 | 14.13 | 344.30 | < 1 | 4.02 | 337.30 | < 1 | 2.16 | 331.10 | < 1 | 0.27 |
| AY6 | 3 | 4 | 8 | 316.30 | 11.34 | 335 | < 1 | 5.80 | 337 | < 1 | 6.39 | 323.5 | < 1 | 2.20 | 316.3 | < 1 | 0 |
| AY 7 | 4 | 2 | 10 | 732.40 | 37.47 | 830.7 | < 1 | 15.57 | 752.6 | < 1 | 3.17 | 748.6 | < 1 | 2.46 | 733.3 | < 1 | 0.14 |
| AY 8 | 4 | 3 | 10 | 611.80 | 123.92 | 686.7 | < 1 | 12.62 | 631.1 | < 1 | 3.52 | 631.4 | < 1 | 3.52 | 612.7 | < 1 | 0.16 |
| AY 9 | 4 | 4 | 10 | 416.50 | 78.80 | 462.1 | < 1 | 11.38 | 436 | < 1 | 4.71 | 432.9 | < 1 | 3.88 | 417 | < 1 | 0.13 |
| AY10 | 5 | 2 | 12 | 1173.10 | 1582.59 | 1345.7 | < 1 | 15.85 | 1202 | < 1 | 2.48 | 1183.5 | < 1 | 0.88 | 1174.7 | < 1 | 0.10 |
| AY11 | 5 | 3 | 12 | 848.30 | 3348.49 | 967.3 | < 1 | 14.87 | 870.4 | < 1 | 2.79 | 865 | < 1 | 2.26 | 850.6 | < 1 | 0.22 |
| AY 12 | 5 | 4 | 13 | 667.70 | 7765.25 | 761.8 | < 1 | 14.76 | 700.20 | < 1 | 4.95 | 685.6 | < 1 | 2.64 | 670 | < 1 | 0.32 |
| Average | | | | 516.46 | 1079.91 | 578.45 | < 1 | 10.74 | 535.28 | < 1 | 4.28 | 528.28 | < 1 | 2.44 | 517.58 | < 1 | 0.29 |

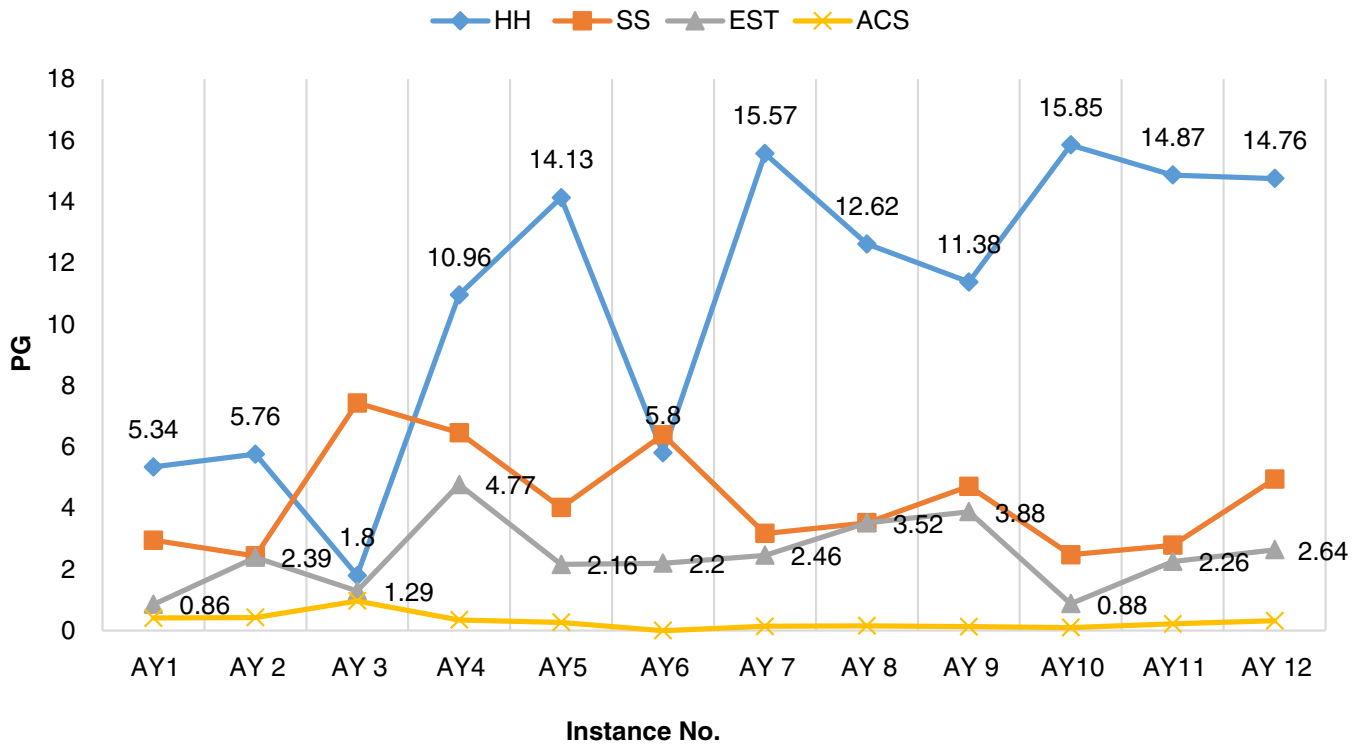


**FIGURE 5** Percentage gap of algorithms with TCT objective function for small data set [Color figure can be viewed at wileyonlinelibrary.com]

optimal solution for existing heuristics. The average PG of EST heuristic is noted as 2.46%, which is good as compared to the existing HH heuristic and SS heuristic. The PG from the optimal solution for ACS is noted as 0.29%, which is close to the optimal solution. Moreover, ACS metaheuristic gives the best results for all 120 small instances problem as compare to HH heuristic, SS heuristics, and EST heuristics.

The CPU time of HH heuristic, SS heuristics, and EST heuristics and ACS meta-heuristic is also reported in Table 3. The comparison of CPU is fair because all the algorithms are executed on the same simulation environment. The CPU time of optimal method is on average 1079.91 seconds for 120 problem instances. It can be observed that the CPU time for optimal method increases exponentially with increase in problem size. The exponential increase in CPU time makes the use of optimal method practically impossible for solving bigger problem instances. This observation also justifies the use of EST heuristic and ACS metaheuristic over HH heuristic and SS heuristics for solving the task scheduling problem.

Figure 5 depicts that the performance of ACS is better than the performance of HH, SS, and EST heuristics. ACS obtains nearby results to optimal solution for total completion time objective.

### 5.1.2 | Experiments on small instances for Makespan objective

The performance results of HH, SS, EST, and ACS for Makespan objective problem for small instances are reported in Table 4 and Figure 6. Table 4 reports the RPD value of HH heuristics, SS heuristics, EST heuristics, and ACS meta-heuristic, viz., 10.03%, 13.12%, 9.19%, and 1.31%, respectively. The PG value of HH heuristics, SS heuristics, and EST heuristics is far away from the optimal solution for Makespan objective. The PG from the optimal solution for ACS is noted as 1.37%, which is very near to the optimal solution.

Figure 5 depicts that the performance of metaheuristic (ACS) is better than the HH, SS and EST. ACS generates the solution close to the optimal solution for Makespan objective. An interesting observation about the CPU time of optimal solution can be made from Tables 3 and 4. The CPU time of Makespan objective is considerably lower than the CPU time of the total completion time objective. The results indicate that solving Makespan objective problem is easier than solving the total completion time objective problem.

**TABLE 4** Experiment results of small instances for Makespan objective problem

| Instance no. | *n* | *m* | *nt* | Optimal Solution | | HH Heuristic | | | SS Heuristic | | | EST Heuristic | | | ACS Heuristic | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | MKS | CPU | MKS | CPU | PG | MKS | CPU | PG | MKS | CPU | PG | MKS | CPU | PG |
| AY1 | 2 | 2 | 6 | 77 | 0.19 | 78.6 | < 1 | 1.95 | 82 | < 1 | 6.64 | 80.8 | < 1 | 4.41 | 77 | < 1 | 0 |
| AY 2 | 2 | 3 | 5 | 55.8 | 0.16 | 57.5 | < 1 | 3.03 | 60.1 | < 1 | 7.39 | 60.3 | < 1 | 7.32 | 56 | < 1 | 0.31 |
| AY 3 | 2 | 4 | 5 | 39 | 0.10 | 39.8 | < 1 | 2.09 | 43 | < 1 | 10.29 | 40.2 | < 1 | 2.46 | 39.5 | < 1 | 1.14 |
| AY4 | 3 | 2 | 8 | 82.1 | 0.62 | 90.6 | < 1 | 10.29 | 93.9 | < 1 | 16.39 | 92.5 | < 1 | 11.34 | 82.9 | < 1 | 0.97 |
| AY5 | 3 | 3 | 8 | 60.3 | 0.59 | 66.8 | < 1 | 10.92 | 68.6 | < 1 | 13.95 | 66.6 | < 1 | 9.18 | 61.1 | < 1 | 1.27 |
| AY6 | 3 | 4 | 8 | 55.4 | 0.77 | 57.3 | < 1 | 3.44 | 64.4 | < 1 | 15.37 | 60.9 | < 1 | 8.09 | 55.6 | < 1 | 0.40 |
| AY 7 | 4 | 2 | 10 | 110 | 3.02 | 124.4 | < 1 | 14.10 | 122.2 | < 1 | 11.74 | 120.1 | < 1 | 8.61 | 111.4 | < 1 | 1.17 |
| AY 8 | 4 | 3 | 10 | 81.1 | 4.03 | 91.3 | < 1 | 12.87 | 92.4 | < 1 | 14.11 | 92.7 | < 1 | 12.11 | 82.1 | < 1 | 1.09 |
| AY 9 | 4 | 4 | 10 | 59 | 3.01 | 65.8 | < 1 | 12.50 | 68.9 | < 1 | 17.60 | 69.3 | < 1 | 14.48 | 60.3 | < 1 | 2.23 |
| AY10 | 5 | 2 | 12 | 140.9 | 11.44 | 158.5 | < 1 | 13.03 | 156.2 | < 1 | 11.17 | 149.1 | < 1 | 5.39 | 142.3 | < 1 | 0.86 |
| AY11 | 5 | 3 | 12 | 94.4 | 18.64 | 110.2 | < 1 | 16.74 | 104.9 | < 1 | 11.13 | 105.7 | < 1 | 10.77 | 96.6 | < 1 | 2.22 |
| AY 12 | 5 | 4 | 13 | 72.2 | 20.72 | 85.8 | < 1 | 19.40 | 87.6 | < 1 | 21.69 | 86.4 | < 1 | 16.16 | 75.3 | < 1 | 4.02 |
| Average | | | | 77 | 5.27 | 85.55 | < 1 | 10 | 87 | < 1 | 13.1 | 85 | < 1 | 9.2 | 78 | < 1 | 1.3 |

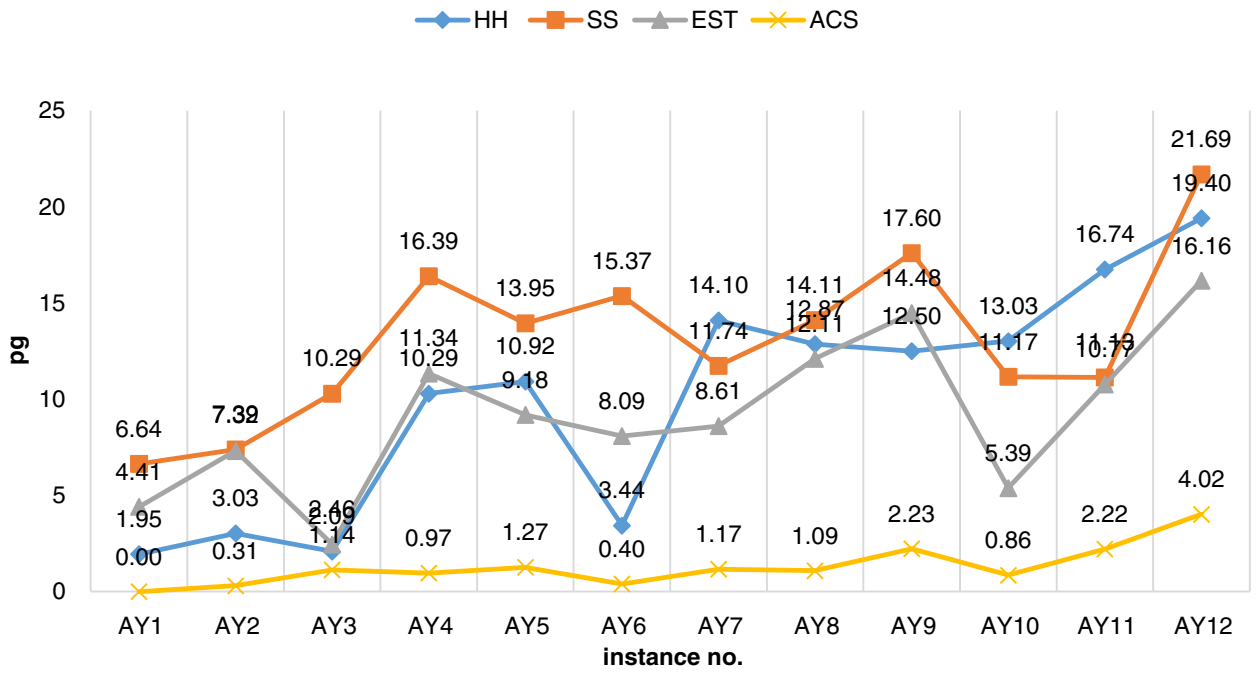


**FIGURE 6** Percentage gap of algorithms with makespan objective function for small data set [Color figure can be viewed at wileyonlinelibrary.com]

## 5.2 | Experiment results of large instance

This section provides the numerical analysis for large problem instances. The number of tasks ($n$) considered is 15, 30, 45, 60, 75, 90, 105, 120, 135, and 150 and the number of resources ($m$) considered is 2, 5, and 7. Thus, the total of 30 groups of problem instances is generated. In a given problem instance, the number of subtasks for a given task is generated in the range of [2, 15]. The process time of subtasks is generated from a uniform distribution in the range of [10, 25].

### 5.2.1 | Experimental results on large instances for total completion time objective

Table 5 reports the experimental results of total completion time objective for the large problem instances. The average RPD value over 300 problem instances indicates that the ACS has the best performance followed by EST heuristic, SS heuristic and then by HH heuristic. The average RPD of ACS, EST heuristic, SS heuristic, and HH heuristic are 0%, 0.45%, 0.69%, and 20.3%, respectively.

Figure 7 visualizes the RPD value of all four algorithms for 30 problem groups. The performance of ACS is better than the performance of other algorithms. The performance of HH heuristic is poor, which is indicated in small problem

**TABLE 5** Numerical results for heuristics and meta-heuristics for total completion time objective problem

| Instance No. | *n* | *m* | *nt* | HH Heuristic | | | SS Heuristic | | | EST Heuristic | | | ACS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TCT | CPU | RPD | TCT | CPU | RPD | TCT | CPU | RPD | TCT | CPU | RPD |
| BG1 | 15 | 2 | 51 | 23 234.7 | < 1 | 16.62 | 20 378.2 | < 1 | 1.04 | 20 349.5 | < 1 | 0.93 | 20 171.30 | 0.70 | 0 |
| BG2 | 15 | 5 | 49 | 8141.1 | < 1 | 16.39 | 7149.4 | < 1 | 1.78 | 7102.0 | < 1 | 1.15 | 7023.20 | 1.10 | 0 |
| BG3 | 15 | 7 | 54 | 7312.5 | < 1 | 18.24 | 6285.4 | < 1 | 1.69 | 6245.2 | < 1 | 1.08 | 6180.30 | 1.70 | 0 |
| BG4 | 30 | 2 | 103 | 92 248.5 | < 1 | 16.42 | 80 250.5 | < 1 | 0.94 | 80 165.9 | < 1 | 0.83 | 79 531.90 | 10.50 | 0 |
| BG5 | 30 | 5 | 101 | 33 339.9 | <1 | 22.33 | 27 646.0 | <1 | 0.93 | 27 554.1 | <1 | 0.61 | 27 399.90 | 13.60 | 0 |
| BG6 | 30 | 7 | 106 | 27 342.9 | <1 | 19.66 | 23 123.7 | <1 | 1.08 | 23 009.6 | <1 | 0.54 | 22 883.00 | 18.50 | 0 |
| BG7 | 45 | 2 | 152 | 210 796.4 | <1 | 18.25 | 181 339.4 | <1 | 0.81 | 181 030.5 | <1 | 0.61 | 179 964.00 | 41.40 | 0 |
| BG8 | 45 | 5 | 150 | 80 986.1 | <1 | 21.52 | 67 609.4 | <1 | 0.60 | 67 447.2 | <1 | 0.37 | 67 213.70 | 49.00 | 0 |
| BG9 | 45 | 7 | 153 | 57 569.0 | <1 | 20.06 | 48 784.9 | <1 | 0.78 | 48 541.1 | <1 | 0.25 | 48 424.60 | 58.10 | 0 |
| BG10 | 60 | 2 | 205 | 359 178.1 | <1 | 20.09 | 303 241.2 | <1 | 0.82 | 302 909.2 | <1 | 0.70 | 300 843.50 | 106.90 | 0 |
| BG11 | 60 | 5 | 205 | 141 010.1 | < 1 | 20.16 | 118 385.3 | < 1 | 0.57 | 118 197.8 | < 1 | 0.40 | 117 731.20 | 134.60 | 0 |
| BG12 | 60 | 7 | 206 | 101 209.1 | < 1 | 21.05 | 84 537.5 | < 1 | 0.61 | 84 246.8 | < 1 | 0.27 | 84 034.90 | 152.70 | 0 |
| BG13 | 75 | 2 | 254 | 684 446.9 | < 1 | 15.57 | 598 368.9 | < 1 | 0.55 | 597 737.5 | < 1 | 0.44 | 595 122.00 | 217.20 | 0 |
| BG14 | 75 | 5 | 257 | 232 465.4 | < 1 | 22.21 | 193 354.5 | < 1 | 0.47 | 192 926.7 | < 1 | 0.24 | 192 495.50 | 281.70 | 0 |
| BG15 | 75 | 7 | 257 | 153 139.5 | < 1 | 21.83 | 127 408.5 | < 1 | 0.56 | 127 013.2 | < 1 | 0.24 | 126 709.30 | 319.70 | 0 |
| BG16 | 90 | 2 | 303 | 804 813.9 | < 1 | 19.95 | 681 828.7 | < 1 | 0.73 | 680 945.6 | < 1 | 0.59 | 676 930.60 | 379.90 | 0 |
| BG17 | 90 | 5 | 300 | 300 210.3 | < 1 | 22.76 | 247 651.6 | < 1 | 0.50 | 247 176.3 | < 1 | 0.29 | 246 482.80 | 467.10 | 0 |
| BG18 | 90 | 7 | 305 | 215 329.1 | < 1 | 20.86 | 179 634.3 | < 1 | 0.47 | 179 158.3 | < 1 | 0.20 | 178 809.40 | 579.20 | 0 |
| BG19 | 105 | 2 | 367 | 1 250 548.4 | < 1 | 17.49 | 1 075 103.5 | < 1 | 0.62 | 1 073 774.6 | < 1 | 0.49 | 1 068 609.60 | 689.70 | 0 |
| BG20 | 105 | 5 | 360 | 411 214.1 | < 1 | 24.53 | 333 454.4 | < 1 | 0.44 | 333 115.2 | < 1 | 0.33 | 332 017.50 | 846.80 | 0 |
| BG21 | 105 | 7 | 360 | 309 928.5 | < 1 | 20.40 | 258 690.6 | < 1 | 0.45 | 257 913.3 | < 1 | 0.14 | 257 550.20 | 962.10 | 0 |
| BG22 | 120 | 2 | 405 | 1 456 331.2 | < 1 | 19.49 | 1 235 319.1 | < 1 | 0.63 | 1 234 693.6 | < 1 | 0.56 | 1 228 088.40 | 972.90 | 0 |
| BG23 | 120 | 5 | 407 | 565 726.7 | < 1 | 22.46 | 465 574.8 | < 1 | 0.46 | 464 651.3 | < 1 | 0.26 | 463 463.00 | 1245.40 | 0 |
| BG24 | 120 | 7 | 409 | 415 008.3 | < 1 | 20.55 | 347 439.6 | < 1 | 0.42 | 346 619.2 | < 1 | 0.16 | 346 069.00 | 1452.60 | 0 |
| BG25 | 135 | 2 | 459 | 1 898 078.2 | < 1 | 20.70 | 1 612 879.8 | < 1 | 0.58 | 1 611 788.4 | < 1 | 0.51 | 1 604 489.00 | 1538.40 | 0 |
| BG26 | 135 | 5 | 459 | 736 659.2 | < 1 | 22.02 | 608 457.3 | < 1 | 0.43 | 607 255.0 | < 1 | 0.23 | 605 897.90 | 1937.60 | 0 |
| BG27 | 135 | 7 | 461 | 519 861.5 | < 1 | 22.14 | 429 560.2 | < 1 | 0.41 | 428 478.9 | < 1 | 0.14 | 427 871.40 | 2175.90 | 0 |
| BG28 | 150 | 2 | 508 | 2 285 765.6 | < 1 | 20.39 | 1 929 552.9 | < 1 | 0.57 | 1 928 130.1 | < 1 | 0.48 | 1 919 198.30 | 2171.40 | 0 |
| BG29 | 150 | 5 | 509 | 880 779.3 | < 1 | 23.09 | 723 195.0 | < 1 | 0.42 | 721 789.1 | < 1 | 0.23 | 720 166.60 | 2635.40 | 0 |
| BG30 | 150 | 7 | 515 | 643 509.3 | < 1 | 20.87 | 536 226.5 | < 1 | 0.34 | 535 015.5 | < 1 | 0.12 | 534 410.90 | 2984.40 | 0 |
| Average | 83 | 5 | 281 | 496 872.8 | < 1 | 20.3 | 418 414.4 | < 1 | 0.69 | 417 832.7 | < 1 | 0.45 | 416 192.76 | 748.21 | 0 |

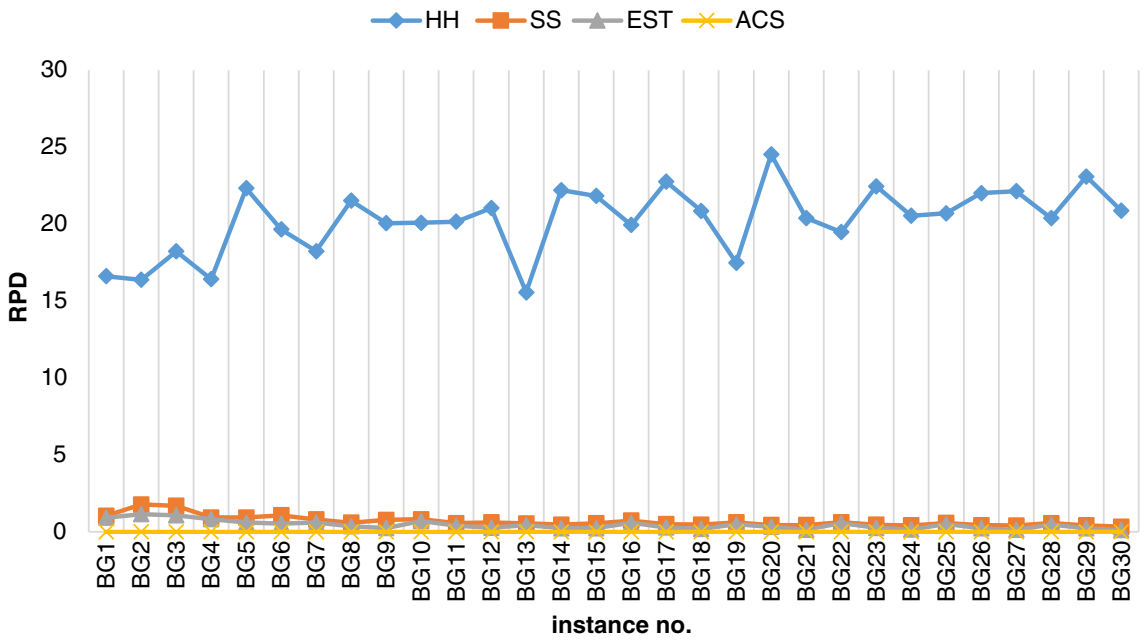


**FIGURE 7** RPD of algorithms with TCT objective function for large data set [Color figure can be viewed at wileyonlinelibrary.com]

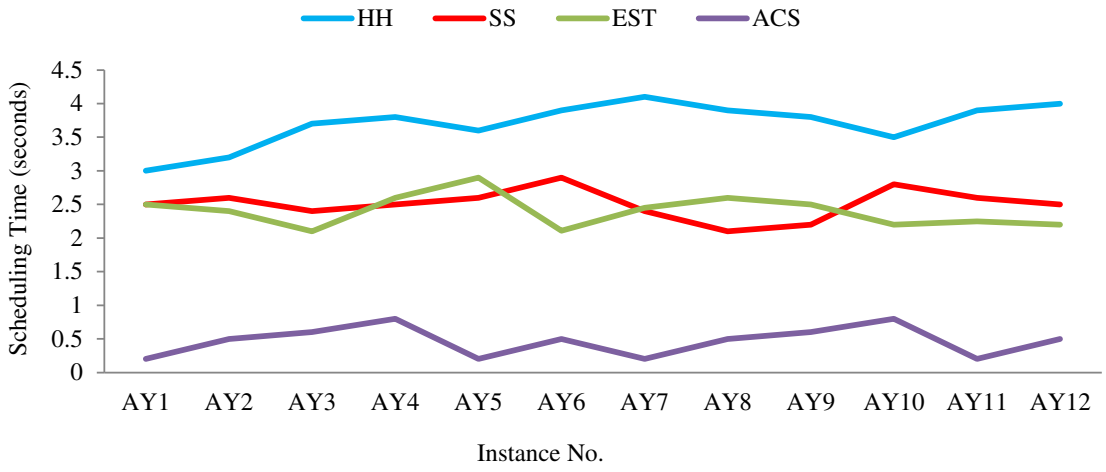


**FIGURE 8** Comparison of scheduling time for algorithms with small dataset [Color figure can be viewed at wileyonlinelibrary.com]

instances as well. One of the reasons for poor performance of the heuristic is that it did not use any list algorithm property of the problem. The performance of SS heuristic is close to the proposed EST heuristic but still inferior than the EST heuristic. The SS heuristic uses earliest available machine assignment rule. We have shown in Section 4 that the earliest machine assignment rule does not provide dominant solution of the problem. This is one of the reasons for the poor perforamnce of SS heuristic as compared to the EST heuristic.

Figure 8 shows the comparison of scheduling time for algorithms with small dataset and ACS performs better than EST, SS, and HH. ACS has 13%, 15.5%, and 21% less than EST, SS, and HH, respectively. The reason behind better performance of ACS is the implementation of precedence task constraints during task scheduling.

### 5.2.2 | Experimental results on large instances for Makespan objective problem

Table 6 and Figure 9 report the performace of HH, SS, EST, and ACS in large problem instances for Makespan objective.

**TABLE 6** Experiment results of large instance for Makespan objective

| Job Instance | *n* | *m* | *nt* | HH Heuristic | | | SS Heuristic | | | EST Heuristic | | | ACS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | MKS | CPU | RPD | MKS | CPU | RPD | MKS | CPU | RPD | MKS | CPU | RPD |
| BG1 | 15 | 2 | 51 | 671.1 | < 1 | 13.4 | 613.30 | < 1 | 3.49 | 608.50 | < 1 | 2.69 | 593.10 | < 1 | 0.0 |
| BG2 | 15 | 5 | 49 | 242.30 | < 1 | 13.7 | 235.20 | < 1 | 10.07 | 229.80 | < 1 | 7.88 | 213.40 | 0.80 | 0.0 |
| BG3 | 15 | 7 | 54 | 200.8 | < 1 | 17.5 | 188.20 | < 1 | 10.35 | 186.20 | < 1 | 9.28 | 170.80 | 1.20 | 0.0 |
| BG4 | 30 | 2 | 103 | 1366.5 | < 1 | 13.2 | 1238.40 | < 1 | 2.43 | 1229.80 | < 1 | 1.67 | 1209.20 | 4.00 | 0.0 |
| BG5 | 30 | 5 | 101 | 501.3 | <1 | 18.9 | 441.10 | <1 | 4.26 | 440.40 | <1 | 4.25 | 423.30 | 7.80 | 0.0 |
| BG6 | 30 | 7 | 106 | 386.1 | <1 | 15.6 | 353.4 | <1 | 5.74 | 353.5 | <1 | 5.64 | 334.30 | 10.4 | 0.0 |
| BG7 | 45 | 2 | 152 | 2086.8 | <1 | 15.3 | 1845.9 | <1 | 1.70 | 1838.1 | <1 | 1.20 | 1815.40 | 13.2 | 0.0 |
| BG8 | 45 | 5 | 150 | 796.9 | <1 | 18.8 | 692.6 | <1 | 3.01 | 690.4 | <1 | 2.68 | 672.40 | 24.3 | 0.0 |
| BG9 | 45 | 7 | 153 | 551.3 | <1 | 16.6 | 490.8 | <1 | 3.74 | 483.5 | <1 | 2.15 | 473.50 | 27.6 | 0.0 |
| BG10 | 60 | 2 | 205 | 2735.6 | <1 | 16.7 | 2391.7 | <1 | 1.80 | 2382.3 | <1 | 1.40 | 2349.80 | 31.4 | 0.0 |
| BG11 | 60 | 5 | 205 | 1036.7 | < 1 | 17.3 | 904.9 | < 1 | 2.43 | 907 | < 1 | 2.57 | 884.40 | 58.4 | 0.0 |
| BG12 | 60 | 7 | 206 | 724.6 | < 1 | 16.8 | 642.4 | < 1 | 3.36 | 634.3 | < 1 | 2.18 | 621.40 | 67.3 | 0.0 |
| BG13 | 75 | 2 | 254 | 3876 | < 1 | 14.1 | 3439.3 | < 1 | 1.14 | 3428.6 | < 1 | 0.80 | 3401.80 | 58.8 | 0.0 |
| BG14 | 75 | 5 | 257 | 1332.9 | < 1 | 19.5 | 1143.4 | < 1 | 2.19 | 1136.6 | < 1 | 1.54 | 1119.40 | 118.9 | 0.0 |
| BG15 | 75 | 7 | 257 | 892.7 | < 1 | 18.2 | 781 | < 1 | 3.16 | 774.7 | < 1 | 2.38 | 757.50 | 119.7 | 0.0 |
| BG16 | 90 | 2 | 303 | 4105.4 | < 1 | 16.6 | 3580.2 | < 1 | 1.20 | 3572.9 | < 1 | 1.01 | 3537.90 | 99.2 | 0.0 |
| BG17 | 90 | 5 | 300 | 1515.5 | < 1 | 19.3 | 1298.1 | < 1 | 1.98 | 1290.4 | < 1 | 1.35 | 1273.70 | 173.4 | 0.0 |
| BG18 | 90 | 7 | 305 | 1055 | < 1 | 17.9 | 915.7 | < 1 | 2.27 | 911.4 | < 1 | 1.82 | 895.60 | 198.4 | 0.0 |
| BG19 | 105 | 2 | 367 | 5188.7 | < 1 | 14.9 | 4577.2 | < 1 | 1.22 | 4568.3 | < 1 | 1.03 | 4522.20 | 167.5 | 0.0 |
| BG20 | 105 | 5 | 360 | 1762.3 | < 1 | 19.3 | 1503.8 | < 1 | 1.60 | 1498.7 | < 1 | 1.22 | 1480.60 | 298.5 | 0.0 |
| BG21 | 105 | 7 | 360 | 1281 | < 1 | 17.7 | 1106.2 | < 1 | 1.53 | 1102.9 | < 1 | 1.21 | 1089.40 | 302.4 | 0.0 |
| BG22 | 120 | 2 | 405 | 5551.9 | < 1 | 16.1 | 4850.2 | < 1 | 1.16 | 4844.3 | < 1 | 1.02 | 4796.60 | 226.4 | 0.0 |
| BG23 | 120 | 5 | 407 | 2098.2 | < 1 | 20.5 | 1762.1 | < 1 | 1.08 | 1762.9 | < 1 | 1.13 | 1743.40 | 417 | 0.0 |
| BG24 | 120 | 7 | 409 | 1483.4 | < 1 | 17.8 | 1284.8 | < 1 | 1.82 | 1280.4 | < 1 | 1.42 | 1262.30 | 460.4 | 0.0 |
| BG25 | 135 | 2 | 459 | 6292.6 | < 1 | 16.7 | 5489.2 | < 1 | 1.08 | 5476.6 | < 1 | 0.85 | 5431.50 | 334.6 | 0.0 |
| BG26 | 135 | 5 | 459 | 2395.3 | < 1 | 18.6 | 2043.8 | < 1 | 1.18 | 2034.1 | < 1 | 0.69 | 2020.40 | 602.4 | 0.0 |
| BG27 | 135 | 7 | 461 | 1668 | < 1 | 18.5 | 1425.7 | < 1 | 1.08 | 1423.9 | < 1 | 0.92 | 1411.10 | 629.7 | 0.0 |
| BG28 | 150 | 2 | 508 | 6937 | < 1 | 17.0 | 6020.8 | < 1 | 1.09 | 6007.4 | < 1 | 0.87 | 5956.60 | 453.9 | 0.0 |
| BG29 | 150 | 5 | 509 | 2602.4 | < 1 | 19.4 | 2211 | < 1 | 1.18 | 2204.6 | < 1 | 0.90 | 2184.60 | 840.8 | 0.0 |
| BG30 | 150 | 7 | 515 | 1849.4 | < 1 | 18.1 | 1590.2 | < 1 | 1.38 | 1581.7 | < 1 | 0.83 | 1568.9 | 893.8 | 0.0 |
| Average | 83 | 5 | 281 | 2106.3 | < 1 | 17.1 | 1835.4 | < 1 | 2.66 | 1829.5 | < 1 | 2.15 | 1807.2 | 221.4 | 0 |

The average RPD over 300 problem instances indicates that the ACS and EST has the best performance over the existing SS heuristic and HH heuristic. The average RPD of ACS and ETS heuristic is noted as 0% and 2.15%, whereas the RPD value of SS heuristic and HH heuristic is noted as 2.66% and 17.1%, respectively.

The performance of ACS is better compared to the performance of EST, SS, and HH heuristics in all 120 problem instances. Another interesting observation about the CPU time of ACS can be made from Tables 5 and 6. The CPU time of ACS for Makespan objective is lower than the TCT objective. The similar trend is also observed for the CPU time of optimal solution for solving Makespan and TCT objective function. These results indicate that solving Makespan objective is easier than solving TCT objective.

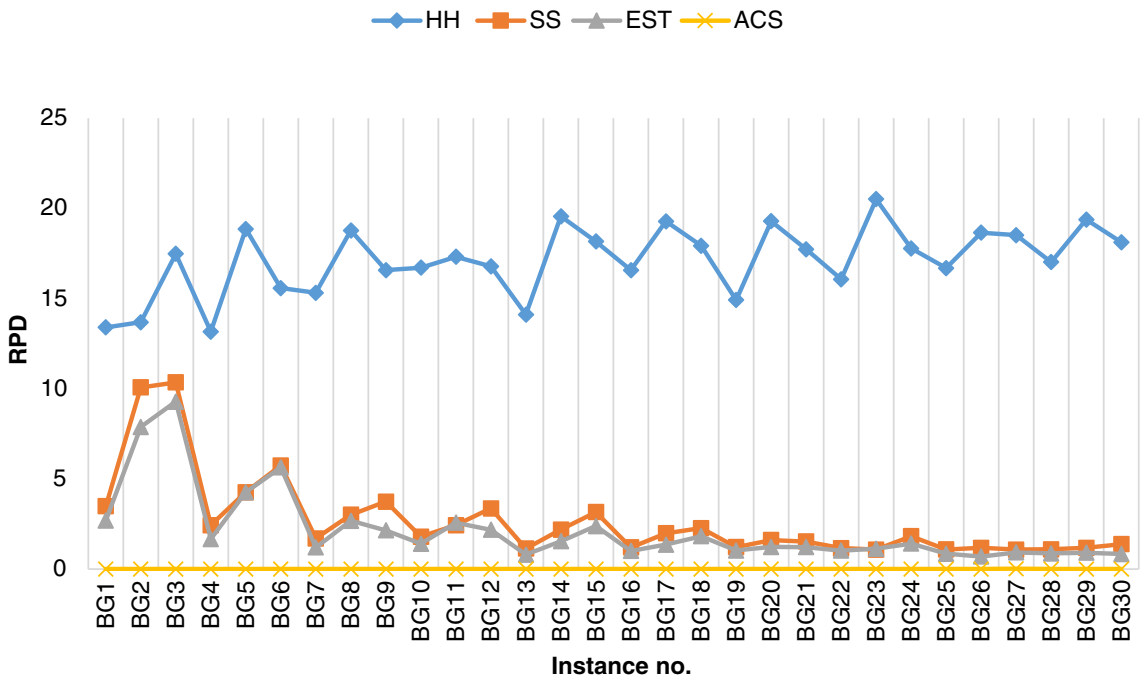


**FIGURE 9** RPD of algorithms with makespan objective function for large data set [Color figure can be viewed at wileyonlinelibrary.com]

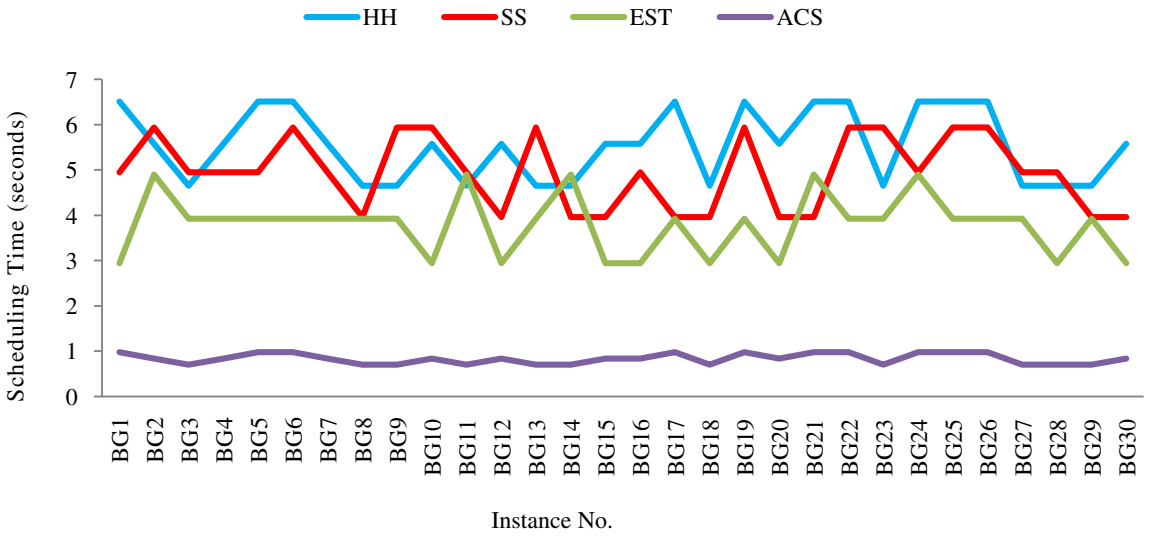


**FIGURE 10** Comparison of scheduling time for algorithms with large dataset [Color figure can be viewed at wileyonlinelibrary.com]

Figure 10 shows the comparison of scheduling time for algorithms with large dataset and ACS performs better than EST, SS, and HH. ACS has 16%, 18%, and 19% less than EST, SS, and HH, respectively. The reason behind better performance of ACS is the implementation of precedence task constraints during task scheduling.

## 6 | CONCLUSIONS AND FUTURE SCOPE

A cloud computing scheduling problem is considered in this article where jobs are processed in the parallel computing resources with precedence constraints. The tasks consist of many interdependent subtasks that can be processed in one

of the unrelated parallel computing resources. This kind of problem is considered as an NP-hard problem. For NP-hard problem, the heuristics and metaheuristics solutions become an obvious choice as heuristics and metaheuristics promise a suitable solution method. In this article, two existing heuristic HH and SS are considered as most relevant heuristics for this specific problem domain. The existing heuristic did not utilize the scheduling algorithm property of unrelated parallel machine scheduling problem. The SS heuristic use earliest available machine assignment rule. This paper shows that the earliest available machine assignment rule is non dominant for the unrelated parallel machine problem with precedence constraints. This article proposes an EST heuristic and an ACS metaheuristic, which utilize the list scheduling algorithm property of the problem. The experimental results reveal the superior performance of proposed EST heuristic over existing heuristics. The use of list scheduling algorithm helps proposed heuristic to perform better than existing heuristic. The experimental results also indicate that solving Makespan objective is easier than solving TCT objective. The performance of ACS is found to be the best for minimizing the Makespan objective as well as minimizing the total completion time objective and scheduling time. ACS shows the promising results which will help the CCSPs to render the quality services for its end user cloud service users.

Future directions include implementation and testing in a distributed Fog-Cloud setup using FogBus framework provided by Tuli et al.[63] To achieve this, data sharing techniques need to be tested to ensure that the ACS algorithm works seamlessly in a distributed setup. We also propose to extend this work to more sophisticated environments wherein we consider the myriad of factors crucial in a large-scale cloud/grid setup. Such factors include geographic distance, costs of the machine, network bandwidth for communication, resource utilization, and so on. We will explore scalability and model limitations of proposed work in the future. Further research directions include integration of ensemble methods[64] to achieve low scheduling times and explore other optimization methods like the ones using Pareto efficiency to test the robustness of the proposed algorithms.

## ACKNOWLEDGEMENTS

We thank Prof. Satish N. Srirama (Editor) and anonymous reviewers for their valuable comments and suggestions for improving our research paper. This research is partially supported by NSERC discovery Grant 318689.

## ORCID

*Sukhpal Singh Gill* https://orcid.org/0000-0002-3913-0369

## REFERENCES

1. Khiat A, Tari A, Guérout T. MFHS: a modular scheduling framework for heterogeneous system. *Softw Pract Exp*. 2020;50:1463-1497. https://doi.org/10.1002/spe.2827.
2. Wainer G, Moallemi M. Designing real-time systems using imprecise discrete-event system specifications. *Softw Pract Exp*. 2020;50:1327-1344. https://doi.org/10.1002/spe.2831.
3. Gill SS, Tuli S, Xu M, et al. Transformative effects of IoT, blockchain and artificial intelligence on cloud computing: evolution, vision, trends and open challenges. *Internet Things*. 2019;8:100118.
4. Kaur A, Singh VP, Gill SS. The future of cloud computing: opportunities, challenges and research trends. Paper presented at: 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud). Palladam, India: IEEE; 2019:213–219.
5. Kochan CG, Nowicki DR, Sauser B, Randall WS. Impact of cloud-based information sharing on hospital supply chain performance: a system dynamics framework. *Int J Prod Econ*. 2018;195:168-185.
6. Buyya R, Srirama SN, Casale G, et al. A manifesto for future generation cloud computing: research directions for the next decade. *ACM Comput Surv (CSUR)*. 2018;51(5):1-38.
7. Islam MT, Srirama SN, Karunasekera S, Buyya R. Cost-efficient dynamic scheduling of big data applications in apache spark on cloud. *J Syst Softw*. 2020;162:110515.
8. Enterprise Cloud Computing Survey. https://clutch.co/cloud#survey. Accessed June 9, 2017.
9. The Changing Faces of the Cloud. http://www.bain.com/Images/BAIN_BRIEF_The_Changing_Faces_of_the_Cloud.pdf. Accessed June 9, 2017.
10. Zhou B, Dastjerdi AV, Calheiros RN, Srirama SN, Buyya R. mCloud: a context-aware offloading framework for heterogeneous mobile cloud. *IEEE Trans Serv Comput*. 2015;10(5):797-810.
11. Casini D, Biondi A, Buttazzo G. Timing isolation and improved scheduling of deep neural networks for real-time systems. *Softw Pract Exp*. 2020;50:1760-1777. https://doi.org/10.1002/spe.2840.
12. Tuli S, Tuli S. AVAC: a machine learning based adaptive RRAM variability-aware controller for edge devices. IEEE International Symposium on Circuits and Systems (ISCAS), Barceló Sevilla Renacimiento, Seville, Spain; 2020. https://arxiv.org/abs/2005.03077.
13. Germain-Renaud C, Rana O. The convergence of clouds, grids, and autonomics. *IEEE Internet Comput*. 2009;13(6):9.
14. Gill SS, Tuli S, Toosi AN, et al. ThermoSim: deep learning based framework for modeling and simulation of thermal-aware resource management for cloud computing environments. *J Syst Softw*. 2020;166:110596.

15. Malik SUR, Akram H, Gill SS, Pervaiz H, Malik H. EFFORT: energy efficient framework for offload communication in mobile cloud computing. *Softw Pract Exp – Wiley*. 2020;1-14. https://doi.org/10.1002/spe.2850.
16. Srirama SN, Ostovar A. Optimal cloud resource provisioning for auto-scaling enterprise applications. *Int J Cloud Comput*. 2018;7(2):129-162.
17. Li j, Qiu M, Mingb Z, Quan G, Qin X, Gu Z. Online optimization for scheduling preemptable tasks on IaaS cloud systems. *J Parall Distrib Comput*. 2012;72(2012):666-677.
18. Lei D, Liu M. An artificial bee colony with division for distributed unrelated parallel machine scheduling with preventive maintenance. *Comput Ind Eng*. 2020;141:106320.
19. Vallada E, Villa F, Fanjul-Peyro L. Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem. *Comput Oper Res*. 2019;111:415-424.
20. Ezugwu AE. Enhanced symbiotic organisms search algorithm for unrelated parallel machines manufacturing scheduling with setup times. *Knowl-Based Syst*. 2019;172:15-32.
21. Tsai JT, Fang J, Chou JH. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. *Comput Oper Res*. 2013;40:3045-3055.
22. Bhatt A, Dimri P, Aggarwal A. Self-adaptive brainstorming for jobshop scheduling in multicloud environment. *Softw Pract Exp*. 2020;50:1381-1398. https://doi.org/10.1002/spe.2819.
23. Gill SS, Buyya R. Resource provisioning based scheduling framework for execution of heterogeneous and clustered workloads in clouds: from fundamental to autonomic offering. *J Grid Comput*. 2019;17(3):385-417.
24. Fang Y, Wang F, Ge J. A task scheduling algorithm based on load balancing in cloud computing. Paper presented at: the International Conference on Web Information Systems and Mining, Chengdu, China; 2010.
25. Dutta D, Joshi R. A genetic: algorithm approach to cost-based multi-QoS job scheduling in cloud computing environment. Paper presented at: the Proceedings of the International Conference & Workshop on Emerging Trends in Technology, Mumbai, Maharashtra, India; 2011.
26. Jang SH, Kim TY, Kim JK, Lee JS. The study of genetic algorithm-based task scheduling for cloud computing. *Int J Control Automat*. 2012;5(4):157-162.
27. Liu J, Luo X-G, Zhang X-M, Zhang F, Li B-N. Job scheduling model for cloud computing based on multi-objective genetic algorithm. *Int J Comput Sci Issue*. 2013;10(1):134-139.
28. Li J-F, Peng J. Task scheduling algorithm based on improved genetic algorithm in cloud computing environment. *Jisuanji Yingyong/J Comput Appl*. 2011;31(1):184-186.
29. Pandey S, Wu L, Guru SM, Buyya R. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. Paper presented at: the 24th IEEE International Conference on the Advanced Information Networking and Applications (AINA), Perth, Western Australia; 2010.
30. Cook SA. The complexity of theorem-proving procedures. Paper presented at: the Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio; 1971.
31. Garey MR, Johnson DS. *Computers and intractability*. Vol. 174. San Francisco: Freeman; 1979.
32. Luis F-P, Ruiz R. Iterated greedy local search methods for unrelated parallel machine scheduling. *Eur J Oper Res*. 2010;207(2010):55-69.
33. Lin YK, Pfund ME, Fowler JW. Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Comput Oper Res*. 2011;38(6):901-916.
34. Lin YK, Pfund ME, Fowler JW. Multiple-objective heuristics for scheduling unrelated parallel machines. *Eur J Oper Res*. 2013;227(2013):239-253.
35. Jose EC, Joseph Y, Leung T. Scheduling unrelated parallel batch processing machines. *Comput Oper Res*. 2017;78:117-128.
36. Shahvari O, Logendran R. An enhanced tabu search algorithm to minimize a bi-criteria objective in batching and scheduling problems on unrelated-parallel machines with desired lower bounds on batch sizes. *Comput Oper Res*. 2017;77:154-176.
37. Joo CM, Kim BS. Hybrid genetic algorithms with dispatching rules for unrelated parallel machine scheduling with setup time and production availability. *Comput Ind Eng*. 2015;85:102-109.
38. Cheng CY, Huang LW. Minimizing total earliness and tardiness through unrelated parallel machine scheduling using distributed release time control. *J Manuf Syst*. 2017;42:1-10.
39. Diana ROM, de França Filho MF, de Souza SR, de Almeida Vitor JF. An immune-inspired algorithm for an unrelated parallel machines' scheduling problem with sequence and machine dependent setup-times for makespan minimisation. *Neurocomputing*. 2015;163(2015):94-105.
40. Luis F-P, Perea F, Ruiz R. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *Eur J Oper Res*. 2017;260(4):82-93.
41. Oleh S, Lars M. Heuristic approaches for scheduling jobs in large-scale flexible job shops. *Comput Oper Res*. 2016;68:97-109.
42. Wang W, Cheng Yu. Optimal charging scheduling for electric vehicles considering the impact of renewable energy sources. Paper presented at: 2020 5th Asia Conference on Power and Electrical Engineering (ACPEE). Chengdu, China: IEEE; 2020:1150–1154.
43. Deng R, Luo F, Ranzi G, Zhao Z, Yan X.. A MILP based two-stage load scheduling approach for building load's peak-to-average ratio reduction. Paper presented at: 2020 5th Asia Conference on Power and Electrical Engineering (ACPEE). Chengdu, China: IEEE; 2020:771–775.
44. Herrmann J, Proth M, Sauer N. Heuristics for unrelated machine scheduling with precedence constraints. *Eur J Oper Res*. 1997;102:528-537.

45. Liu C, Yang S. A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints. *J Softw*. 2011;6(6):1146-1153.
46. Afzalirad M, Rezaeian J. Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Comput Ind Eng*. 2016;98(2016):40-52.
47. Gacias B, Artigues C, Lopez P. Parallel machine scheduling with precedence constraints and setup times. *Comput Oper Res*. 2010;37(12):2141-2151.
48. Tuli S, Ilager S, Ramamohanarao K, Buyya R. Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks. *IEEE Transactions on Mobile Computing, 10.1109/TMC.2020.3017079*. 2020. http://buyya.com/papers/DynSchedulingEdgeCloudNets.pdf.
49. Hunkeler U., Truong HL, Stanford-Clark A. MQTT-S—A publish/subscribe protocol for wireless sensor networks. Paper presented at: 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08). Bangalore, India: IEEE; 2008:791–798.
50. Hurink J, Knust S. List scheduling in a parallel machine enviornment with precedence constraints and setup times. *Oper Res Lett*. 2001;29:231-239.
51. Riccardo Mancini, Shreshth Tuli, Tommaso Cucinotta, and Rajkumar Buyya, iGateLink: A Gateway Library for Linking IoT, Edge, Fog and Cloud Computing Environments, Proceedings of the International Conference on Intelligent and Cloud Computing (ICICC-2019, Springer, Germany), Bhubaneswar, India, December 16-17, 2019.
52. Arnaout J-P, Rabadi G, Musa R. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *J Intell Manuf*. 2010;21(6):693-701.
53. Behnamian J, Zandieh M, Ghomi SF. Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm. *Expert Syst Appl*. 2009;36(6):9637-9644.
54. Gao Y, Guan H, Qi Z, Hou Y, Liu L. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *J Comput Syst Sci*. 2013;79(8):1230-1242.
55. Hua X-y, Zheng J, Hu W-x. Ant colony optimization algorithm for computing resource allocation based on cloud computing environment. *J East China Norm Univ Nat Sci*. 2010;1(1):127-134.
56. Gajpal Y, Rajendran C. An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops. *Int J Prod Econ*. 2006;101(2):259-272.
57. Zhang S, Gajpal Y, Appadoo SS, Abdulkader MM. Electric vehicle routing problem with recharging stations for minimizing energy consumption. *Int J Prod Econ*. 2018;203:404-413.
58. Thiruvady D, Singh G, Ernst AT, Meyer B. Constraint-based ACO for a shared resource constrained scheduling problem. *Int J Prod Econ*. 2013;141:230-242.
59. Ting CJ, Chen CH. A multiple ant colony optimization algorithm for the capacitated location routing problem. *Int J Prod Econ*. 2013;141:34-44.
60. Thepphakorn T, Pongcharoen P, Hicks C. An ant colony based timetabling tool. *Int J Prod Econ*. 2014;149:131-144.
61. Hong J, Diabat A, Panicker VV, Rajagopalan S. A two-stage supply chain problem with fixed costs: an ant colony optimization approach. *Int J Prod Econ*. 2018;204:214-226.
62. Stützle T, Hoos HH. Max-min ant system. *Futur Gener Comput Syst*. 2000;16(8):889-914.
63. Tuli S, Mahmud R, Tuli S, Buyya R. Fogbus: a blockchain-based lightweight framework for edge and fog computing. *J Syst Softw*. 2019;154:22-36.
64. Tuli S, Basumatary N, Gill SS, Kahani M, Arya RC, Wander GS, Buyya R. Health Fog: An ensemble deep learning based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in integrated IoT and fog computing environments. *Future Generation Computer Systems*. 2020;104:187–200. http://dx.doi.org/10.1016/j.future.2019.10.043.


**How to cite this article:** Bhardwaj AK, Gajpal Y, Surti C, Gill SS. HEART: Unrelated parallel machines problem with precedence constraints for task scheduling in cloud computing using heuristic and meta-heuristic algorithms. *Softw: Pract Exper*. 2020;1–21. https://doi.org/10.1002/spe.2890