Hindawi
Complexity
Volume 2020, Article ID 1385049, 13 pages
https://doi.org/10.1155/2020/1385049

Research Article

Nature-Inspired Metaheuristics for Two-Agent Scheduling with Due Date and Release Time

Hongwei Li,[1] **Yuvraj Gajpal**,[1] **Chirag Surti**,[2] **Dongliang Cai**,[3] **and Amit Kumar Bhardwaj**[4]

[1]*Department of Supply Chain Management, I.H. Asper School of Business, University of Manitoba, Winnipeg R3T 5V4, Manitoba, Canada*
[2]*Department of Information System, Analytics and Supply Chain Management College of Business, Rider University, 2 083 Lawrenceville Rd, Lawrenceville 08648, NJ, USA*
[3]*School of Finance, Southwestern University of Finance and Economics, Chengdu 610000, Sichuan, China*
[4]*L.M. Thapar School of Management, Thapar Institute of Engineering & Technology, Dera Bassi Campus, Patiala, Punjab, India*

Correspondence should be addressed to Dongliang Cai; caidl@swufe.edu.cn

Received 29 July 2020; Revised 14 November 2020; Accepted 9 December 2020; Published 29 December 2020

Academic Editor: Wei Zhou

This paper delves into a two-agent scheduling problem in which two agents are competing for a single resource. Each agent has a set of jobs to be processed by a single machine. The processing time, release time, weight, and the due dates of each job are known in advance. Both agents have their objectives, which are conflicting in nature. The first agent tries to minimize the total completion time, while the second agent tries to minimize the number of tardy jobs. The two agents' scheduling problem, an NP-hard problem, has a wide variety of applications ranging from the manufacturing industry to the cloud computing service provider. Due to the wide applicability, each variation of the problem requires a different algorithm, adapted according to the user's requirements. This paper provides mathematical models, heuristic algorithms, and two nature-based metaheuristic algorithms to solve the problem. The algorithm's performance was gauged against the optimal solution obtained from the AMPL-CPLEX solver for both solution quality and computational time. The outlined metaheuristics produce a solution that is comparable with a short computational time. The proposed metaheuristics even have a better solution than the CPLEX solver for medium-size problems, whereas the computation times are much less than the CPLEX solvers.

1. Introduction

The scheduling problems have always been an intriguing topic for scholars because of their wide range of applications. The academic community has extensively explored scheduling problems as they are extensively utilized in manufacturing and service industries to optimize resource allocation. Often, scheduling is a critical process as it influences the productivity of an organization. The scheduling problem has a wide range of applications in different industries and different environments. For example, in the engineering discipline, the scheduling problem is used for disassembly sequence to maximize revenue (Feng et al. [1]; Feng et al. [2]), for scheduling multicluster tools for wafer fabrication in the process industry (Zhu et al. [3]). The machine-scheduling problem is mainly motivated by the manufacturing industry. The usual scheduling problem involves finding a sequence of jobs in a machine, flow of jobs through different machines, or assigning jobs to different machines.

The machine-scheduling problem involves the allocation of resources to process different jobs under different machine settings such as single-machine setting [4], flowshop machine setting [5], job shop machine setting [6], and parallel machine setting [7]. The classical machine-scheduling involves the processing of different jobs for a single customer/agent. When the jobs belong to different customers, the resultant scheduling problem is a multiagent scheduling problem.

The research on the multiagent scheduling models started approximately fourteen years ago. An application of multiple agents scheduling problem can be encountered in many industries such as aircraft landing problem [8], railroad track allowance problem [9], networks [10], and cloud computing services [7]. Two-agent scheduling problem is a special case of a multiagent scheduling problem.

Agnetis et al. [11] and Baker and Smith [4] started a new branch in scheduling problems in which two agents use the same resource for processing jobs. Each agent has its objective function to be optimized. Two-agent scheduling problems can have the special challenge of optimizing two different objective functions belonging to two different agents. Two measures were proposed by Suresh and Chaudhuri [12]. The first measure assigns a weight to each agent's objective function to convert the multiobjective problem to a single objective problem. The second measure minimizes the first agent's objective function but tries to keep the objective function of the second agent under some predefined limit. Cheng et al. [13] named these two types of multiagent scheduling problems as the "minimality model" and "feasibility model," respectively. In this paper, we follow the second research direction and study the feasibility model, where the objective function of the first agent is minimized, and the objective function of the second agent is kept under some limit. This paper aims to provide a mathematical formulation of the model and develop a metaheuristic to find a good quality solution.

Although many two-agent scheduling models have been studied in recent years, the studies related to the release date-related two-agent scheduling model are relatively limited [14]. The number of tardy jobs minimization is aimed at only a few papers. Informed about the two-agent scheduling with due date literature, we consider a series of two-agent scheduling problems with due date-related objective in this paper. We have defined the single-machine-scheduling problems with unequal job release date constraints to minimize the total completion time, the total weighted completion time, and the number of tardy job objectives.

We have considered the scheduling problems of a two-agent single machine which is donated as $1|\sum U^B \le Q, r|\sum w^A C^A$ (Problem 1), $1|\sum U^B \le Q, r|\sum C^A$ (Problem 2), $1|\sum U^B \le Q|\sum w^A C^A$ (Problem 3), and $1|\sum U^B \le Q|\sum C^A$ (Problem 4) with the following assumptions, where A and B represent two agents in the problem:

(i) The jobs from the two agents have a real release time

(ii) The jobs from agent A have positive weight

(iii) The jobs from agent B have a positive due date for all four problems

(iv) The job will not be released until the shared machine fully processes it

(v) The jobs from the two agents have a nonzero processing time

(vi) Preemption is not allowed

The paper contributes to the scheduling literature in many ways. In our knowledge, this paper first time develops algorithms to solve the two-agent scheduling problem to minimize the weighted completion and number of tardy job objectives. A mathematical model based on MILP formulation is developed to find the optimal solution for the problem. Also, this paper investigates two nature-based metaheuristics to solve the problem. The novel features of the proposed metaheuristics are the hybridization of the algorithm with local search schemes. The proposed local search scheme not only improves the solution quality but also obtains a feasible solution.

The remainder of the paper is organized as follows. Section 2 investigates the single machine-scheduling literature. Section 3 addresses the problem definition associated with the mathematical model. Section 4 introduces the proposed ant colony optimization (i.e., ACO) and intelligent water drop (IWD) algorithm. Section 5 provides the numerical experiment results, related analysis, and a comparison of the proposed ACO algorithm with the optimal solution generated by the AMPL software. Section 6 outlines the conclusion and suggests future research directions.

2. Literature Review

Since the publication of Baker and Cole Smith [4] and Agnetis et al. [11], many scholars have explored a two-agent single-machine-scheduling problem. Two-agent scheduling problem considers two agents in which each agent has a set of jobs for processing in a single-machine. Generally speaking, the research of a single-machine and two-agent scheduling models could be classified into two main categories. The first category is called a minimality model, and the second category is called a feasibility model. Each research area has further research directions that can be classified based on their features. These features include different objective functions of each agent, release date, weight, the nature of processing time, due date, and among other constraints.

2.1. Two-Agent Scheduling Problem with the Number of Tardy Job Objectives. Agnetis et al. [11] investigated three due date criteria based on two-agent scheduling models to minimize the number of tardy jobs. These problems are denoted by $1|\sum U_j^B \le Q|\sum U_i^A$ (P1), $1|\sum U_j^B \le Q|\sum C_i^A$ (P2), and $1|\sum U_j^B \le Q|\sum w_i^A C_i^A$ (P3). They have proven the computational complexity of the problems mentioned above. The first problem could be solved in $O(n^3)$ time while the complexity of the second problem is proved to be a binary NP-hard. The complexity of the second problem is still unknown. Ng et al. [15] provided many lemmas for$1|\sum U_j^B \le Q|\sum C_i^A$ problem: the scheduling of all A-type jobs is done in the shortest processing time (SPT) rule and the scheduling of all B-type jobs is done in the earliest due date first rule (EDD) following an optimal schedule. Furthermore, they developed a pseudopolynomial-time algorithm to solve the problem of searching for an optimal solution. Ng et al. [15] studied the complexity of several two-agent scheduling problems to minimize tardy jobs. They also

pointed out that the problem is strongly NP-hard if the upper bound is equal to zero.

Leung et al. [16] referred the Agnetics et al.'s [11] work and proved the complexity of a two-agent scheduling problem to minimize the total tardiness objective of the first agent while limiting the total number of tardy jobs of the second agent within a prespecified upper limit (i.e., $1|\sum U_j^B \le Q|\sum T_j^A$) is binary NP-hard. Lee et al. [17] extend Ng et al.'s [15] problem (i.e., $1|\sum U_j^B = 0|\sum w_i^A C_i^A$), considering an actual processing time approach. They proposed a branch-and-bound algorithm and heuristics to solve the problem. Yin et al. [18] studied the single-machine two-agent scheduling problem with release times and due date for minimizing the number of tardy jobs of first agent while keeping the maximum lateness of the jobs of the other agent within prespecified value. Yuan [19] studied the multiagent scheduling problem to minimize the weighted number of tardy jobs from each agent. Yin et al. [20] studied the two-agent scheduling problem for determining the due dates. The literature studies other variants of the two-agent scheduling problem. However, they differ from the problem considered in this paper in terms of either objective function (Wan et al. [21]; Mor and Mosheiov [22]) or machine settings (Mor and Mosheiov [23]; Lei [24]; Yu et al. [25]). A literature survey by Li et al. [26] and Yin et al. [27] can be referred to find different variants of the two-agent scheduling problem with due date-related objectives.

The articles by Cheng et al. [14]; Yin et al. [28]; and Chen et al. [29] are closely related to the problem considered in our paper. Cheng et al. [14] considered the two-agent scheduling problem with total weighted completion time and number of tardy jobs minimization. Although Cheng et al. [14] used the same objective functions considered in this paper, they followed the minimality model and minimized the two objective functions' weighted sum. Yin et al. [28] proposed a two-agent scheduling model for unrelated parallel machine-scheduling problem to minimize the number of total completion time and the number of tardy jobs criteria. They proposed a column generation method embedded with branch-and-price algorithms to find the optimal solution. The complexity of the problem considered in this paper is proved to be NP-hard by Chen et al. [29]. Their research work was based on proving the complexity, and they did not provide an algorithm to solve the problem. Motivated by Chen et al.'s [29] complexity, this paper proposes a mixed linear integer programming (MILP) formulation and a metaheuristic algorithm to solve the problem.

2.2. Nature-Inspired Algorithm. This paper uses two nature-inspired algorithms to solve the problem under consideration. The foraging behavior of real ants inspires the ACO while the flow of water drops inspires the IWD. Nature-inspired algorithms have been used to solve a wide variety of scheduling problems. Feng et al. [1] used ACO to solve the disassembly problem for CNC machine tools. Decision-making involves finding the sequence of disassembly operations to maximize demand satisfaction and minimize disassembly cost simultaneously. Zhang et al. [30] used ACO for cross docks operations in which decision-making involves finding the assignment of receiving and shipping doors. Jia et al. [31] used a bee colony algorithm to solve bike repositioning in bike-sharing systems in which decision-making involves finding the routes of the repositioning vehicles and the number of parked bikes of the corresponding station. Yagmahan and Yenisey [5] used ACO to solve the flowshop scheduling problem for finding a sequence of jobs. Li et al. [32] used an ant colony algorithm to solve two agents' scheduling problems to minimize total weighted completion time and makespan objectives. Hosseini [33] implements the IWD algorithm to solve the traveling salesman problem. Alijla et al. [34] used IWD to solve three combinatorial optimization problems; (1) rough set for subset feature selection (RSFS), (2) multiple knapsack problem (MKP), and (3) traveling salesman problem (TSP). The successful application of algorithms inspired by natural phenomena to solve different problems motivated us to use them to solve the two agents' scheduling problems considered in this paper.

3. Problem Definition and Notation

This section first describes the terminologies and notations used in this paper and then provides the linear programming formulation for the problem.

3.1. Problem Definition. In all four problems, we consider that we have n number of jobs from agent A and agent B that are to be scheduled at a single machine. Each of the A and the B jobs has a positive processing time and a positive release time. Additionally, A-type jobs also have a positive weight, and B-type jobs have a positive due date. The problems involve assigning jobs to the shared machine and then finding sequence for the assigned job so that the total weighted completion time or total completion time of A-type jobs is as minimum as possible while limiting the number of tardy jobs of B-type jobs within a prespecified limit. We use the following notations to describe and formulate the problems:

X: set of agents, $X = \{A, B\}$

n_x: the number of jobs for the agent $x \in X$

n: total number of jobs, $n = n_A + n_B$

N: set of all n jobs, $N = \{1, 2, 3, \ldots, n\}$

N_B: set of B-type jobs, consisting of n_B jobs, $N_B = \{n_A + 1, n_A + 2, \ldots, n\}$.

J^X: job set of the agent $x \in X$

p_i^x: the processing time of i^{th} job for the agent $x \in X$

r_i^x: release date of i^{th} job for the agent $x \in X$

w_i^x: weight of i^{th} job for the agent $x \in X$

δ_i^x: the density of i^{th} job for the agent $x \in X$, $\delta_i^x = w_i^x / p_i^x$

d_i^x: due date of i^{th} job for the agent $x \in X$

C_i^x: completion time of i^{th} job for the agent $x \in X$

σ: ordered set of jobs already scheduled

$f^x(\sigma)$ for a given sequence σ, the value of the objective function for the agent $x \in X$

Q: the upper bound, a constant number.

$bigM$: a large number

In this paper, we consider four problems, denoted as follows:

Problem 1: $1|\sum_{m=1}^{n_B} U_m^B \le Q, r|\sum_{l=1}^{n_A} w_l^A C_l^A$
Problem 2: $1|\sum_{m=1}^{n_B} U_m^B \le Q, r|\sum_{l=1}^{n_A} C_l^A$
Problem 3: $1|\sum_{m=1}^{n_B} U_m^B \le Q|\sum_{l=1}^{n_A} w_l^A C_l^A$
Problem 4: $1|\sum_{m=1}^{n_B} U_m^B \le Q|\sum_{l=1}^{n_A} C_l^A$

Problem 1 is a single-machine two-agent scheduling problem. This problem minimizes the total weighted completion time of A-type jobs subjected to an upper bound on the total number of tardy B-type jobs. It also has arbitrary release dates and arbitrary weights. Problem 2 minimizes the total completion time of A-type of jobs. Problem 3 and Problem 4 have the same objective functions as Problem 1 and Problem 2, respectively, but they do not consider release date. Problem 3 focuses on finding such a schedule that minimizes the total weighted completion time of A-type jobs and/or ensures that the maximum number of tardy B-type jobs does not exceed the upper bound Q. Problem 4 is an unweighted case of Problem 3. The objective of Problem 4 is to minimize the total completion time of A-type jobs.

3.2. NP-Hardness. Agnetis et al. [11] have proved that the problems $1|f_{\max}^B \le Q|\sum w_j^A C_j^A$ and $1|\sum U_j^B \le Q|\sum w_j^A C_j^A$ are binary NP-hard. They also reported that problem $1|\sum U_j^B \le Q|\sum C_j^A$ is still open. Therefore, Problem 1 is also NP-hard. Recently, Chen et al. [29] proved that problem $1|\sum_{m=1}^{n_B} U_m^B \le Q|\sum_{l=1}^{n_A} w_l^A C_l^A$ is NP-hard.

3.3. The General Mathematical Model of Problems. All four problems considered in this paper can be formulated in a common mathematical model. Problem 2 is an unweighted case of Problem 1. By putting weights of all jobs into one, Problem 1 is thereby reduced to Problem 2. By placing all jobs' release dates to zero, Problem 3 and Problem 4 can be reduced from Problem 1 and Problem 2, respectively. As a result, we can formulate a general mathematical model for all four problems.

We use the following decision variables to formulate the problem:

X_{ij}: a binary variable for assigning job i at position j, $i, j \in [1, n]$

Y_j: a binary variable for starting job at position j just after completion of the job at position $j-1$, $j \in [1, n]$

U_i: a binary variable for tardy B-type jobs, $i \in [n_A + 1, n]$

CT_i^1: completion time of job $i \in [1, n]$

CT_i^2: completion time of job at position j, $j \in [1, n]$

ST_j: starting time of job at position j, $j \in [1, n]$

The mixed-integer linear programming (MILP) formulation for the problem can be described as follows:

$$\text{minimize } Z = \sum_{i=1}^{n_A} \mathrm{CT}_i^1 \times w_i^A, \tag{1}$$

subject to

$$\sum_{j=1}^{n} X_{ij} = 1; \quad \forall i \in N, \tag{2}$$

$$\sum_{i=1}^{n} X_{ij} = 1; \quad \forall j \in N, \tag{3}$$

$$\mathrm{CT}_1^2 = \sum_{i=1}^{n} X_{i1} \times (r_i + p_i); \tag{4}$$

$$\mathrm{CT}_j^2 = \mathrm{ST}_j + \sum_{i=1}^{n} X_{i1} \times p_i; \quad \forall j \in N\ j \ge 2, \tag{5}$$

$$\mathrm{ST}_j \ge \mathrm{CT}_{j-1}^2, \quad \forall j \in N, \tag{6}$$

$$\mathrm{ST}_j \ge \sum_{i=1}^{n} X_{ij} \times r_i, \quad \forall j \in N\ j \ge 2, \tag{7}$$

$$\mathrm{ST}_j \le \mathrm{CT}_{j-1}^2 + \mathrm{big}M \times \left(1 - Y_j\right); \quad \forall j \in N\ j \ge 2, \tag{8}$$

$$\mathrm{ST}_j \le \sum_{i=1}^{n} X_{ij} \times r_i + \mathrm{big}M \times Y_j; \quad \forall j \in N\ j \ge 2, \tag{9}$$

$$\mathrm{CT}_i^1 \ge \mathrm{CT}_j^2 - \mathrm{big}M \times \left(1 - X_{ij}\right); \quad \forall i, j \in N, \tag{10}$$

$$\mathrm{CT}_i^1 \le \mathrm{CT}_j^2 + \mathrm{big}M \times \left(1 - X_{ij}\right); \quad \forall i, j \in N, \tag{11}$$

$$\mathrm{CT}_i^1 \le d_i + \mathrm{big}M \times U_i; \quad \forall i \in N_B, \tag{12}$$

$$\sum_{i=1}^{N_B} U_i \le Q; \tag{13}$$

$$\mathrm{CT}_i^1 \ge 0; \quad \forall i \in N, \tag{14}$$

$$\mathrm{CT}_j^2 \ge 0; \quad \forall j \in N, \tag{15}$$

$$\mathrm{ST}_j \ge 0; \quad \forall j \in N. \tag{16}$$

Equation (1) represents the minimization of the objective function of agent A. Constraint (2) and (3) make sure that every job is assigned to only one position of the machine. The completion time of the job at position j will be calculated using constraint (4) and constraint (5). In case a job is scheduled at the first position in the sequence, then constraint (4) is used to calculate the first job's completion time, while constraint (5) is used to calculate the rest of the jobs' completion times. Constraints (6)–(9) are used to determine

jobs' starting time. Constraint (6) ensures that any job will not be scheduled at any position before the previous job is completed. Constraint (7) states that any job will not be scheduled at any position before its release date. Constraints (8) and (9) ensure that the job's starting time is the same as the previous job's completion time. Constraint (10) and (11) calculates the completion time of job i. These constraints also ensure that the job i is scheduled at position j then $C_i^1 = C_j^2$. Constraint (12) is used to determine that if the processing of the job i is finished after its due date, then the job is marked as a tardy. Constraint (13) confirms that the total number of tardy B-type jobs is within the prespecified upper bound Q.

4. Proposed Algorithms

In a single agent scheduling problem, total weighted completion time is minimized by arranging the jobs in nonincreasing order of density. Simultaneously, the number of tardy jobs is minimized by arranging jobs in nondecreasing order of their due dates. These properties are used in the proposed heuristic.

4.1. Heuristic Algorithm. The heuristic first creates partial schedules σ_A and σ_B for A- and B-type of jobs. The jobs in the sequence σ_A are arranged in nonincreasing order of density δ_i^A. The jobs in the sequence σ_B are arranged in nonincreasing order of their due dates d_i^B. A complete solution $\sigma$is then constructed by first sequencing A jobs followed by B jobs. The resultant solution can be infeasible. If the infeasibility is encountered, A jobs are moved towards the end of the sequence to make the solution feasible. The pseudocode of the heuristic is presented below (Algorithm 1).

To understand the heuristic algorithm, consider an instance with 3 A-type of jobs and 3 B jobs. The parameters associated with 6 jobs are provided in Table 1. Consider upper bound Q for the total number of tardy jobs for agent B to be 2.

The heuristic starts with creating partial sequences $\sigma_A = \{J2 - J1 - J3\}$ and $\sigma_B = \{J5 - J6 - J4\}$. The initial solution is created by combining these two sequences $\sigma = \{J2 - J1 - J3 - J5 - J6 - J4\}$. The start time and completion time of each job for a sequence $\sigma$are shown in Table 2.

The total weighted completion time of agent A for the sequence σ is 398 (i.e., $U = 13 \times 8 + 27 \times 8 + 39 \times 2 = 398$. The total number of tardy jobs for agent B for the sequence σ is 3 (i.e., $U = 3$). Since $U > Q$, the sequence σ is infeasible. Thus, the algorithm calls the feasibility phase for obtaining a feasible solution. When $k = 3$, job $J3$ is selected for moving towards the end of the schedule. Job $J3$ is first reinserted at positions $l = 4$, to create $\sigma^{\text{temp}} = \{J2 - J1 - J5 - J3 - J6 - J4\}$ with $U^{\text{temp}} = 3$. Since the number of tardy jobs remains greater than Q, the job is now inserted at position $l = 5$ and then finally at position $l = 6$ to create a sequence $\sigma^{\text{temp}} = \{J2 - J1 - J5 - J6 - J4 - J3\}$. At the end of the l loop, the sequence $\sigma$is replaced by σ^{temp} to form the sequence $\sigma = \{J2 - J1 - J5 - J6 - J4 - J3\}$. When $k = 2$, job $J1$ is selected and inserted at positions $l = 3$, $l = 4$ and $l = 5$, and finally sequence σ is updated as $\sigma = \{J2 - J5 - J6 - J4 - J1 - J3\}$. When $k = 1$, job $J2$ is selected and inserted at position $l = 2$ to form the sequence $\sigma^{\text{temp}} = \{J5 - J2 - J6 - J4 - J1 - J3\}$ with $U^{\text{temp}} = 2$. At this time, l loop breaks because the total number of tardy jobs is less than or equal to Q. After the exit of the l loop, the sequence $\sigma$is replaced by σ^{temp}. Finally, the sequence $\sigma = \{J5 - J2 - J6 - J4 - J1 - J3\}$ is reported as the solution of the heuristic algorithm.

4.2. Ant Colony Optimization (ACO) Algorithm. Ant colony optimization (ACO) algorithm was designed by [35]. They took their inspiration to develop the algorithm from the foraging behavior of ants. The ants always use the shortest path while seeking food from their nest. The max-min ant system (MMAS) is employed in the proposed algorithm. The fundamental procedures of the proposed ACO algorithm are listed below, followed by a detailed description.

Step 1: generate an initial solution using the heuristic algorithm described in Section 4.1

Step 2: set parameters, trail intensity matrix τ_{ip}, trail intensities upper bound ($\tau_{\max}$) and the trail intensity lower bound ($\tau_{\min}$)

Step 3: follow the below steps and terminate this step until the condition of termination is not met

(i) Generate solutions using the trail intensities
(ii) Carry out the local search to improve the solution
(iii) Update the trail intensity
(iv) Update the best solution found so far

Step 4: report the best solution as a final solution of ACO

4.2.1. Parameter Initialization. The trail intensity is denoted by τ_{ip}, where τ_{ip} depicts the trail intensity of the scheduling job i in the position p. In this paper, the initial trail intensity is set $\tau_{ip} = 1/((1 - \rho) \times \text{IniObj})$. Here, IniObj represents the objective function value of the initial solution. The index ρ indicates the trail persistence.

In this paper, we take the max-min ant system (MMAS) into consideration. The MMAS is an improvement plan for the ACO algorithm, and it is designed for fast convergence [36]. In MMAS, the trail intensities are constrained by an upper limit and a lower limit (i.e., $\tau_{\min} \le \tau_{ip} \le \tau_{\max}$). In our proposed ACO algorithm, the lower limit of trail intensity is set to ($\tau_{ip}/200$), and the upper limit of trail intensity is set to $\tau_{ip} \times 200$.

4.2.2. Ant-Sequence Generation. We use ten ants to generate ten different solutions in the ant-sequence generation step. An ant solution is generated in the following two phases:

Trail intensities are used to determine the job, scheduled at position p, where $p \in [1, n]$. Jobs are selected from the first five unscheduled jobs in the set US from the best solution. First, the cumulative trail intensity $T_{ip} = \sum_{q=1}^{p} \tau_{ip}; \forall i \in US$ is calculated for each unscheduled

(1) /* Initial Solution */
(2) Create a partial sequence σ_A and σ_B
(3) Create a solution $\sigma \longleftarrow \{\sigma_A \cup \sigma_B\}$ //combine sequences σ_A and σ_B to construct σ
(4) Calculate $U \longleftarrow f^B(\sigma)$ //Calculate the weighted number of tardy job for sequence σ
(5) /* Feasibility phase */
(6) **for** $k = n_1$ to 1 **do**
(7) Select job $J = \sigma(k)$
(8) **for** $l = k+1$ to $n - (k - n_1)$ **do**
(9) $\sigma^{\text{temp}} \longleftarrow$ Create new sequence from σ by reinserting job J at position l
(10) $U^{\text{temp}} \longleftarrow f^B(\sigma^{\text{temp}})$ //Calculate the weighted number of tardy job for σ^{temp}
(11) **if** $U^{\text{temp}} \leq Q$**then**
(12) break l loop
(13) **end if**
(14) **end for**
(15) $\sigma \longleftarrow \sigma^{\text{temp}}; U \longleftarrow U^{\text{temp}}$
(16) **if** $U^{\text{temp}} \leq Q$ **then**
(17) break k loop
(18) **end if**
(19) **end for**
(20) **return solution** σ

Algorithm 1: Heuristic algorithm.

Table 1: A numerical example with 6 jobs.

Job	Agent	r_i^x	p_i^x	w_i^x	δ_i^x	d_i^x
J1	A	3	14	8	0.57	—
J2	A	1	12	8	0.67	—
J3	A	0	12	2	0.17	—
J4	B	3	6	—	0.17	62
J5	B	0	12	—	0.42	13
J6	B	21	6	—	0.50	16

Table 2: Completion time of sequence σ

Sequence (σ)	*J2*	*J1*	*J3*	*J5*	*J6*	*J4*
Start time	1	13	27	39	51	57
Completion time	13	27	39	51	57	63
Due date	—	—	—	13	16	62

job. One of the following three rules is used to select a job randomly. The first rule selects the first unscheduled job from the best sequence found so far. The second rule selects the job with the highest value of cumulative trail intensity T_{ip}. The third rule uses probability $\psi_{ip} = (T_{ip}/\sum_{k\in K} T_{kp})$to select jobs from the set *US*. The three rules are applied randomly with probabilities 0.4, 0.4, and 0.2, respectively.

4.2.3. Insertion Local Search. The local search procedure is employed to improve the quality of the ant-schedules generated by the last step and improve the solution quality. The local search is evaluation using a modified objective function $\overline{f}(\sigma) = f^A(\sigma) + \text{Big}M \times \max(0, Q - f^B(\sigma))$, where *BigM* is a large number. The modified objective function penalizes the objective function for infeasibility, which forces the solution to move towards feasibility during the search process. We use a simple insertion local search to improve the solution quality. In the local search, a job is rescheduled at all other positions. The best solution is updated if a better schedule is encountered during the process. The procedure is repeated for all the jobs one by one. Once the rescheduling of all the jobs at all the positions is completed, the whole procedure is repeated. The procedure is repeated until the modified objective function $\overline{f}(\sigma)$ keeps on improving.

4.2.4. Updating Trail Intensities. The trail intensities are updated based on the best sequence as well as the current sequence. Let $\text{Obj}_{\text{current}}$ and Obj_{best} denote the objective function value for the current sequence and the best sequence, respectively. L is the trail intensity deposit $L_{ip}^c$from the current sequence and the trail intensity deposit L_{ip}^* from the best sequence can be calculated using the following equations:

$$L_{ip}^{c} = \begin{cases} \dfrac{1}{\text{Obj}_{\text{current}}}, & \text{if job is in the } p^{\text{th}} \text{ position of current sequence,} \\ 0, & \text{otherwise,} \end{cases}$$
$$L_{ip}^{*} = \begin{cases} \dfrac{1}{\text{Obj}_{\text{best}}}, & \textit{if} \text{ job is in the } p^{\text{th}} \text{ position of best sequence,} \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

Using the abovementioned expressions, the trail intensities are updated by the following equations:

$$\tau_{ip}^{\text{new}} = \rho \times \tau_{ip}^{\text{old}} + L_{ip}^{c} + L_{ip}^{*}. \tag{18}$$

In this paper, ACO is stopped after 100 iterations. We use the values available in the literature to set different parameters for ACO. The trail persistence factor ρ is set to 0.95.

4.3. Intelligent Water Drop (IWD) Algorithm. The intelligent water drop (IWD) algorithm is a swarm-based method used for solving combinatorial optimization problems [33, 34]. In nature, we often observe that the water takes the path of least resistance, which translates into a water flow, such as a river, choosing the shortest path to their final destination, such as a lake or the sea. The IWD algorithm is based on this natural phenomenon of water flow in a river and its impact on the river bed's soil. The velocity of the water drops and the amount of the soil on the way mainly affect the river's flow. The IWD uses these two properties (soil and velocity) to find the combinatorial optimization problem. We consider that the water flows from the first position to the n$^{\text{th}}$ position through different jobs while solving a scheduling problem. This paper adopts the IWD algorithm used by Alijla et al. [34], which mainly uses the following variables:

$\text{soil}(i, j)$: amount of soil carried at position i, if water flows through job j

soil^k: amount of soil carried by water drop k

$\text{vel}^k(t)$: the velocity of water drop k at time t

iwd: number of water drops

The fundamental procedures of the IWD algorithm are listed below, followed by a detailed description.

Step 1: initialize variables $\text{soil}(i, j)$, soil^k, and $\text{vel}^k(t)$

Step 2: follow the below steps and terminate this step until the termination condition is not met

(i) Generate the solution using the principle of IWD for all iwd water drops
(ii) Carry out the local search to improve the solution for all iwd water drops
(iii) Update the best solution found so far
(iv) Update the global soil

Step 3: report the best solution as a final solution of ACO

The IWD starts with the initialization of variables. We use the values proposed by Alijla et al. [34] to initialize the variables and to set the parameter values. The variables are initialized as follows:

$$\begin{aligned} \text{soil}(i, j) &= 1000, \\ \text{soil}^k &= 4, \\ \text{vel}^k &= 100. \end{aligned} \tag{19}$$

4.3.1. Solution Construction. A solution of a single-machine-scheduling problem consists of finding the jobs for each position of the sequence. A water drop k starts with the first position and moves towards the n^{th} position. The sequence of jobs determines the path of the water drop. The IWD algorithm selects the job j from the unscheduled job set V_{usj}^k for sequencing at position i using the following probability distribution formula:

$$p_i^k(j) = \frac{f(\text{soil}(i, j))}{\sum_{l \in V_{usj}^k} f(\text{soil}(i, l))}, \tag{20}$$

where

$$f(\text{soil}(i, j)) = \frac{1}{0.01 + g(\text{soil}(i, j))},$$
$$g(\text{soil}(i, j)) = \begin{cases} \text{soil}(i, j), & \text{if } \min\limits_{l \in V_{usj}^k} (\text{soil}(i, l)) \geq 0, \\ \text{soil}(i, j) - \min\limits_{l \in V_{usj}^k} (\text{soil}(i, l)), & \text{else.} \end{cases} \tag{21}$$

After selecting the job j for sequencing at position i, the following variables are updated:

$$\begin{aligned}
\mathrm{vel}^k(t+1) &= \mathrm{vel}^k(t) + \frac{a_v}{b_v + c_v \times \mathrm{soil}(i,j)},\\
\Delta\mathrm{soil}(i,j) &= \frac{a_s}{b_s + c_s \times (1/\mathrm{vel}^k(t+1))},\\
\mathrm{soil}^k &= \mathrm{soil}^k + \Delta\mathrm{soil}(i,j),\\
\mathrm{soil}(i,j) &= (1-\rho_n) \times \mathrm{soil}(i,j) - \rho_n \times \Delta\mathrm{soil}(i,j).
\end{aligned} \tag{22}$$

Here, $a_v, b_v, c_v, a_s, b_s, c_s$, and ρ_n are the parameters. We set the value of these parameters according to the value set by Alijla et al. [34] as 1, 0.01, 1, 1, 0.01, 1, and 0.9.

Once the complete solution of a water drop is built, the solution is improved using the local search method described in Subsection 4.2.3.

4.3.2. Updating the Global Soil. The global soil updating step is referred to as a reinforcement phase. In this step, global soil is updated using the local best solution found among *iwd* solutions. Let $q(T^{lB})$represent the objective function of the local best solution. The global soil value is updated as follows:

$$\mathrm{soil}(i,j) = \left(1+\rho_{iwd}\right) \times \mathrm{soil}(i,j) - \rho_{iwd} \times \mathrm{soil}_{lb}^k \times \frac{1}{q\left(T^{lb}\right)}, \tag{23}$$

where ρ_{iwd} is a parameter value set as 0.90 according to the value set by Alijla et al. [34].

4.3.3. Termination Phase. In every iteration, *iwd* number of solutions are created and improved using local search schemes. After generating the solutions, global soil is updated. These two processes are repeated until a prespecified number of iteration, and finally, the best solution found so far is reported. The structure of IWD is similar to the structure of ACO. To make the two algorithms comparable, we used the same local search schemes and ran the algorithm for the same number of prespecified iterations. Thus, the IWD algorithm is stopped after 100 iterations for both IWD and the ACO.

5. Numerical Results Analysis

The performance of the proposed algorithm is evaluated using both small and large size problem instances. The results obtained by the proposed ant colony optimization algorithm are compared with the solutions generated by using the CPLEX solver. The linear programming-based mathematical model is solved by AMPL software with CPLEX solver. The AMPL is running on an iMac desktop with 3.3 GHz with 8 GB of RAM. The coding of the proposed algorithms is done in the C++ programming language and is implemented on AMD Opteron 2.3 GHz with 16 GB RAM.

In this paper, we generated 116 problems (4 problems $*$ 29 problem instances each). The smallest instance has 5 jobs for each agent, while the largest instance has 150 jobs. The jobs' processing time and weight are generated using a random number between [1, 15] and [1, 10], respectively. The upper bound value Q is generated as $Q = \alpha \times n_B$, where α was assigned randomly between 0.4 and 0.6.

The due date of each job is generated in the uniform distribution range of $[0.1 \times \sum_{i=1}^{n} p_i, 0.9 \times \sum_{i=1}^{n} p_i]$. The value of the release date of each job is generated according to the formula, as $r_i = \max\{(\beta \times d_i - p_i), 0\}$. The experiment uses the following two types of datasets:

(1) Small and medium datasets are used to weigh the proposed algorithms' results and the CPLEX result

(2) Large datasets are used to evaluate the scalability of the computation time of the proposed algorithms

The small dataset represents those problem instances, which can be solved optimally by mathematical models within reasonable CPU time. Due to the complexity of the problems, the mathematical model was applied to solve only small- and medium-size problem instances in a specific time, and one hour was set as the time at which the mathematical model will obtain a solution. The developed mathematical model's optimal solution can be solved using the AMPL software with the CPLEX solver. The quality of results is evaluated by using relative percentage deviation (i.e., RPD). The formula to calculate RPD_k of the proposed algorithm or CPLEX result for the k^{th} problem instance is

$$\mathrm{RPD}_k = \frac{H_k - B_k}{B_k} \times 100\%. \tag{24}$$

Here, H_k represents the proposed metaheuristic solution in the k^{th} problem instance, and B_k represents the best algorithm solution in the k^{th} problem instance.

The following notations are used to report and evaluate the numerical results:

CPLEX: mathematical model solved by CPLEX solver

OBJ: absolute value of objective function generated by algorithms

RPD: relative percentage of deviation

CPU: running time of the proposed algorithm

ACO: ant colony algorithm

IWD: intelligent water drop algorithm

5.1. Numerical Experiment with a Small and Medium Dataset. We compare the projected ant colony optimization-based metaheuristic results with the result of CPLEX. Tables 3–6 show the solution and CPU time of the proposed algorithms and CPLEX result for Problem 1, Problem 2, Problem 3, and Problem 4, respectively. The results with * represent the optimal solution.

The proposed ACO and IWD algorithm can efficiently solve small problem instances and balance CPU time with result quality. It is clear from Tables 3–6 that the proposed ACO algorithm and the IWD algorithm can solve all small problem instances in minutes. The average RPD of the two algorithms is summarized and showed in Table 7.

Table 3: Numerical results of Problem 1 for small and medium problem instances.

Instance	n	CPLEX		ACO			IWD		
		OBJ	CPU	OBJ	CPU	RPD	OBJ	CPU	RPD
1	10	527*	0.38	527	0.07	0.00	527	0.07	0.00
2	12	902*	1.20	902	0.13	0.00	902	0.13	0.00
3	14	1382*	2.76	1382	0.23	0.00	1382	0.22	0.00
4	16	1105*	35.50	1105	0.34	0.00	1105	0.32	0.00
5	18	2451*	444.82	2574	0.41	5.02	2574	0.44	5.02
6	20	1885*	1777.12	1885	0.67	0.00	1885	0.61	0.00
7	22	3022	3600	3018	0.9	0.00	3018	0.87	0.00
8	24	3787	3600	3768	1.07	0.13	3763	1.07	0.00
9	26	3865	3600	3885	1.43	0.52	3865	1.49	0.00
10	28	4887	3600	4911	1.86	1.99	4815	1.81	0.00
11	30	3738	3600	3738	2.66	1.25	3692	2.33	0.00
12	32	4007	3600	3967	2.89	0.00	3967	2.92	0.00
13	34	5305	3600	5832	3.22	14.47	5095	3.42	0.00
14	36	6696	3600	6680	5.06	0.00	6680	4.57	0.00
15	38	5603	3600	5685	5.98	2.17	5564	5.66	0.00
16	40	8560	3600	8542	6.44	0.22	8523	6.3	0.00
Average		3607.63	2391.362	3650.06	2.08	**1.61**	3584.81	2.01	**0.31**

Table 4: Numerical results of Problem 2 for small and medium problem instances.

Instance	n	CPLEX		ACO			IWD		
		OBJ	CPU	OBJ	CPU	RPD	OBJ	CPU	RPD
1	10	109*	0.356	109	0.06	0.00	109	0.05	0.00
2	12	211*	2.389	211	0.1	0.00	211	0.1	0.00
3	14	247*	6.069	247	0.18	0.00	247	0.16	0.00
4	16	213*	25.276	213	0.25	0.00	213	0.25	0.00
5	18	437	3600	446	0.31	2.06	437	0.32	0.00
6	20	262	3600	262	0.51	0.00	262	0.44	0.00
7	22	571	3600	567	0.67	0.00	567	0.67	0.00
8	24	719	3600	719	0.85	0.00	725	0.83	0.83
9	26	675	3600	673	1.17	0.30	671	1.15	0.00
10	28	1044	3600	1028	1.52	0.59	1022	1.47	0.00
11	30	732	3600	720	2.09	0.00	720	1.81	0.00
12	32	919	3600	910	2.55	0.00	910	2.34	0.00
13	34	1280	3600	1414	2.57	10.47	1309	2.79	2.27
14	36	1430	3600	1336	3.8	0.00	1336	3.33	0.00
15	38	1429*	2288.91	1450	4.69	2.04	1421	4.63	0.00
16	40	1732	3600	1727	5.55	1.59	1700	4.95	0.00
Average		750.63	2620.19	752.00	1.68	**1.07**	741.25	1.58	**0.19**

The average RPD of IWD is less than 0.4%, which indicates that the solution of IWD is not more than 0.4% of the optimal solution. The RPD of ACO is also less than 2%. It means that our proposed ACO and IWD algorithm has an excellent performance in solving small problem instances.

The experiment's purpose with small-sized instances is to check the proposed algorithms' effectiveness by comparing them with the optimal solution. Results indicate that the performance of the proposed metaheuristics is close to the optimal solution. However, the CPU time of proposed metaheuristics does not increase exponentially, unlike with the CPLEX solver used to obtain an optimal solution. Overall, the IWD metaheuristic performance is better than ACO's performance in terms of both solution quality and the CPU time.

5.2. Numerical Experiment with Large Dataset. The large dataset is used to compare the performance of proposed algorithms with each other. The numerical results of four problems for large datasets are reported in Tables 8–11.

The results reported in Tables 8–11 exhibit that the performance of IWD is the best followed by the ACO and heuristic algorithm. The IWD produced the best results for almost all instances (55 out of 55 instances). On average, the proposed IWD algorithm's solutions are 4.70%, 4.85%, 4.05%, and 3.75% better than the solutions of the ACO algorithm for four problems, respectively. At the same time, the CPU time of the IWD algorithm is less than the CPU of ACO. The results indicate the better performance of IWD over ACO. The proposed heuristic algorithm can solve all problems within 0.2 seconds, but the heuristic algorithm's performance is worst, with on average RPD of 135.95%.

Table 5: Numerical results of Problem 3 for small and medium problem instances.

Instance	n	CPLEX		ACO			IWD		
		OBJ	CPU	OBJ	CPU	RPD	OBJ	CPU	RPD
1	10	180*	0.138	180	0.07	0.00	180	0.06	0.00
2	12	867*	1.198	867	0.12	0.00	867	0.11	0.00
3	14	1046*	7.719	1046	0.19	0.00	1046	0.18	0.00
4	16	694*	29.361	694	0.28	0.00	694	0.26	0.00
5	18	1760	3600	1760	0.35	0.00	1760	0.38	0.00
6	20	1615	3600	1615	0.53	0.00	1615	0.46	0.00
7	22	1385	3600	1385	0.72	0.00	1385	0.72	0.00
8	24	2819	3600	2819	0.95	0.00	2819	0.9	0.00
9	26	2161	3600	2161	1.23	0.00	2161	1.24	0.00
10	28	3317	3600	3434	1.51	3.53	3317	1.56	0.00
11	30	2310	3600	2310	2.27	0.00	2310	1.93	0.00
12	32	2833	3600	2821	2.31	0.00	2821	2.31	0.00
13	34	4015	3600	4063	2.99	1.63	3998	3	0.00
14	36	3412	3600	3397	3.45	0.00	3397	3.34	0.00
15	38	3794	3600	3687	4.5	1.18	3644	4.45	0.00
16	40	6095	3600	6071	5.17	0.00	6077	4.91	0.10
Average		2393.94	2702.40	2394.38	1.67	**0.40**	2380.69	1.61	**0.01**

Table 6: Numerical results of Problem 4 for small and medium problem instances.

Instance	n	CPLEX		ACO			IWD		
		OBJ	CPU	OBJ	CPU	RPD	OBJ	CPU	RPD
1	10	79*	0.42	79	0.06	0.00	79	0.06	0.00
2	12	193*	1.23	193	0.12	0.00	193	0.11	0.00
3	14	203*	28.98	203	0.19	0.00	203	0.18	0.00
4	16	167*	174.68	167	0.26	0.00	167	0.24	0.00
5	18	285	3600	285	0.34	0.00	285	0.38	0.00
6	20	199	3600	199	0.51	0.00	199	0.45	0.00
7	22	344	3600	344	0.71	0.00	344	0.68	0.00
8	24	585	3600	577	0.92	0.00	577	0.87	0.00
9	26	479	3600	478	1.22	0.00	478	1.22	0.00
10	28	787	3600	806	1.43	2.41	791	1.6	0.51
11	30	568	3600	568	2.21	0.00	568	1.91	0.00
12	32	754	3600	739	2.25	0.00	739	2.31	0.00
13	34	1070	3600	1081	2.97	1.22	1068	2.83	0.00
14	36	833	3600	826	3.41	0.00	826	3.32	0.00
15	38	1106	3600	1069	4.27	1.42	1054	4.43	0.00
16	40	1263	3600	1242	4.89	0.00	1256	4.72	1.13
Average		557.19	2712.83	553.50	1.61	**0.32**	551.69	1.58	**0.10**

Table 7: The average computation time and average relative percentage deviation of four problems.

	ACO		IWD	
	Avg. CPU	Avg. RPD	Avg. CPU	Avg. RPD
Problem 1	2.08	1.61	2.01	0.31
Problem 2	1.68	1.07	1.58	0.19
Problem 3	1.67	0.40	1.61	0.01
Problem 4	1.61	0.32	1.58	0.10

If there is a need for an accurate solution and the CPU time is not the priority consideration, the IWD algorithm can be adopted as a solution method. IWD can be imbedded with entrepreneur resource planning (ERP) software for creating a master production schedule. The results indicate that the IWD algorithm has the edge over ACO algorithms for embedding with any ERP software since it provides a better quality solution in shorter CPU time. One of the drawbacks of the metaheuristic algorithm is the excessive CPU time compared to the heuristic algorithm. Therefore, if the solution is quickly required, the heuristic algorithm can be adopted as a solution.

Table 8: Numerical results of Problem 1 for large problem instances.

Ins.	n	Heuristic			ACO			IWD		
		OBJ	CPU	RPD	OBJ	CPU	RPD	OBJ	CPU	RPD
17	60	35758	<0.2	146.81	14586	28	0.68	14488	25	0
18	80	73276	<0.2	148.33	30705	74	4.06	29507	67	0
19	100	130473	<0.2	119.85	61528	164	3.67	59347	140	0
20	120	231016	<0.2	162.94	87935	271	0.09	87860	245	0
21	140	338574	<0.2	192.79	118691	472	2.64	115636	438	0
22	160	337444	<0.2	154.99	140503	720	6.17	132336	680	0
23	180	502816	<0.2	165.51	193231	1056	2.04	189376	996	0
24	200	583422	<0.2	182.36	216617	1407	4.84	206624	1287	0
25	220	659063	<0.2	120.33	339253	1800	13.42	299125	1814	0
26	240	830265	<0.2	181.52	317844	2513	7.77	294922	2302	0
27	260	1049144	<0.2	168.96	424774	3095	8.89	390081	3095	0
28	280	1182812	<0.2	197.50	408244	4001	2.68	397580	3910	0
29	300	1295486	<0.2	127.93	591626	4861	4.09	568370	4852	0
Average		557657.6	<0.2	159.22	226579.8	1574	**4.70**	214250.2	1527	**0**

Table 9: Numerical results of Problem 2 for l problem instances.

Ins.	n	Heuristic			ACO			IWD		
		OBJ	CPU	RPD	OBJ	CPU	RPD	OBJ	CPU	RPD
17	60	8113	<0.2	55.31	3626	26	0.00	3672	22	1.27
18	80	17579	<0.2	64.28	6452	70	2.67	6280	62	0
19	100	23074	<0.2	52.22	11294	143	2.39	11024	128	0
20	120	38695	<0.2	61.11	15076	256	0.18	15049	222	0
21	140	54679	<0.2	60.87	21966	449	2.59	21397	391	0
22	160	60634	<0.2	58.53	27235	694	7.68	25143	655	0
23	180	86965	<0.2	60.12	36126	1019	4.00	34680	922	0
24	200	110060	<0.2	65.10	39943	1351	3.83	38414	1271	0
25	220	139835	<0.2	56.71	69946	1719	13.45	60539	1770	0
26	240	156011	<0.2	65.82	56678	2450	5.91	53327	2244	0
27	260	198803	<0.2	62.42	83398	2979	10.43	74703	2869	0
28	280	211252	<0.2	64.55	78850	3922	5.01	74898	3762	0
29	300	238829	<0.2	58.98	104442	4763	6.21	97956	4645	0
Average		103425.3	<0.2	60.46	42694.77	1526.2	**4.95**	39775.54	1459	**0.10**

Table 10: Numerical results of Problem 3 for large problem instances.

Ins.	n	Heuristic			ACO			IWD		
		OBJ	CPU	RPD	OBJ	CPU	RPD	OBJ	CPU	RPD
17	60	25181	<0.2	167.88	9448	20	0.51	9400	18	0
18	80	48490	<0.2	166.49	19054	52	4.72	18196	46	0
19	100	94389	<0.2	155.98	38407	129	4.16	36873	107	0
20	120	117355	<0.2	132.00	50584	187	0.00	50584	151	0
21	140	173695	<0.2	125.22	78170	374	1.36	77123	298	0
22	160	215415	<0.2	154.97	86387	621	2.25	84486	483	0
23	180	320249	<0.2	157.16	125942	888	1.13	124534	805	0
24	200	317046	<0.2	143.63	130808	1201	0.52	130135	991	0
25	220	434938	<0.2	183.04	180184	1673	17.26	153668	1561	0
26	240	468509	<0.2	175.29	170604	2267	0.24	170189	1889	0
27	260	592630	<0.2	190.96	229646	2532	12.75	203681	2607	0
28	280	702472	<0.2	215.68	224262	3631	0.78	222527	3044	0
29	300	882460	<0.2	195.33	319851	4513	7.04	298806	4156	0
Average		337909.9	<0.2	166.43	127949.8	1391.4	**4.05**	121554	1242.8	**0**

Table 11: Numerical results of Problem 4 for large problem instances.

Ins.	n	Heuristic			ACO			IWD		
		OBJ	CPU	RPD	OBJ	CPU	RPD	OBJ	CPU	RPD
17	60	5975	<0.2	122.20	2708	34	0.71	2689	33	0
18	80	10659	<0.2	141.92	4638	86	5.27	4406	80	0
19	100	17609	<0.2	125.79	8106	215	3.94	7799	174	0
20	120	20813	<0.2	116.55	9611	323	0.00	9611	253	0
21	140	31751	<0.2	102.17	16117	639	2.62	15705	543	0
22	160	39526	<0.2	137.09	17296	649	3.75	16671	914	0
23	180	57559	<0.2	138.70	24451	899	1.40	24114	810	0
24	200	59202	<0.2	122.63	26763	1769	0.64	26592	1572	0
25	220	91178	<0.2	135.63	44615	1919	15.30	38696	1670	0
26	240	88735	<0.2	153.62	35148	3272	0.46	34988	2273	0
27	260	116397	<0.2	148.96	52314	3837	11.89	46753	4131	0
28	280	134196	<0.2	173.21	49550	5584	0.88	49118	4534	0
29	300	157294	<0.2	148.91	64611	5829	2.25	63192	5122	0
Average		63914.9	<0.2	135.95	27379.1	1927	**3.78**	26179.54	1701	**0**

6. Conclusions

This paper evaluates a set of single-machine with two-agent scheduling problem to minimize the total weighted completion time and total completion time of the first agent while keeping the number of tardy jobs of the second agent under a prespecified limit. The service and manufacturing industries find many applications in the problems. In this paper, we proposed a general mathematical model for all four problems and an ACO-based metaheuristic to solve the problems. The proposed mathematical model is solved by using AMPL software with the CPLEX solver.

It can be deduced from the results that the proposed metaheuristic has obtained near-optimal solutions, as the reported variation between the proposed metaheuristic and mathematical model's outcomes are extremely low among four problems. More importantly, the proposed metaheuristic can reach a better solution than the CPLEX solver, and it consumes much less computation time than the CPLEX solver. Subsequent future research directions include developing other metaheuristics to provide even better results for larger sized problem instances in a shorter time.

In the future, the algorithm used in this paper can be extended to solve different scheduling problems such as flow shop scheduling problems, job shop scheduling problems, parallel scheduling problem, and disassembly sequencing problems. The results exhibit the excellent performance of the IWD algorithm over the ACO algorithm. The results indicate the call for the application of the IWD algorithm on solving other scheduling problems.

Data Availability

Data used in this research work are available from Yuvraj Gajpal (yuvraj.gajpal@umanitoba.ca).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research is partially supported by NSERC, grant 318689.

References

[1] Y. Feng, M. Zhou, G. Tian et al., "Target disassembly sequencing and scheme evaluation for CNC machine tools using improved multi-objective ant colony algorithm and fuzzy integral," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 12, pp. 2438–2451, 2018.

[2] Y. Feng, Y. Gao, G. Tian, Z. Li, H. Hu, and H. Zheng, "Flexible process planning and end-of-life decision-making for product recovery optimization based on hybrid disassembly," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 1, pp. 311–326, 2018.

[3] Q. Zhu, Y. Qiao, and N. Wu, "Optimal integrated schedule of entire process of dual-blade multi-cluster tools from start-up to close-down," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 2, pp. 553–565, 2019.

[4] K. R. Baker and J. Cole Smith, "A multiple-criterion model for machine scheduling," *Journal of Scheduling*, vol. 6, no. 1, pp. 7–16, 2003.

[5] B. Yagmahan and M. M. Yenisey, "A multi-objective ant colony system algorithm for flow shop scheduling problem," *Expert Systems with Applications*, vol. 37, no. 2, pp. 1361–1368, 2010.

[6] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 904–916, 2019.

[7] A. K. Bhardwaj, Y. Gajpal, C. Surti, and S. S. Gill, "HEART: unrelated parallel machines problem with precedence constraints for task scheduling in cloud computing using heuristic and meta-heuristic algorithms," *Software: Practice and Experience*, vol. 50, no. 12, pp. 2231–2251, 2020.

[8] M. J. Soomer and G. J. Franx, "Scheduling aircraft landings using airlines' preferences," *European Journal of Operational Research*, vol. 190, no. 1, pp. 277–291, 2008.

[9] P. J. Brewer and C. R. Plott, "A binary conflict ascending price (BICAP) mechanism for the decentralized allocation of the right to use railroad tracks," *International Journal of Industrial Organization*, vol. 14, no. 6, pp. 857–886, 1996.

[10] J. M. Peha, "Heterogeneous-criteria scheduling: minimizing weighted number of tardy jobs and weighted completion time," *Computers & Operations Research*, vol. 22, no. 10, pp. 1089–1100, 1995.

[11] A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici, "Scheduling problems with two competing agents," *Operations Research*, vol. 52, no. 2, pp. 229–242, 2004.

[12] V. Suresh and D. Chaudhuri, "Bicriteria scheduling problem for unrelated parallel machines," *Computers & Industrial Engineering*, vol. 30, no. 1, pp. 77–82, 1996.

[13] T. C. E. Cheng, C. T. Ng, and J. J. Yuan, "Multi-agent scheduling on a single machine with max-form criteria," *European Journal of Operational Research*, vol. 188, no. 2, pp. 603–609, 2008.

[14] T. C. E. Cheng, C.-Y. Liu, W.-C. Lee, and M. Ji, "Two-agent single-machine scheduling to minimize the weighted sum of the agents' objective functions," *Computers & Industrial Engineering*, vol. 78, pp. 66–73, 2014.

[15] C. T. Ng, T. C. E. Cheng, and J. J. Yuan, "A note on the complexity of the problem of two-agent scheduling on a single machine," *Journal of Combinatorial Optimization*, vol. 12, no. 4, pp. 387–394, 2006.

[16] J. Y.-T. Leung, M. Pinedo, and G. Wan, "Competitive two-agent scheduling and its applications," *Operations Research*, vol. 58, no. 2, pp. 458–469, 2010.

[17] W.-C. Lee, W.-J. Wang, Y.-R. Shiau, and C.-C. Wu, "A single-machine scheduling problem with two-agent and deteriorating jobs," *Applied Mathematical Modelling*, vol. 34, no. 10, pp. 3098–3107, 2010.

[18] Y. Yin, S. R. Cheng, T. C. E. Cheng, W. H. Wu, and C. C. Wu, "Two-agent single-machine scheduling with release times and deadlines," *International Journal of Shipping and Transport Logistics*, vol. 5, no. 1, pp. 75–94, 2013.

[19] J. Yuan, "Multi-agent scheduling on a single machine with a fixed number of competing agents to minimize the weighted sum of number of tardy jobs and makespans," *Journal of Combinatorial Optimization*, vol. 34, no. 2, pp. 433–440, 2017.

[20] Y. Yin, D.-J. Wang, C.-C. Wu, and T. C. E. Cheng, "CON/SLKdue date assignment and scheduling on a single machine with two agents," *Naval Research Logistics (NRL)*, vol. 63, no. 5, pp. 416–429, 2016.

[21] L. Wan, J. Yuan, and L. Wei, "Pareto optimization scheduling with two competing agents to minimize the number of tardy jobs and the maximum cost," *Applied Mathematics and Computation*, vol. 273, pp. 912–923, 2016.

[22] B. Mor and G. Mosheiov, "A two-agent single machine scheduling problem with due-window assignment and a common flow-allowance," *Journal of Combinatorial Optimization*, vol. 33, no. 4, pp. 1454–1468, 2017.

[23] B. Mor and G. Mosheiov, "Polynomial time solutions for scheduling problems on a proportionate flowshop with two competing agents," *Journal of the Operational Research Society*, vol. 65, no. 1, pp. 151–157, 2014.

[24] D. Lei, "Variable neighborhood search for two-agent flow shop scheduling problem," *Computers & Industrial Engineering*, vol. 80, pp. 125–131, 2015.

[25] F. Yu, P. Wen, and S. Yi, "A multi-agent scheduling problem for two identical parallel machines to minimize total tardiness time and makespan," *Advances in Mechanical Engineering*, vol. 10, no. 2, p. 1687814018756103, 2018.

[26] H. Li, Y. Gajpal, and C. R. Bector, "A survey of due-date related single-machine with two-agent scheduling problem," *Journal of Industrial & Management Optimization*, vol. 13, no. 5, p. 1, 2019.

[27] Y. Yin, D. Wang, and T. C. E. Cheng, *Due Date-Related Scheduling with Two Agents*, Springer, Singapore, 2020.

[28] Y. Yin, Y. Chen, K. Qin, and D. Wang, "Two-agent scheduling on unrelated parallel machines with total completion time and weighted number of tardy jobs criteria," *Journal of Scheduling*, vol. 22, no. 3, pp. 315–333, 2019.

[29] R. Chen, J. Yuan, and Y. Gao, "The complexity of CO-agent scheduling to minimize the total completion time and total number of tardy jobs," *Journal of Scheduling*, vol. 22, no. 5, pp. 581–593, 2019.

[30] Y. H. Zhang, Y. J. Gong, W. N. Chen, T. L. Gu, H. Q. Yuan, and J. Zhang, "A dual-colony ant algorithm for the receiving and shipping door assignments in cross-docks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 7, pp. 2523–2539, 2018.

[31] H. Jia, H. Miao, G. Tian et al., "Multi-objective bike repositioning in bike-sharing systems via a modified artificial bee colony algorithm," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 2, pp. 909–920, 2019.

[32] H. Li, Y. Gajpal, and C. R. Bector, "Single machine scheduling with two-agent for total weighted completion time objectives," *Applied Soft Computing*, vol. 70, pp. 147–156, 2018.

[33] H. S. Hosseini, "Problem solving by intelligent water drops," in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, pp. 3226–3231, IEEE, Singapore, September 2007.

[34] B. O. Alijla, L.-P. Wong, C. P. Lim, A. T. Khader, and M. A. Al-Betar, "A modified intelligent water drops algorithm and its application to optimization problems," *Expert Systems with Applications*, vol. 41, no. 15, pp. 6555–6569, 2014.

[35] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proceedings of the 1999 Congress on Evolutionary Computation CEC 99*, San Diego, CA, USA, July 1999.

[36] D. Maruthanayagam and D. R. U. Rani, "Enhanced ant colony system based on the RASA algorithm in grid scheduling," *IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 2, no. 4, pp. 1659–1674, 2011.