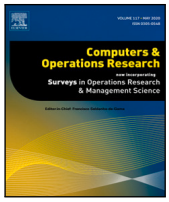
Computers & Operations Research 157 (2023) 106266

Contents lists available at ScienceDirect

Computers and Operations Research

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



Manufacturing rescheduling after crisis or disaster-caused supply chain disruption

Hongguang Bo [a,1], Xiao Alison Chen [b,*,1], Qian Luo [c,1], Wenpeng Wang [a,1]

[a] School of Economics and Management, Dalian University of Technology, 2 Linggong Rd, Dalian, China
[b] Peter T Paul College of Business and Economics, University of New Hampshire, 10 Garrison Ave, Durham, NH, USA
[c] International Business School Suzhou, Xi'an Jiaotong-Liverpool University, 111 RenAi Rd, Suzhou, China

ARTICLE INFO

Keywords:
Production rescheduling
Supply chain disruption
Two-stage genetic algorithm
Estimation of distribution algorithm
Productivity recovery

ABSTRACT

In this paper, we study the problems a repair shop has with rescheduling after major supply disruptions. The repair shop provides repair and maintenance services to its customers. After a major disruption to production, the repair shop faces delays in production and order delivery due to shortages in materials and/or labor, which requires rescheduling of all the unfinished parts. We observe that the finished parts incur high holding costs until the entire order is completed, while any unfinished parts (in the form of raw material or work-in-progress) incur low holding costs until production starts. Moreover, the repair shop incurs a setup cost when switching between different types of parts. Considering these new features, we formulate the rescheduling problem for the repair shop under a coordinated supply chain as an integer program to minimize the total tardiness, setup cost, and holding cost. To solve the model, we propose an innovative two-stage genetic algorithm, which utilizes the estimation of distribution algorithm (EDA) to improve the search process of the optimal solution. We test the performance of this algorithm on a dataset generated from the order data of a heavy machinery maintenance provider. The numerical results show that our model generates solutions that outperform the initial schedule, which was obtained by minimizing holding and setup costs without disruption. In addition, using other closely-related genetic algorithms as benchmarks, we show that our algorithm outperforms the benchmarks without sacrificing the computational time. We also discuss an extension of the main model by considering the recovery of productivity in terms of processing time.

1. Introduction

Production scheduling has been heavily studied in supply chain and operations management because of its importance and complexity. It is a crucial part of many industries, including manufacturing, healthcare, and distribution (see, for example, Graves 1981, Portougal and Robb 2000, Guido et al. 2017, Sun et al. 2020, Helo et al. 2019). To achieve optimal operations performance, researchers have worked on different types of job scheduling or worker scheduling models. As technology and the industries evolve, scheduling problems incorporate more constraints and flexibility to model specific scenarios. In recent years, rescheduling models have become an emerging field of research. Rescheduling models study how to schedule jobs again when the initial production plan cannot be further continued as certain conditions or constraints no longer hold. Rescheduling is often necessary when there is a disruption in the supply and production system to reduce or avoid losses caused by the interruption; such interventions can save millions of dollars (Sheffi 2005, Kim et al. 2010).

An example of such a disruption is the recent COVID-19 pandemic. In the case of the COVID-19 pandemic, city lockdowns caused countless issues and sustained impacts on transportation, food processing, manufacturing, and the healthcare industry. The interruptions resulted in shortages in raw materials, consumer goods, and labor. Millions of people lost jobs, housing, and healthcare due to the COVID-19 pandemic, which has become a social and economic crisis for many countries with far-reaching and long-lasting effects.

During this crisis, many manufacturing plants have faced raw material shortages, labor shortages, shipping delays, and plant shutdowns, all of which call for production rescheduling (Marinova and Bitri 2021, Magableh 2021, Sheffi 2021). A particular problem we have observed in practice is the delay of production and order delivery due to the mandatory shutdown of a job shop that performs maintenance and repair services in Qingdao. This repair shop orders parts from a designated manufacturer in the same city. During regular operation, the repair shop uses a scheduling tool to generate an optimal production

* Corresponding author.
E-mail addresses: hgbo@dlut.edu.cn (H. Bo), Alison.Chen@unh.edu (X.A. Chen), Qian.Luo@xjtlu.edu.cn (Q. Luo), wwpeng@mail.dlut.edu.cn (W. Wang).
[1] All authors contributed equally to the manuscript.

https://doi.org/10.1016/j.cor.2023.106266
Received 29 July 2022; Received in revised form 19 April 2023; Accepted 25 April 2023
Available online 30 April 2023

schedule based on the due dates, inventory holding cost, and setup costs. When everything goes according to plan, all orders can be produced and shipped by the due date. However, a mandatory lockdown in Qingdao disrupted the production schedule for weeks. Some parts had already been produced by the manufacturer while others had not. The repair shop had to wait for all parts to be produced before delivering the entire order. After the production resumes, many orders will be delayed if the repair shop continues to produce according to the initial production schedule. Orders past the due date will result in costly penalties. To reduce these costs, the repair shop must reschedule its jobs to minimize the total holding cost, setup cost, and penalty cost. Motivated by this real-life problem in the repair shop, we investigate and model a rescheduling problem under disruption.

In this paper, we consider a single-machine rescheduling problem for a repair shop and its manufacturing facility after a prolonged disruption to production. These two parties coordinate production schedules and make joint decisions. Throughout the paper, the two together are referred to as the *service provider*. The manufacturing facility produces parts for the repair shop, which provides maintenance and repair services to customers. The service provider minimizes tardiness and inventory costs. At the beginning of the disruption, some parts have already been produced. The finished parts incur high holding costs until the entire order is completed, while any unfinished parts (in the form of raw material) incur low holding costs until production starts. The service provider incurs a late penalty cost for orders past the due date. There is a setup cost during production if parts produced consecutively are not the same item type. The service provider's goal is to minimize the total cost, including holding costs, setup costs, and late penalty costs. We design a two-stage genetic algorithm that utilizes the estimation of distribution algorithm. Through numerical tests, we show that this algorithm outperforms a few other genetic algorithms that are designed for scheduling needs.

Note that the problem studied in this paper is more complex than a conventional single-machine scheduling problem that only considers tardiness. First, this paper considers two types of holding costs: the low holding cost of raw materials before production and the high holding cost of finished goods after production. By imposing these holding costs, the model will try to complete an order as fast as possible once the production of the order has started, even if there is plenty of time left until the due date. Second, when rescheduling after the disruption, the model needs to take into consideration those partially completed orders. Parts completed before the disruption incur high holding costs as long as the entire order is incomplete. Thus, at the time of rescheduling, the model must consider the parts produced prior to disruption. To incorporate this feature, we calculate the holding costs starting from the beginning of production after disruption to the time when all orders are completed. Lastly, our model considers the setup costs between different types of production job. When time is abundant, the model will group the same kind of parts together in the production. When time is limited, our model will balance the setup cost against the late penalty cost.

As discussed above, supply chain disruption caused by a social crisis, such as a pandemic, may affect the global supply chain. Another example of a major social crisis is the Russia and Ukraine war, which began in 2022. European countries and the United States imposed sanctions on Russia and suspended all business with the country. As a result, natural gas and crude oil supply have dropped significantly. Other commodities, such as wheat, barley, titanium, nickel, and palladium, have all experienced supply issues as a result of the war (Kilpatrick, 2022). The conflict has also disrupted day-to-day industrial activities in both countries. International logistics that used to travel through these two countries now take detours, resulting in shipment delays. All of these factors lead to disruptions in the global supply chain.

Our model is not exclusively for rescheduling in response to supply chain disruptions caused by social crises. Natural disaster is also a common cause of supply chain disruptions. In August 2005, Hurricane Katrina struck the Gulf Coast, breaching levees and causing significant damage and thousands of deaths. Power outages and road closures along the Gulf Coast led to severe supply chain disruptions. Businesses had to reroute their supply chains to other areas and reschedule production in alternative plants. It took them months to recover from this disaster. Faced with such rescheduling problems, it is necessary to consider the holding costs of finished goods and unfinished (raw material or work-in-progress) products.

We note that supply chain disruptions caused by social crises and natural disasters may result in severe damage to manufacturing plants, which could delay the production and delivery of orders for weeks, sometimes even months. Nonetheless, in addition to social crises and natural disasters, our model can be applied to any scenario that involves long-term disruption, such as resource shortages. Resource shortages can cause supply and production delays and usually arise from disruptions to transportation. One example of an interruption to raw material transportation is the Suez Canal obstruction of March 2021. The container ship Ever Given was grounded in the world's busiest canal for six days, blocking over 369 ships that were waiting to pass through the canal. It is estimated that 9.6 billion worth of trade was stuck during these six days, causing a massive supply shortage worldwide. Another example of resource shortage is the Colonial Pipeline cyberattack of May 2021. The Colonial Pipeline suffered a ransomware cyberattack that affected its system. The company halted all of its operations to contain the attack, affecting fuel supply to the southeastern United States for many days. This was the largest cyberattack on oil infrastructure in the history of the United States. Similar supply chain disruption incidences can be seen daily, causing delays in the production schedule.

The above examples demonstrate the severe impact of supply chain disruptions and the need for rescheduling. Through proper scheduling and coordination, supply chain costs can be greatly reduced (Hall and Potts, 2003). As we discover from the literature in Section 2, there are various models and algorithms available for rescheduling. However, these existing models are not directly applicable to our context due to the idiosyncratic features we described above, and the algorithms currently available have relatively poor performance, underlining the significance and contribution of our work.

This paper is organized as follows. Section 2 reviews the related literature. In Section 3, we present the rescheduling model with a model extension. In Section 4, we introduce a two-stage genetic algorithm with the estimation of distribution algorithm (EDA) incorporated (we refer to it as EAGA throughout this paper). Section 5 contains the results of numerical tests showing the performance of our algorithm when applied to data from a heavy machinery maintenance service provider. We compare the total cost of the new schedule created using our model with those of the initial schedule. We also compare the performance of the proposed algorithm with five related genetic algorithms. Finally, Section 6 concludes the analysis and results. Future research directions are also suggested in this section.

2. Literature review

In this section, we review the related literature. Our work contributes to three areas: single-machine scheduling problems, rescheduling under disruption, and algorithms for scheduling problems.

First, our work is related to single-machine scheduling problems. Graves (1979) discusses two single-machine multi-product scheduling problems with deterministic demand and shows that these problems are identical to a one-warehouse, multiple-retailer inventory problem. Earlier work by Santos and Magazine (1985), Dobson et al. (1987), Naddef and Santos (1987), and Coffman et al. (1990) study single-machine batch scheduling problems to minimize inventory holding cost. Dobson et al. (1987) and Naddef and Santos (1987) also propose heuristic solutions for their models. In addition to minimizing inventory holding costs, some work includes other costs in the objective function.

Table 1
Comparison between our paper and the reviewed rescheduling papers.

Features	Cost Features				Other Features				
	Holding	Setup	Schedule	Delivery	Tardiness	Makespan	Task priority, urgency & timeliness	Instability	Machine Workload
Hall and Potts (2004)			√		√				
Selvarajah and Steiner (2009)	√			√					
Steiner and Zhang (2011)					√				
Hoogeveen et al. (2012)					√	√			
Liu and Ro (2014)					√	√			
Liu et al. (2016)					√	√			
Yin et al. (2016)					√	√			
Gao et al. (2018)						√		√	√
Yu et al. (2018)					√				
Samarghandi (2019)						√			
Zhang et al. (2019)							√		
Zhang et al. (2021)					√	√			√
Our paper	√	√			√				

For example, Hall and Potts (2003) study the supply chain scheduling problem of minimizing inventory and delivery costs. Selvarajah and Steiner (2009) study a similar problem, whereby a delivery cost is associated with each batch. They propose approximation algorithms and prove that these algorithms have worst-case bounds.

The second, related area is rescheduling problems under disruption (we summarize and compare the main modeling features of our work and the rescheduling papers in Table 1). The study by Hall and Potts (2004) is one of the earliest papers on rescheduling problems. This work considers a rescheduling problem with a new set of job arrivals that creates a disruption. Liu and Ro (2014) study a rescheduling problem that minimizes a certain cost objective. The manufacturer needs to reschedule the jobs without excessively disrupting the original schedule.

Steiner and Zhang (2011) study a model that reschedules orders with a simultaneous assignment of revised due dates. Their objective is to minimize due date escalation and tardiness penalties for the supplier. Hoogeveen et al. (2012) discuss a model where new jobs arrive during production and the manufacturer needs to reschedule the original and new jobs. The objective is to minimize the makespan and total time deviation. Yin et al. (2016) study the rescheduling problem on parallel machines with multiple machine disruptions. Their objective is to minimize the total completion time and the total time deviation. Some recent papers consider real-time rescheduling, whereby new orders and disruptions constantly arrive, and the schedule needs to be adjusted accordingly. Gao et al. (2018) discuss the rescheduling problem that arises when a high-priority new job is inserted into the production plan. This model considers two objectives: instability and a time index. A heuristics named DJaya is introduced, and numerical tests are done to compare DJaya with other algorithms. Zhang et al. (2021) address the rescheduling problem of a dynamic, flexible job shop with disruptions incorporated. This model considers three objectives (makespan, maximum machine workload, and total tardiness) and a hybrid genetic algorithm is designed for this model. Our work contributes to this stream of literature by capturing the setup costs when switching between job types and the holding cost for partially finished orders.

Another area to which we contribute is algorithms for solving scheduling problems. The current literature contains three main streams of scheduling algorithms: exact algorithms, approximation algorithms, and heuristics (we summarize and compare the scheduling algorithms of our work with those of the reviewed papers in Table 2). Because the scheduling problems are integer-programming problems, most models are NP-hard. Three of the most classic exact algorithms are dynamic programming, branch-and-bound, and cutting-planes. Researchers design new algorithms based on these classic exact algorithms for specific scheduling problems. Bodur and Luedtke (2017) use a mixed-integer rounding technique in the branch-and-cut algorithm to solve stochastic integer programming scheduling models. Drozdowski et al. (2017) propose branch-and-bound and local search algorithms to solve scheduling problem for a single machine with maintenance restrictions. Kowalczyk and Leus (2017) consider a parallel machine scheduling problem and introduce an exact algorithm based on branch and price that combines multiple algorithms. Emadikhiav et al. (2020) propose a branch-and-check and branch-and-price algorithm to solve a combined routing and scheduling problem. Gmys et al. (2020) introduce an efficient branch-and-bound algorithm that combines dynamic branching and lower bound refinement strategies.

To speed up computation time, people have proposed approximation algorithms to solve for near-optimal solutions. Liu and Ro (2014) study a pseudopolynomial time optimal algorithm, an approximation algorithm, and a fully polynomial time approximation scheme. Levi et al. (2019) develop a class of scheduling models with exploration–exploitation tradeoff and introduce approximation algorithms. Baardman et al. (2019) explore the scheduling of promotion vehicles and develop approximation algorithms that achieve a near-optimal result.

In recent years, heuristics have been widely used for scheduling problems. Based on the original genetic algorithm (GA) in Holland (1975), researchers have developed various modifications to GA to improve its solution quality and solving speed. Liu et al. (2016) propose adaptive genetic algorithms (AGA) to solve scheduling problems with due date changes and machine breakdowns. Yu et al. (2018) design a genetic algorithm with an updated decoding process (GAD) to solve a hybrid flow shop scheduling problem. Zhang et al. (2019) design a hybrid adaptive genetic algorithm (HAGA) to solve a task-scheduling problem with limited time and energy resources. Samarghandi (2019) develops a genetic algorithm (GANW) to handle large-scale job scheduling problems with no waiting time. Soares and Carvalho (2020) propose a parallel biased random-key genetic algorithm for a scheduling problem with identical parallel machines. Our paper differs from these papers by proposing an EDA-incorporated two-stage GA (denoted by EAGA), which aims to further improve the efficiency and efficacy in searching for the optimal solution.

Among all related research discussed, Selvarajah and Steiner (2009) is the most relevant to our work on model settings. This work is motivated by the need to reschedule due to production disruptions caused by resource shortages, natural disasters, or social crises. In this paper, we study the rescheduling problem that occurs when a manufacturer produces and supplies replacement parts to a service provider. The service provider processes repair jobs and delivers the finished jobs to the customers. After a major disruption to production, the service provider needs to reschedule its jobs to minimize tardiness and inventory costs. We further extend the model to incorporate cases with some processing time modifications. A two-stage genetic algorithm is then proposed to solve the rescheduling problem. Numerical experiments show that the proposed rescheduling model can effectively reduce tardiness and total

Table 2
Comparison of the algorithms used.

Exact algorithm	Approximation algorithms	Heuristic algorithms
Bodur and Luedtke (2017)	Liu and Ro (2014)	Holland (1975)
Drozdowski et al. (2017)	Levi et al. (2019)	Liu et al. (2016)
Kowalczyk and Leus (2017)	Baardman et al. (2019)	Yu et al. (2018)
Emadikhiav et al. (2020)	Selvarajah and Steiner (2009)	Zhang et al. (2019)
Gmys et al. (2020)		Samarghandi (2019)
		Soares and Carvalho (2020)
		Our paper (EAGA)

inventory costs. In addition, the proposed algorithm performs better than the five closely-related genetic algorithms in the literature.

3. Models

3.1. Notation and sequence of events

This model describes a scheduling problem under supply chain disruption. Consider a service provider (consisting of a repair shop and a manufacturing facility) who processes maintenance and repair jobs for its customers. At the beginning of the planning horizon, the service provider receives maintenance and repair orders from its customers. Each order consists of a few replacement parts. All parts are produced by the manufacturing facility and assembled by the repair shop. Since the parts are make-to-order, they cannot be produced before orders arrive. An order is considered complete only when all parts have been produced. In this centralized setting, the service provider needs to generate a production schedule to minimize the total cost (TC), which includes the holding costs and the setup cost. In our model, there are two types of holding cost: a low holding cost for storing raw material or unfinished parts, and a high holding cost for storing the finished products. This assumption is reasonable and common in the automotive and heavy machinery industry. Raw material, such as steel and gearwheels, is usually stored in covered outdoor spaces, whereas finished parts, such as suspension gears and passenger doors, are stored in monitored warehouses with advanced inventory management and security systems. The physical properties of the raw material and finished parts also affect the holding cost. For example, glassware has high holding cost while sand has low holding cost, and heat-treated ingots have a high holding cost while steel has low holding cost (Li et al., 2019). The setup cost is used to capture the price of extra labor and the operational costs for changeovers between job types. When production is initially scheduled, the service provider makes sure that no order will be late. In our model, the parts in each order do not have precedence relationships and parts in the same order do not need to be processed consecutively.

When supply chain disruption occurs, production is interrupted for a period. After production resumes, some orders will miss the due date, thus incurring a late penalty cost. If the service provider continues to follow the initial schedule, the late penalty cost from delayed orders could add up quickly. Thus, the service provider needs reschedule the unfinished orders to minimize the total cost, which now includes the holding cost, the setup cost, and the late penalty cost. To better describe the process, we list the sequence of events below.

1. Before the start of production and servicing, the service provider receives orders from its customers. Each order requires a few replacement parts.
2. Based on the production setup cost S, inventory holding cost H, and due dates d of each order, the service provider generates a schedule to minimize holding and setup costs. We refer to this schedule as the *initial schedule*. Note here, the initial schedule does not consider disruption and the late penalty cost.
3. Production starts. Parts are produced one by one according to the initial schedule.
4. Disruption occurs and production is paused. Orders that have been completed before the disruption will be shipped before the disruption . Orders that have not yet been completed before the disruption will need further processing after it.
5. After a period of production downtime, operation resumes. Orders that cannot be completed by the due date will incur a late penalty cost. The service provider reschedules the production sequence to minimize the late penalty cost P, setup cost S, and inventory holding cost H. This is the rescheduling problem we focus on in this paper.
6. Once all required parts from an order have been produced, the service provider performs the maintenance service and ships the completed order to the customer.

Without loss of generality, we set the starting time after disruption as 0. A negative time indicates that the parts have been produced before the disruption. See Fig. 1 for the timeline of the events. Next, we define the notation used in our model.

Let M denote the number of orders received by the service provider. Order i consists of N_i parts. Throughout this paper, we use *job* to refer to the task of producing one of the parts in an order. Jobs are indexed by subscripts ij, where i denotes the order number and j denotes the parts in that order. The completion of job ij only indicates the completion of the jth job in order i. The machine cannot process in batches; therefore, an identical job from another order would need to be processed separately. Before disruption occurs, some jobs have already been completed. To simplify the notation, we assume, without loss of generality, that only N jobs[2] need to be rescheduled in each order. In practice, our model allows for any number of jobs to be rescheduled. One can simply replace N with the exact number of jobs that need to be rescheduled in each order.

In the following, we define the parameters and variables in this model.

- t_{ij}: the processing time of job j in order i, $i = 1, \dots, M$, $j = 1, \dots, N$.
- c_{ij}: the completion time of job j in order i after rescheduling. For jobs completed before the disruption, we set $c_{ij} = 0$.
- $\bar{c}_i = \max\{c_{ij}\}$: the completion time of the entire order i after rescheduling.
- d_i: the due date of order i.
- ρ: the per unit per unit time late penalty cost incurred by the service provider if the firm fails to complete an order by the due date.
- h_L: the per unit per unit time holding cost of a job before processing. The holding cost of raw material is usually lower before processing.
- h_H: the per unit per unit time holding cost of a job after processing. The holding cost of finished jobs is usually higher after processing.

[2] Here, we index the jobs that need to be scheduled as jobs 1 to N. Since jobs do not have precedence relationships, this index system only simplifies the notation. It will not change the model itself.

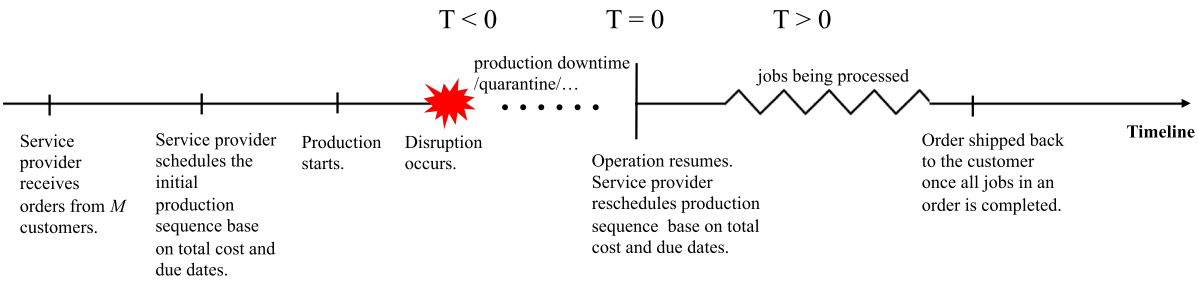


Fig. 1. Timeline of the events.

- F: the setup cost when switching from one type of job to another. There is no setup cost if the manufacturing facility continues to produce the same type of jobs.
- $a_{ij,g}$: an indicator parameter to denote if job ij is of type g.

$$a_{ij,g} = \begin{cases} 1, & \text{if } ij \text{ is of type } g; \\ 0, & \text{otherwise.} \end{cases}$$

- G: the set of all job types.
- $O = \{O_1, \ldots, O_k, \ldots, O_{M\times N}\}$: an ordered set that denotes the production sequence.
- $x_{ij,k}$: a binary variable that denotes the position of job ij in the ordered set O.

$$x_{ij,k} = \begin{cases} 1, & \text{if } ij \text{ is assigned to the } k\text{th position;} \\ 0, & \text{otherwise.} \end{cases}$$

- C_k: a variable that defines the completion time of the job in the kth position. C_0=0.
- y_k: a binary variable that denotes if there is a setup after the kth position.

$$y_k = \begin{cases} 1, & \text{if there is a setup after the } k\text{th position;} \\ 0, & \text{otherwise.} \end{cases}$$

The service provider needs to determine the production sequence of jobs to minimize the total cost. The total cost consists of three parts, the late penalty cost, the holding cost, and the setup cost. The late penalty cost P can be written as:

$$P = \sum_{i=1}^{M} \rho \max\{0, \bar{c}_i - d_i\}.$$

The inventory cost H can be written as:

$$H = \sum_{i=1}^{M}\sum_{j=1}^{N_i} h_H(\bar{c}_i - c_{ij}) + \sum_{i=1}^{M}\sum_{j=1}^{N} h_L(c_{ij} - t_{ij}).$$

The first part is the holding cost for the completed jobs. The finished goods require a better warehousing environment; thus, the per unit holding cost is higher. This high holding cost is calculated for the parts that are completed, both before and after the disruption, as long as the entire order is not completed. The second part is the low holding cost for the incomplete jobs (the per unit holding cost is low when the inventory is in the raw material stage). Note here, we only consider the holding costs incurred after the disruption. All costs incurred before the disruption are fixed; thus, this should not affect the rescheduling problem.

Lastly, the setup cost S can be written as:

$$S = F \sum_{k=1}^{M\times N} y_k.$$

Again, the setup cost S only includes the setup cost that is incurred after the disruption.

3.2. Service provider's rescheduling model

In this section, we formulate the service provider's rescheduling model.

$$\min \quad TC(O) = P + H + S$$

$$\text{s.t.} \quad \sum_{k=1}^{M\times N} x_{ij,k} = 1, \qquad \forall i \in \{1,\ldots,M\}, j \in \{1,\ldots,N\} \quad (1a)$$

$$\sum_{i=1}^{M}\sum_{j=1}^{N} x_{ij,k} = 1, \qquad \forall k \in \{1,\ldots,M\cdot N\} \quad (1b)$$

$$C_k \geq C_{k-1} + (x_{ij,k-1} + x_{pq,k} - 1)t_{ij}, \qquad \forall k \in \{1,\ldots,M\cdot N\} \quad (1c)$$

$$c_{ij} \geq C_k - (1 - x_{ij,k})M, \qquad \forall i \in \{1,\ldots,M\}, j \in \{1,\ldots,N\} \quad (1d)$$

$$y_k = \sum_{i=1}^{M}\sum_{j=1}^{N} x_{ij,k}\cdot a_{ij,g} - \sum_{i=1}^{M}\sum_{j=1}^{N} x_{ij,k+1}\cdot a_{ij,g}, \qquad \forall k \in \{1,\ldots,M\cdot N\}, \forall g \in G \quad (1e)$$

$$\bar{c}_i = \max_j\{c_{ij}\}, \qquad \forall i \in \{1,\ldots,M\} \quad (1f)$$

$$x_{ij,k} \in \{0,1\}, y_k \in \{0,1\}, c_{ij} \geq 0, C_k \geq 0 \qquad \forall i,j,k$$

where

$$P = \sum_{i=1}^{M} \rho \max\{0, \bar{c}_i - d_i\};$$

$$H = \sum_{i=1}^{M}\sum_{j=1}^{N_i} h_H(\bar{c}_i - c_{ij}) + \sum_{i=1}^{M}\sum_{j=1}^{N} h_L(c_{ij} - t_{ij});$$

$$S = F \sum_{k=1}^{M\times N} y_k.$$

The objective is to minimize the total cost (TC) of a sequence of jobs, which is the sum of the penalty cost, holding costs, and setup cost. Constraint (1a) requires that each job must exist exactly once in the sequence. Constraint (1b) requires that each position in the sequence must correspond to exactly one job. Constraint (1c) calculates the completion time for the job at the kth position. Constraint (1d) relates the completion time of job ij with its assigned position. M is the generic notation for a sufficiently large number, which could take on any value that is larger than the sum of processing times in our model. Constraint (1e) enforces $y_k = 1$ if there is a setup operation between the kth and the $(k+1)$th position, and $y_k = 0$ otherwise. Constraint (1f) defines the completion time of order i.

The service provider's model is a mixed integer linear programming problem (MILP) after some simple transformation. The decision variables $x_{ij,k}$ and y_{ij} are constrained to be integers, while variables c_{ij} and C_k are continuous. The objective function and constraints are linear or can be written in linear forms. To linearize P, one can define new variables τ_i as a tardiness variable to replace $\max\{0, \bar{c}_i - d_i\}$ in P and

add a group of linear inequality constraints:

$$\tau_i \geq \bar{c}_i - d_i,$$
$$\tau_i \geq 0.$$

Constraint (1f) can be replaced with a group of linear inequalities, $\bar{c}_i \geq c_{ij}, \forall j$.

Next, we briefly discuss the complexity of our model. We claim that a simplified version of our model is equivalent to a single machine-scheduling problem that minimizes total weighted tardiness, denoted by $1//\bar{T}$, using the three-field classification of scheduling problems. By setting the holding costs and setup cost to 0, the objective function becomes minimizing the total tardiness of the orders. As proved by Du and Leung (2017), minimizing the total tardiness on a single machine is NP-hard. With the additional constraints and cost objectives, our model is at least as hard as the $1//\bar{T}$ problem. Solving this problem on a large scale using an exact algorithm will be time-consuming.

3.3. Model extension

In this part, we discuss an extension of the model to incorporate changes in processing time. In the base model, we assume that the processing time t_{ij} remains the same after a disruption. However, this may not always be the case. After a disruption, it usually takes a while for factories to recover to full production capacity. Thus, the processing time during that transition period could be longer than normal. One example we saw during the COVID-19 pandemic is that factories' productivity dramatically decreases due to insufficient labor resources. Factories are unable to return to their full capacity for productivity until workers are released from quarantine or social-distancing rules are lifted. When production lines have been destroyed in a natural disaster or social crisis, factories have to operate under low levels of productivity while rebuilding new production lines. This reduction in productivity will lead to extended processing times, which could impact the completion time of the orders, thus affecting the total cost. When rescheduling the production sequence under such scenarios, factories need to consider the changes in capacity or productivity.

Consider a case where a factory's productivity is affected following a disruption. In the first T_r periods, the factory produces with only a subset of its requisite workers and equipment. Thus, the processing time t'_{ij} in the first T_r periods becomes l times the original processing time, or $t'_{ij} = lt_{ij}$, where l could be any number greater than 1. Starting from period T_r+1, the factory is back to its full capacity. The processing time returns to the original processing time, $t'_{ij} = t_{ij}$. If a job is scheduled to be processed during the first T periods after disruption, its processing time will be l times the original processing time. However, if the job is scheduled to be processed after period T_r, its processing time will be normal. Therefore, we introduce a new variable (z_{ij}) to this model to denote whether or not job ij is scheduled to be completed before time T_r.

$$z_{ij} = \begin{cases} 1, & \text{if } c_{ij} \leq T_r; \\ 0, & \text{otherwise.} \end{cases}$$

The new processing time for job ij now becomes:

$$t'_{ij} = lt_{ij}z_{ij} + t_{ij}(1 - z_{ij}).$$

The rest of the notation is kept the same as in Section 3.1.

The rescheduling model for this extension shares the same objective function and constraints with the main model after replacing t_{ij} with t'_{ij}. The new constraints for this extension are as follows:

$$t'_{ij} = lt_{ij}z_{ij} + t_{ij}(1 - z_{ij}), \qquad \forall i, j \qquad (2a)$$
$$c_{ij} - T_r \leq M(1 - z_{ij}), \qquad \forall i, j \qquad (2b)$$
$$T_r - c_{ij} \leq Mz_{ij}, \qquad \forall i, j \qquad (2c)$$
$$z_{ij} \in \{0, 1\}, \qquad \forall i, j$$

M is the generic notation for a sufficiently large number. Constraints (2b) and (2c) guarantee that when $c_{ij} \leq T_r, z_{ij} = 1$ and when $c_{ij} \geq T_r, z_{ij} = 0$. Note that since the processing time is no longer a parameter, this model becomes a nonlinear mixed-integer problem. In Section 5, we test our algorithm on this model and present the results.

4. Two-stage genetic algorithm

A genetic algorithm, or GA, is a search algorithm inspired by the theory of natural evolution. A GA simulates the process of natural selection, where the best individuals are selected to produce offspring. GA is widely used in optimization and search problems due to its efficiency and has a higher chance of finding the global optimum than other algorithms. However, GA still has shortcomings. There is still some possibility of being trapped into the local optimum and having premature convergence. To overcome this issue, the estimation of distribution algorithms (EDA) is used to guide the search for the optimum employing an explicit probability distribution.

In this section, we propose the EAGA, a two-stage hybrid AGA that combines EDA with an adaptive local search operator. The EAGA works with a set of encoded individuals called a population. Each individual is represented by a string of genes called chromosome. The objective function value used to evaluate an individual is called *fitness*. The current population generates *offsprings* through the process of the EAGA. In the first stage of the EAGA, we use the AGA proposed by Liu et al. (2016). The dominant population obtained from the first stage is used as the initial population for the second stage. In the second stage of the EAGA, we use an EDA with an adaptive local search operator to explore the fitness of offspring and update the population until an optimal or near-optimal individual is obtained, or the maximum iteration is reached. The framework of the EAGA is shown in Fig. 2.

The EAGA is designed to greatly reduce the likelihood of the GA converging to a local optimum while still maintaining fast convergence. Specifically, the EDA in the second stage utilizes the global statistical information to ensure the diversity of the population in each generation. The adoption of the adaptive local search operator used in this algorithm guarantees the best individuals are retained in the next generation. We also note that the quality of the initial population heavily impacts the efficiency of the EDA. Thus, to ensure the rate of convergence of the algorithm, we use an AGA in the first stage to generate a high-quality initial population rather than to just randomly generate a population. Therefore, our two-stage algorithm aims to improve the efficiency and efficacy in searching for the optimal solution in contrast to the existing genetic algorithms. In the following parts, we explain the algorithm in detail.

4.1. Encoding and initializing the population

Before the EAGA starts the search for the optimal solution, the initial population should be generated. Many papers in job scheduling use a single-layer chromosome-encoding scheme. This approach encodes an individual using one chromosome to represent its job permutation. This encoding scheme works well for problems with simple job sequencing. However, in our model, jobs belong to different types, and we consider the setup cost incurred when switching production between types. The classic, single-layer chromosome-encoding approach is no longer suitable. Thus, we adopt a two-layer chromosome-encoding scheme to represent an individual. Before the upper layer is coded, all jobs (i.e., $\sum N_i$) from customers are indexed in increasing order. The upper layer represents job permutations, and the lower layer represents the job types (see Fig. 3). In our work, only individuals with a lower total cost after rescheduling will be retained in the initial population to ensure its quality.

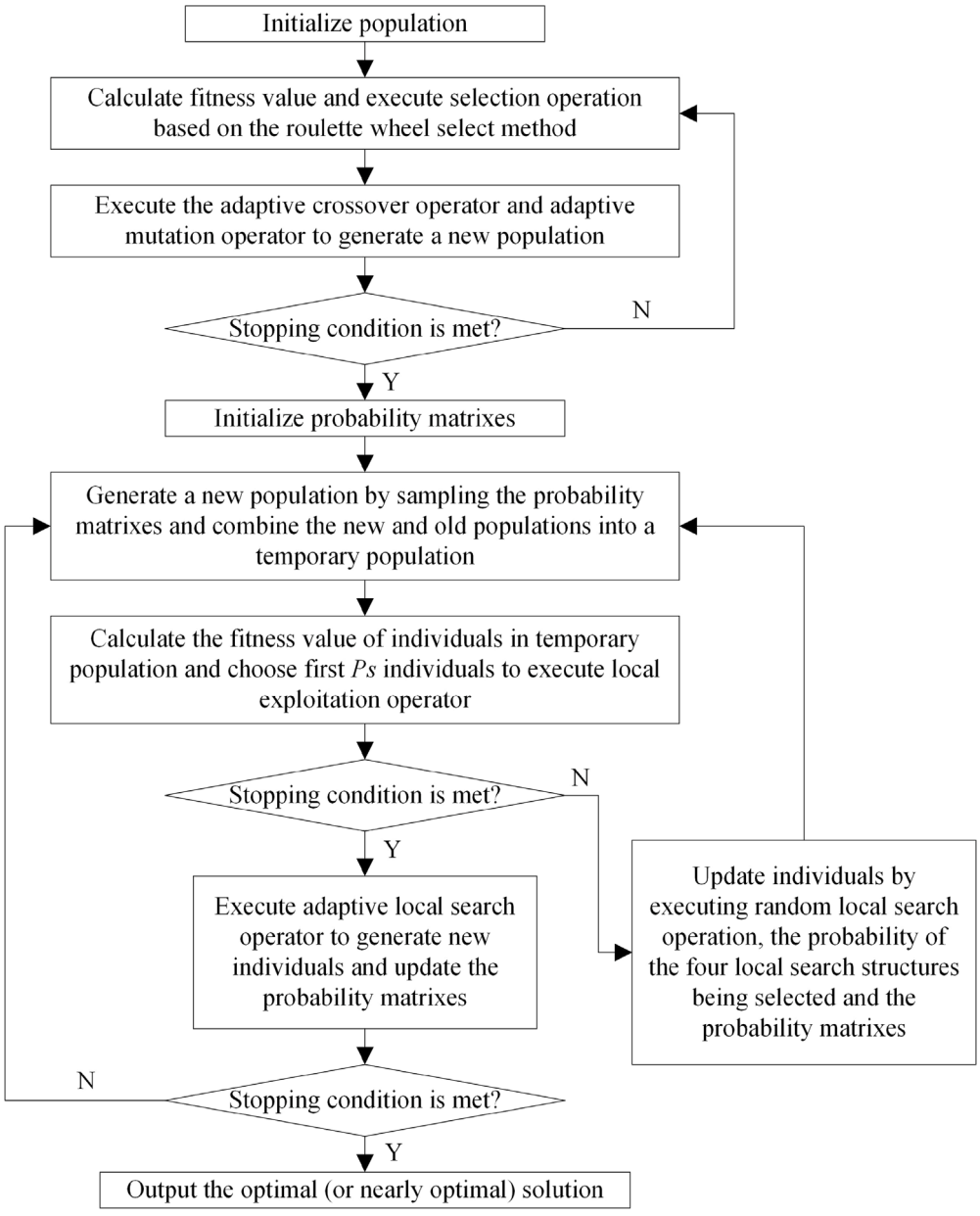


Fig. 2. The framework of EAGA.

Upper layer	**4**	1	2	5	3	8	6	7
Lower layer	**2**	1	3	1	2	3	3	1

Fig. 3. The two-layer chromosome-encoding scheme.

4.2. The first stage of EAGA

The purpose of the first stage is to obtain a high-quality population as the initial population for the second stage. The steps of the first stage are as follows:

1. Define parameters:
   - population size, Ps;
   - maximum iteration, $\text{iter}_{\max}$;
   - the minimum crossover probability, $Pc_{\min}$;
   - the maximum crossover probability, $Pc_{\max}$;
   - the minimum mutation probability, $Pm_{\min}$;
   - the maximum mutation probability, $Pm_{\max}$.

   Set starting time $T = 0$. Encode and initialize population.
2. Calculate the fitness of the individuals within in the population. Use the roulette-wheel selection method to select the parents. The selection probability of the yth individual, P_y, is calculated using the following function:

$$P_y = \frac{f_y}{\sum_{y=1}^{N} f_y},$$

   where f_y denotes the fitness of the yth individual.
3. Conduct the adaptive crossover operation based on the following crossover probability: Pc.

$$Pc = \begin{cases} Pc_{\max} - \frac{(Pc_{\max} - Pc_{\min})(f - f_{avg.})}{f_{\max} - f_{avg.}}, & f \geq f_{avg.} \\ Pc_{\max}, & \text{otherwise,} \end{cases} \quad (1)$$

where $f, f_{\max}, f_{avg.}$ denote the larger fitness of the two individuals being crossovered, the maximum fitness of the entire population, and the average fitness of the entire population, respectively.

4. Conduct the adaptive mutation operation based on the following mutation probability Pm.

$$Pm = \begin{cases} Pm_{\max} - \frac{(Pm_{\max} - Pm_{\min})(f - f_{avg.})}{f_{\max} - f_{avg.}}, & f \geq f_{avg.} \\ Pm_{\max}, & \text{otherwise.} \end{cases} \tag{2}$$

5. Update $t = t + 1$. If $t < \text{iter}_{\max}$, return to Step 2; otherwise, exit first stage.

In Step 3, the adaptive crossover probability is defined to accelerate evolution speed and increase individual diversities. This approach is adopted from Wang and Tang (2011). After the two chromosomes performed a crossover operation, if the larger fitness is greater than or equal to the population's average fitness (indicating a worse individual), its crossover probability decreases; otherwise, it remains unchanged.

4.3. The second stage of EAGA

In the second stage of the proposed algorithm, we search to discover whether better individuals exist. The final population from the first stage is used as the initial population for the second stage. In this section, we first introduce the algorithms used in the process; we then summarize the second stage at the end of this section.

4.3.1. Probability model and updating mechanism

Unlike GA, EDA generates offspring by sampling according to a probability model instead of crossover and mutation operators. Thus, the probability model in EDA has a great impact on the performance of the algorithm. In this stage, the probability is modeled by an $n \times n$ probability matrix, as follows:

$$P(t) = \begin{bmatrix} p_{11}(t) & p_{12}(t) & \dots & p_{1n}(t) \\ \dots & & & \\ p_{n1}(t) & p_{n2}(t) & \dots & p_{nn}(t) \end{bmatrix} \tag{3}$$

where $p_{jk}(t)$ denotes the probability that job j is processed in the kth position of the sequence in generation t. At the beginning of this stage ($t = 0$), we set all $p_{jk}(0) = 1/n$.

In generation t, we produce new individuals by sampling from the population using the probability matrix $P(t)$. To ensure that each job is assigned to one and only one position, a parameter r is randomly generated within the range of $(0, 1)$. For the kth position, we assign the first job j that satisfies the condition $r \leq \sum_{i=1}^{j} p_{ik}(t)$. To avoid a job j being assigned to other positions after it is already in position γ, all other $p_{jk}(t)$s are set to 0. At the same time, the probabilities of assigning other jobs to position γ are also set to 0. After generating a new individual, its fitness is evaluated. If the fitness after rescheduling is smaller than before rescheduling, the new individual will be retained in the population; otherwise, a new individual will be generated.

Next, we discuss how to update the probability matrix $P(t)$. In the updating mechanism, we select the top $\eta \cdot Ps$ individuals from the population as samples to update the probability matrix. Here, η is predefined and denotes the proportion of the top individuals in a population and Ps denotes the population size. In this paper, an incremental learning method is used to update the probability matrix. Define $\alpha \in (0, 1)$ as the learning rate. We update the probability matrix using the following equation:

$$p_{jk}(t+1) = (1-\alpha)p_{jk}(t) + \frac{\alpha}{\eta Ps} \sum_{l=1}^{\eta Ps} I_{jk}^{l}(t), \quad \forall j, k \tag{4}$$

where $I_{jk}^{l}(t)$ is the indicator function of the lth top individual in generation t. It is defined as:

$$I_{jk}^{l}(t) = \begin{cases} 1, & \text{if job } j \text{ is assigned to position } k \\ 0, & \text{otherwise.} \end{cases}$$

4.3.2. Adaptive local search operator

An effective local search can dramatically improve the performance of evolutionary algorithms (Wang et al., 2013). The EDA makes effective use of global statistical information rather than local information about individuals during the evolution process (Zhou et al., 2016). An efficient algorithm should find a balance between global information and local information. Therefore, we propose an adaptive local search operator to improve the local search performance of EDA. We mainly consider four local search structures: swap-based structure, insert-based structure, invert-based structure, and double swap-based structure. See Fig. 4 for an illustration of each of the structures. In each structure, the chromosome on the top is original sequence and the chromosome on the bottom is the new sequence after the local search operator.

Compared with the traditional local search, we propose a two-step adaptive local search operator. In the first step, we generate new individuals using the four local search structures and initialize the probability distribution, $Q(0)$, for the local search structure selection in the second step. The process to generate a new individual is as follows:

1. Randomly select a search structure and generate a new individual based on the structure.
2. Compare the fitness of the new individual with the original individual.
3. If the new individual is better, stop. If the old individual is better, randomly select another search structure from the rest of the structures.
4. Repeat 1, 2, and 3 until a better individual is found or all four structures have been tried. If no new individual is found after all four search structures, stop and discard the new individual. Move on to the next individual in the population.

In this process, an individual is only updated if the algorithm finds a new individual that is better than the original one. During the process, we record the number of successful updates of each local search structure ($N_i, i = 1, \dots, 4$). After all individuals are updated, we define $\hat{q}_i(0)$ as follows:

$$\hat{q}_i(0) = \frac{N_i}{\sum_{g=1}^{4} N_g} + \delta, \quad \forall i = 1, \dots, 4.$$

Here, we manually set $\delta = 0.05$ to avoid the probability of any local search structure being initialized to 0. Then, we normalize $\hat{q}_i(0)$ so that the sum of all four probabilities equals 1. The probability after normalization, $Q(0) = \{q_i(0)\}$, is used to select the local search structures in the next step.

In the second step, we use $Q(0)$ as the starting probability to select a local search structure. This probability is updated during the process using the number of successful individual updates. At the beginning of the second stage, we reset all N_is to 0. The probability of selecting search structure i in generation t is denoted as $q_i(t)$. In each generation, $q_i(t)$ is updated using the following equation:

$$q_i(t) = (1-\beta)q_i(t-1) + \beta \frac{N_i}{\sum_{g=1}^{4} N_g},$$

where $\beta \in (0, 1)$ is the predefined learning rate. Once we iterate through all the individuals in the population, we stop and exit the algorithm.

4.3.3. Process of the second stage

1. Reset t = 0. Define the learning rates α and β for $P(t)$ and $Q(t)$. Define the top percentage of the population, η. Initialize probability matrix $P(0)$.

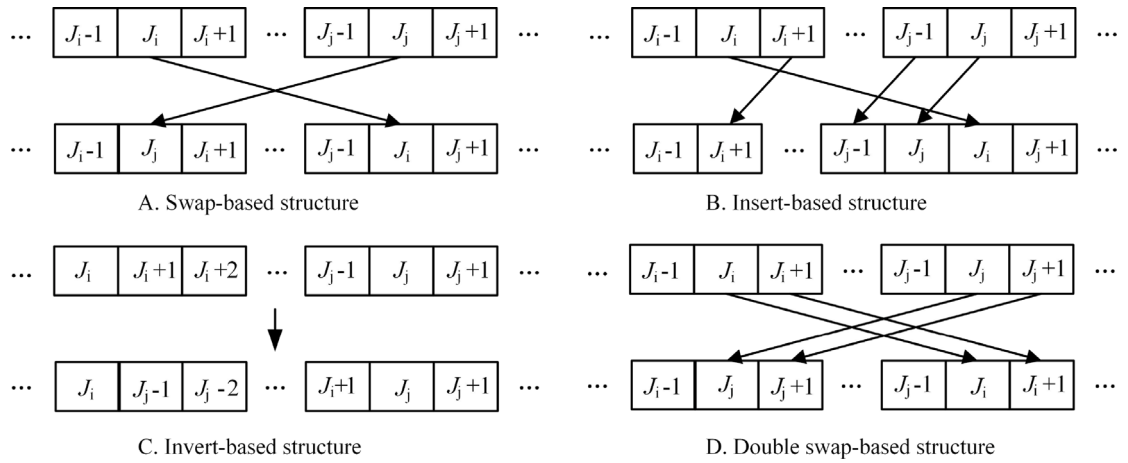


Fig. 4. Local search structures.

2. Generate a new population based on $P(0)$ and combine the new and old generation into a temporary population of size $2Ps$.
3. Calculate the fitness of each individual in the temporary population and sort individuals from the best to the worst.
4. Keep the first Ps individuals in the temporary population. Apply the two-step adaptive local search operator.
5. Update $t = t + 1$. If $t < \text{iter}_{max}$, return to Step 2. Otherwise, stop and exit the algorithm.

Once the first and second stages have been executed, the EAGA will return a production sequence and its fitness. In the next section, we show the performance of the EAGA and compare it with a few other algorithms from the models presented in Section 3.

5. Numerical results

To test the performance of the proposed model and the algorithm, we apply our model to a heavy-machinery service factory in Qingdao, China. In this section, we will discuss the result of the algorithm and compare the performance with a few recent and closely related algorithms used in scheduling and/or rescheduling.

This service factory provides maintenance and repair services to various forms of heavy machinery and automobiles, such as locomotives, cranes, and wheel loaders. Its customers are located in different regions of China. When the heavy machinery or automobiles are due for maintenance or need repair, they are sent to the service factory. The machinery first undergoes a thorough inspection by technicians at the factory, then services are scheduled, and orders for replacement parts are placed. The service factory has a contract manufacturer who produces all the replacement parts. The manufacturer receives orders from the service factory and sends parts to the service factory once production is done. When all parts are installed, and all services are completed, the machinery is sent back to the customer.

We obtain the service request data and production data from this factory. Based on the dataset, we generate some instances as test data. For each instance, we execute ten runs and collect the best fitness, the average fitness, and the worst fitness. The algorithm is coded in C# and executed on an Intel(R)-i5, 1.6 GHz computer with 4 GB of RAM.

Now, we introduce the parameters used in the numerical experiments. The parameters are approximated from the data gathered from the heavy-machinery service factory where the research project was conducted. In the numerical experiments, we set the parameters as follows:

- Penalty cost $\rho = 16$;
- Low inventory cost per unit $h_L = 4$;
- High inventory cost per unit $h_H = 6$;
- Setup cost, $F = 6000$;
- Minimum mutation probability, $Pm_{\min} = 0.01$;
- Maximum mutation probability, $Pm_{\max} = 0.1$;
- Minimum crossover probability, $Pc_{\min} = 0.6$;
- Maximum crossover probability, $Pc_{\max} = 0.9$;
- Population size, $Ps = 50$;
- Percent of top individuals, $\eta = 0.1$;
- Learning rate in the probability matrix in the EDA, $\alpha = 0.2$;
- Learning rate in the adaptive local search operator, $\beta = 0.2$;
- Number of job types, $G = 5$.

To show the necessity of using an approximation algorithm, we run our model for a real-life-size problem using CPLEX MILP solver. We compare the efficiency of the EAGA and the exact algorithm from the mixed-integer optimizer. We test our algorithm for order count $M = 20$, 50, and total job count of $\sum N_i = 1000, 1500$. For each instance, we generate five sets of order data. For each set of data, we calculate the percent of improvement from the exact algorithm to the EAGA. At the current problem scale, optimal solutions based on CPLEX are not obtainable within a reasonable amount of time. Thus, we set a fixed computation time of one hour for both algorithms and compare the solution quality. In the numerical experiments, it turns out that EAGA only takes 70 s to 210 s for the cases we tested, while CPLEX reaches the maximum time limit. We use *Reduction Percentage* (RP) to measure the cost reduction of EAGA when comparing with the exact scheduling algorithms based on CPLEX. Define RP as follows:

$$\text{RP} = \frac{\bar{C} - C}{\bar{C}} \times 100\%,$$

where $\bar{C}$ is the cost applying the CPLEX-based exact algorithm and C is the cost using our proposed rescheduling algorithm. A negative RP indicates an increase in the cost. The results are listed in Table 3.

Results show that, our algorithm greatly improves the solution quality compared with the exact algorithm in CPLEX. As illustrated in Table 3, the improvement can be up to 20.65% (that is, EAGA can reduce the cost by 20.65%). The overall average improvement is calculated as 12.13%. Thus, when applying this model to a practical scenario, exact algorithms tend to be time-consuming and a suboptimal solution from the exact algorithm could diverge significantly from the actual optimum. In this regard, using an approximation algorithm such as EAGA to solve the focal problem in this paper is necessary.

In the following subsections, we test our models and algorithm under various combinations of order counts and total job counts and compare the results with the initial schedule and various genetic algorithms. Due to the large number of test runs we need to conduct in the numerical experiments, running large datasets will be time-consuming. Thus, the scale of the problem has been reduced in the numerical experiments to gain efficiency.

Table 3
Total cost comparisons between EAGA and CPLEX-based exact algorithm.

$\sum N_i$	M	Run ID	Exact algorithm	EAGA	RP
1000	20	1	54151.26	49708.84	8.20%
		2	58509.76	49440.58	15.50%
		3	53975.12	48754.85	8.17%
		4	53563.40	49187.63	8.17%
		5	54558.76	47465.21	13.00%
		Average	54951.66	27649.91	10.97%
	50	1	57790.75	52819.98	8.60%
		2	60801.76	52309.59	13.97%
		3	60139.73	50287.61	16.38%
		4	59546.69	51199.68	14.02%
		5	58369.09	52263.51	10.46%
		Average	59329.60	51788.76	12.71%
1500	20	1	88747.64	70431.07	20.64%
		2	84722.29	75198.22	11.24%
		3	85194.96	81194.15	4.70%
		4	84390.13	76313.92	9.57%
		5	87150.91	75032.54	13.90%
		Average	86041.19	75633.98	12.10%
	50	1	91468.94	79440.58	13.15%
		2	91696.68	78105.27	14.82%
		3	91376.98	85426.79	6.50%
		4	90796.78	72050.18	20.65%
		5	91144.58	83287.98	8.62%
		Average	91296.79	79674.87	12.73%

5.1. Model comparison with the initial schedule

In the numerical experiments, we test 21 combinations of order counts (M) and the total job counts ($\sum N_i$). The number of jobs in each order is evenly distributed or as close to an even distribution as possible. In each table, the first column is the total number of jobs. The second column is the number of customer orders. We first compare the cost of the rescheduled production sequence with the initial schedule. The initial schedule is calculated at the beginning of the planning horizon (before disruption occurs) by minimizing the total holding cost and setup cost. The penalty cost, holding cost, fixed cost, and RP are shown in Table 4. In this part, when calculating RP, $\bar{C}$ represents the cost of applying the initial schedule to the problem with disruption. The first five columns are the total cost (TC), the total low holding cost (H_L), the total high holding cost (H_H), the total setup cost (S), and the penalty cost (P) of the initial schedule. The next few columns are the costs after rescheduling and the corresponding RP.

From Table 4, we can see that in all cases, the total cost after rescheduling is less than the initial schedule. In most cases, the RP of the total cost after rescheduling is over 20%. When there are more jobs, RP tends to be higher. From the next few columns, we observe that the main contribution of cost reduction is from high holding cost H_H and penalty cost P. At the same time, H_L and S are increasing, indicating that orders with an approaching deadline have higher priorities after rescheduling. Similar jobs that were scheduled consecutively in the initial schedule have to be broken into multiple setup rounds to avoid delays. Jobs from the same order are more likely to be produced in clusters, and the entire order takes less time to finish. Thus, the high holding cost of the finished parts is lower, and the total low holding cost of the raw material is higher.

We also run the same instances for the extension model with the reduced productivity, with $l = 2$. The results are shown in Table 5. Again, the first five columns show the costs from the initial schedule. The costs are higher than those in Table 4 because of the longer processing times following the supply chain disruption. The next few columns show the cost after rescheduling. From the table, we can see that rescheduling becomes crucial when processing capacity is limited for a period of time. In most cases, the RP of the total cost after rescheduling is over 25%, and some can go as high as 47%. Under this model, the low holding cost H_L and setup cost S increase after rescheduling. The main drivers of the total cost reduction are the high holding cost H_H and penalty cost P.

5.2. EAGA performance comparison with other algorithms

Next, we compare the performance of the EAGA with five closely-related genetic algorithms. We apply the following five genetic algorithms to the same scheduling model.

1. the original genetic algorithm (GA) (Holland, 1975)
2. the adjusted genetic algorithm (AGA) (Liu et al., 2016)
3. the hybrid adaptive genetic algorithm (HAGA) (Zhang et al., 2019)
4. the genetic algorithm with updated decoding process (GAD) (Yu et al., 2018)
5. a genetic algorithm for no-waiting scheduling (GANW) (Samarghandi, 2019).

To ensure the performance of each algorithm, we set the algorithm-specific parameters to the suggested values in their original work. Cost parameters are consistent with those used in our EAGA algorithm. Using these five algorithms, we calculate the same rescheduling problem and obtain the best fitness, the average fitness, and the worst fitness for each setting. We use the *Improvement Rate* (IR) to measure the percent improvement of the EAGA in contrast to other GAs. Here, we calculate the cost reduction of an algorithm by taking the difference between the initial schedule TC and the solution generated by this algorithm $TC_{alg.}$:

$$\text{cost reduction}_{alg.} = TC - TC_{alg.}.$$

Using cost reduction, we define IR as follows:

$$\begin{aligned}\text{IR} &= \frac{\text{cost reduction}_{EAGA} - \text{cost reduction}_{alg.}}{\text{cost reduction}_{EAGA}} \times 100\% \\ &= \frac{TC_{alg.} - TC_{EAGA}}{\text{cost reduction}_{EAGA}} \times 100\%.\end{aligned}$$

A positive IR indicates that the EAGA's reschedule solution has a lower cost than the benchmark algorithm. The larger the IR, the better the solution from EAGA in contrast to each specific algorithm.

For each algorithm, we execute ten runs and collect the best fitness, the average fitness, and the worst fitness to undertake the comparison. In Tables 6, 7, and 8, we show the total cost after rescheduling using each algorithm and the Improvement Rate (IR), as defined at the beginning of Section 5. From Table 6, we can see that when the number of jobs is small (below 20), most algorithms converge to the same solution in the best scenario. However, as the number of jobs increases, the advantage of the EAGA begins to show. The EAGA always has a lower total cost than the other five algorithms. As the number of jobs becomes larger, the IR of the EAGA tends to become larger. Even in the best scenario where we only compare the lowest total cost from the ten runs, the IR still ranges from $2\% - 45\%$. Comparing the five selected algorithms, the classic GA is the worst performer.

A similar pattern was observed in both Tables 7 and 8. When N is small, most algorithms converge to the same solution. As N becomes larger, the EAGA's cost reduction advantage becomes more evident in all instances. From Table 7 we can see that, on average, the EAGA has the highest improvement when the benchmark is the classic GA, which ranges between 4% and 49%, depending on the number of total jobs, with the majority falling between 10% and 40%. The other four algorithms perform better than the classic GA but are still below the EAGA, with the majority of IR falling between 5% and 18%.

In summary, the rescheduling models proposed in this work will greatly reduce the total cost when disruptions occur in the supply chain. The EAGA substantially outperforms the five closely-related genetic algorithms in all best, average, and worst scenarios. In addition, as the number of jobs becomes larger, the benefit of using the EAGA

Table 4
Total cost breakdown and cost reduction percentage of the main model.

$\sum N_i$	M	TC'	H'_L	H'_F	S'	P'	TC		H_L		H_H		S		P	
							Cost	RP	Cost	RP	Cost	RP	Cost	RP	Cost	RP
10	2	311.5	42.6	94.4	48.0	126.6	234.4	14.4%	51.1	1.3%	53.7	42.8%	24.0	50.0%	105.5	1.3%
	3	325.8	42.6	57.9	48.0	177.3	224.3	14.9%	56.5	−32.8%	12.8	77.8%	30.0	37.5%	125.0	29.5%
	5	350.6	42.6	29.8	48.0	230.2	304.1	7.6%	49.6	−16.5%	16.2	45.6%	24.0	50.0%	214.3	6.9%
20	2	744.2	104.4	332.7	96.0	211.1	540.3	20.8%	132.1	−26.5%	157.1	52.8%	84.0	12.5%	167.1	20.8%
	3	718.5	104.4	224.3	96.0	293.8	511.1	21.8%	130.1	−24.6%	79.1	64.7%	84.0	12.5%	217.8	25.9%
	5	876.8	104.4	222.3	96.0	454.2	641.1	21.1%	121.4	−16.3%	75.5	66.1%	84.0	12.5%	360.3	20.7%
30	2	1336.2	166.3	737.9	138.0	294.0	942.7	25.5%	224.8	−35.2%	346.2	53.1%	144.0	−4.4%	227.8	22.5%
	3	1435.1	166.3	699.9	138.0	430.9	929.3	28.8%	234.8	−41.2%	227.7	67.5%	156.0	−13.0%	310.8	27.9%
	5	1527.0	166.3	564.6	138.0	658.0	1008.9	28.3%	241.1	−45.0%	140.1	75.2%	150.0	−8.7%	477.7	27.4%
40	2	2112.0	227.9	1309.0	198.0	377.0	1492.0	27.1%	362.5	−59.0%	635.5	51.5%	204.0	−3.0%	290.0	23.1%
	3	2137.8	227.9	1174.2	198.0	537.7	1379.9	29.3%	381.9	−67.6%	397.2	66.2%	210.0	−6.1%	390.8	27.3%
	5	2487.6	227.9	1166.6	198.0	895.1	1472.4	32.2%	384.3	−68.6%	267.1	77.1%	216.0	−9.1%	605.0	32.4%
60	2	4013.4	348.8	2828.2	306.0	530.3	3039.1	25.3%	558.1	−60.0%	1720.4	39.2%	324.0	−5.9%	436.6	17.7%
	3	4278.2	348.8	2827.9	306.0	795.4	2669.4	31.2%	634.6	−81.9%	1132.2	60.0%	324.0	−5.9%	578.6	27.3%
	5	4639.2	348.8	2689.4	306.0	1295.0	2764.1	32.5%	757.5	−117.2%	787.4	70.7%	330.0	−7.8%	889.2	31.3%
80	2	6806.5	488.2	5217.4	396.0	704.9	5298.9	25.8%	924.7	−89.4%	3329.7	36.2%	456.0	−15.2%	588.6	16.5%
	3	6799.1	488.2	4872.6	396.0	1042.3	4769.7	28.7%	1039.0	−112.8%	2459.3	49.5%	456.0	−15.2%	815.4	21.8%
	5	7549.4	488.2	4947.9	396.0	1717.3	4355.7	35.4%	1232.3	−152.4%	1507.0	69.5%	450.0	−13.6%	1166.4	32.1%
100	2	10026.8	607.2	8054.1	504.0	861.5	7955.9	21.6%	1236.7	−103.7%	5422.8	32.7%	576.0	−14.3%	726.7	15.7%
	3	9997.6	607.2	7618.2	504.0	1268.2	7559.9	25.4%	1321.4	−117.6%	4628.5	39.2%	570.0	−13.1%	1040.0	18.0%
	5	10856.4	607.2	7645.8	504.0	2099.4	6889.4	31.7%	1623.8	−167.4%	3148.5	58.8%	582.0	−15.5%	1535.1	26.9%

Table 5
Total cost breakdown and cost reduction percentage of the extension model with reduced productivity.

$\sum N_i$	M	TC'	H'_L	H'_F	S'	P'	TC		H_L		H_H		S		P	
							Cost	RP	Cost	RP	Cost	RP	Cost	RP	Cost	RP
10	2	321.4	43.5	99.8	48.0	130.2	234.6	27.0%	53.9	−24.0%	51.2	48.7%	24.0	50.0%	105.5	18.9%
	3	332.8	43.5	60.6	48.0	180.9	190.9	42.6%	54.8	−26.2%	7.3	88.0%	48.0	0.0%	80.8	55.3%
	5	356.4	43.5	31.1	48.0	233.8	304.1	14.7%	49.6	−14.1%	16.2	48.0%	24.0	50.0%	214.3	8.4%
20	2	833.7	124.1	381.6	96.0	232.0	539.6	35.3%	130.1	−4.8%	158.4	58.5%	84.0	12.5%	167.1	28.0%
	3	804.0	124.1	260.7	96.0	323.2	424.6	47.2%	124.9	−0.6%	47.3	81.9%	84.0	12.5%	168.4	47.9%
	5	973.7	124.1	256.7	96.0	496.9	656.5	32.6%	120.6	2.9%	82.4	67.9%	90.0	6.3%	363.5	26.8%
30	2	1469.2	217.4	795.9	138.0	317.9	954.3	35.1%	232.9	−7.2%	342.2	57.0%	150.0	−8.7%	229.2	27.9%
	3	1580.0	217.4	757.9	138.0	466.8	972.8	38.4%	265.2	−22.0%	236.9	68.8%	162.0	−17.4%	308.6	33.9%
	5	1695.8	217.4	622.6	138.0	717.8	1069.9	36.9%	222.9	−2.6%	203.2	67.4%	144.0	−4.4%	499.8	30.4%
40	2	2261.5	302.2	1362.3	198.0	399.0	1518.2	32.9%	343.1	−13.5%	672.1	50.7%	204.0	−3.0%	299.0	25.1%
	3	2298.3	302.2	1227.5	198.0	570.6	1498.8	34.8%	348.7	−15.4%	525.6	57.2%	216.0	−9.1%	408.4	28.4%
	5	2670.0	302.2	1219.8	198.0	950.0	1578.9	40.9%	381.0	−26.1%	349.0	71.4%	210.0	−6.1%	638.9	32.8%
60	2	4224.0	482.2	2882.8	306.0	553.0	3168.7	25.0%	535.9	−11.1%	1847.8	35.9%	336.0	−9.8%	449.0	18.8%
	3	4500.1	482.2	2882.5	306.0	829.4	2891.3	35.8%	636.5	−32.0%	1315.5	54.4%	336.0	−9.8%	603.4	27.3%
	5	4883.8	482.2	2744.0	306.0	1351.6	2833.4	42.0%	653.2	−35.5%	938.7	65.8%	330.0	−7.8%	911.5	32.6%
80	2	7060.5	668.8	5269.4	396.0	726.4	5174.6	26.7%	815.1	−21.9%	3328.7	36.8%	450.0	−13.6%	580.8	20.0%
	3	7059.8	668.8	4920.5	396.0	1074.6	4798.6	32.0%	951.6	−42.3%	2573.3	47.7%	450.0	−13.6%	823.7	23.4%
	5	7835.7	668.8	4999.8	396.0	1771.1	4776.5	39.0%	1034.1	−54.6%	2032.6	59.4%	456.0	−15.2%	1253.8	29.2%
100	2	10338.5	845.0	8106.2	504.0	883.3	8022.8	22.4%	1307.6	−54.8%	5400.3	33.4%	582.0	−15.5%	732.9	17.0%
	3	10316.8	845.0	7667.0	504.0	1300.9	7367.0	28.6%	1399.7	−65.7%	4375.0	42.9%	570.0	−13.1%	1022.3	21.4%
	5	11200.8	845.0	7697.9	504.0	2153.9	7386.4	34.1%	1473.5	−74.4%	3743.4	51.4%	582.0	−15.5%	1588.6	26.2%

becomes more evident. We compare the computational time of each algorithm under different cases in Fig. 5. The results show that the GANW algorithm has a longer computational time than the others and the classic GA has the shortest computational time. The computational time of our algorithm (the black line) is also sufficiently short and does not increase dramatically along with the number of jobs. The results in Fig. 5, together with our results described above, further demonstrate that the proposed algorithm (i.e., the EAGA) can greatly improve the quality of the solution without sacrificing the computational time.

6. Conclusion and future work

In this paper, we focus on a rescheduling problem that arises from supply chain disruption. Supply chain disruptions can occur for various reasons, such as resource shortages, natural disasters, or social crises. The recent COVID-19 pandemic caused major supply chain disruptions all around the world. Cities were under quarantine for weeks. People were unable to go to work. Factories, including the one we obtained production data from, could not produce at their regular capacities. Because of the supply disruption of raw materials, many industries have suffered from production delays. When production is disrupted at factories, it is crucial to reconsider the production schedule based on customers' order requests and to reschedule production when necessary.

We study a single-machine rescheduling problem where a service provider (consists of a repair shop and a manufacturing facility) produces parts and provides maintenance services to customers in the presence of disruption. When regular production and services are disrupted, the service provider needs to optimally reschedule the jobs to minimize the total cost, including production setup costs, inventory holding costs (high and low, depending on whether production

Table 6
Comparison of the best values between the EAGA and other algorithms.

$\sum N_i$	M	GA	AGA	HAGA	GAD	GANW	EAGA	EAGA VS Others				(IR(%))
								GA	AGA	HAGA	GAD	GANW
	2	77.2	77.2	77.2	77.2	77.2	77.2	0.00	0.00	0.00	0.00	0.00
10	3	101.5	101.5	101.5	101.5	101.5	101.5	0.00	0.00	0.00	0.00	0.00
	5	46.5	46.5	46.5	46.5	46.5	46.5	0.00	0.00	0.00	0.00	0.00
	2	189.7	204.0	204.0	204.0	204.0	204.0	7.00	0.00	0.00	0.00	0.00
20	3	192.3	207.5	207.5	207.5	207.5	207.5	7.33	0.00	0.00	0.00	0.0
	5	231.0	235.7	235.7	235.7	235.7	235.7	2.01	0.00	0.00	0.00	0.00
	2	349.2	386.0	371.3	387.0	366.7	393.5	11.26	1.89	5.64	1.66	6.80
30	3	459.7	485.0	499.1	485.0	488.8	505.8	9.13	4.12	1.33	4.12	3.36
	5	469.9	494.5	497.7	497.7	491.2	518.0	9.28	4.54	3.93	3.93	5.17
	2	559.2	588.1	597.6	587.1	593.6	620.0	9.81	5.15	3.62	5.30	4.26
40	3	652.4	695.1	708.9	710.2	728.9	757.9	13.92	8.29	6.47	6.30	3.82
	5	855.2	901.1	978.8	969.2	950.6	1015.2	15.76	11.24	3.58	4.53	6.36
	2	793.5	876.4	891.2	887.6	876.4	974.2	18.56	10.04	8.52	8.89	10.04
60	3	1213.7	1472.4	1472.4	1487.7	1490.6	1608.7	24.55	8.48	8.48	7.53	7.35
	5	1517.8	1662.2	1780.4	1757.1	1755.5	1875.1	19.06	11.36	5.05	6.29	6.38
	2	1155.1	1329.5	1341.2	1332.3	1328.5	1507.6	23.38	11.81	11.04	11.63	11.88
80	3	1403.2	1718.7	1724.6	1758.1	1781.0	2029.4	30.86	15.31	15.02	13.37	12.24
	5	2053.2	2777.5	2767.2	2771.9	2786.3	3193.7	35.71	13.03	13.35	13.21	12.75
	2	1198.9	1726.7	1701.1	1714.6	1690.7	2070.9	42.11	16.62	17.86	17.21	18.36
100	3	1326.4	1966.5	1958.2	1997.9	1967.9	2437.7	45.59	19.33	19.67	18.04	9.27
	5	2482.6	3303.1	3296.1	3313.4	3114.1	3967.0	37.42	16.74	16.91	16.48	21.50

Table 7
Comparison of the average values between the EAGA and other algorithms.

$\sum N_i$	M	GA	AGA	HAGA	GAD	GANW	EAGA	EAGA VS Others				(IR(%))
								GA	AGA	HAGA	GAD	GANW
	2	74.9	77.2	77.2	77.2	77.2	77.2	3.02	0.00	0.00	0.00	0.00
10	3	96.9	101.5	101.5	101.5	101.5	101.5	4.48	0.00	0.00	0.00	0.00
	5	46.3	46.5	46.5	46.5	46.5	46.5	0.40	0.00	0.00	0.00	0.00
	2	174.0	193.9	189.7	195.4	192.4	198.9	12.53	2.51	4.63	1.75	3.28
20	3	177.2	193.9	196.0	194.9	196.5	202.0	12.29	4.03	2.98	3.54	2.76
	5	208.1	218.9	219.9	215.7	213.5	230.8	9.84	5.15	4.72	6.56	7.53
	2	311.9	340.2	339.2	343.9	335.6	380.9	18.11	10.68	10.95	9.69	11.88
30	3	406.4	444.8	457.9	445.8	422.1	487.1	16.56	8.67	5.99	8.47	13.34
	5	430.2	458.4	447.4	458.8	450.5	483.7	11.06	5.23	7.51	5.16	6.87
	2	468.3	511.4	511.7	516.4	517.0	571.8	18.11	10.58	10.52	9.70	9.59
40	3	584.1	641.0	657.1	657.6	619.6	712.4	18.01	10.02	7.76	7.70	13.02
	5	859.8	892.8	917.9	913.0	897.7	981.0	12.35	9.00	6.43	6.93	8.50
	2	720.4	788.9	803.7	804.8	796.2	898.4	19.82	12.19	10.54	10.42	11.38
60	3	1112.0	1324.9	1323.6	1341.2	1307.5	1467.6	24.23	9.73	9.81	8.62	10.91
	5	1363.8	1549.6	1672.0	1625.1	1684.6	1758.4	22.44	11.88	4.92	7.58	4.20
	2	787.3	1122.8	1101.1	1124.1	1156.0	1265.8	37.80	11.30	13.01	11.19	8.67
80	3	1166.0	1573.1	1572.0	1593.1	1595.6	1784.6	34.66	11.85	11.92	10.73	10.59
	5	1668.5	2348.0	2340.9	2341.8	2398.0	2676.2	37.66	12.27	12.53	12.50	10.40
	2	1000.4	1412.6	1432.9	1335.5	1445.6	1629.5	38.61	13.31	12.06	18.04	11.29
100	3	1097.8	1766.7	1744.3	1763.6	1790.2	2152.0	48.99	17.91	18.95	18.05	16.81
	5	2129.2	2903.5	2981.4	2944.7	2916.4	3596.9	40.80	19.28	17.11	18.13	18.92

is completed), and penalty costs for delayed deliveries. We formulate the rescheduling problem using a mixed integer linear program (MILP) model. An extension model with reduced productivity during the disruption recovery stage is also discussed. We design a two-stage genetic algorithm (EAGA), which combines the genetic algorithm with an adaptive local search operator to solve the model and efficiently search for the optimal solution. Our results show that our model greatly reduces the total cost and the EAGA algorithm outperforms a few related genetic algorithms for scheduling.

There are some limitations to our research, which give rise to a number of interesting areas for future work. First, this paper considers a single-machine rescheduling problem, while multiple-machine manufacturing is also common in practice. Our model can be extended to multiple-machine contexts to solve the rescheduling problems. Second, in this work, we focus on a deterministic model; however, uncertainties that could affect the rescheduling may arise from different parts of the system (for example, raw material and/or labor supply uncertainties during disruptions). In future studies, we could extend our model to capture these uncertainties. For example, to incorporate the stochastic processing time into our model, we could use a probability distribution to model processing times for the same types of jobs. Third, the system we analyzed is a static system where new job arrivals are not considered. In practice, new job arrivals could occur. Depending on the arrival of new jobs (deterministic or stochastic), future research could incorporate corresponding features into the model and make it a dynamic system. For all the proposed model extensions and variations

Table 8
Comparison of the worst values between the EAGA and other algorithms.

N	M	GA	AGA	HAGA	GAD	GANW	EAGA	EAGA VS Others				(IR(%))
								GA	AGA	HAGA	GAD	GANW
10	2	77.2	77.2	77.2	77.2	77.2	77.2	0.00	0.00	0.00	0.00	0.00
	3	99.7	101.5	101.5	101.5	101.5	101.5	1.68	0.00	0.00	0.00	0.00
	5	46.1	46.5	46.5	46.5	46.5	46.5	0.80	0.00	0.00	0.00	0.00
20	2	148.0	185.9	182.9	183.4	185.6	194.6	23.94	4.46	6.02	5.74	4.61
	3	159.6	184.1	190.4	192.2	191.8	195.4	18.36	5.80	2.60	1.64	1.87
	5	187.6	201.5	201.5	195.9	194.3	226.2	17.07	10.90	10.90	13.41	14.11
30	2	248.8	298.8	300.6	300.6	308.8	354.3	29.78	15.67	15.16	15.16	12.86
	3	313.3	388.2	393.5	395.4	374.7	477.4	34.36	18.67	17.56	17.16	21.51
	5	367.5	383.0	395.4	393.2	383.0	458.0	19.76	16.38	13.67	14.15	16.38
40	2	386.8	455.2	439.8	463.4	444.0	536.1	27.85	15.09	17.97	13.56	17.19
	3	510.0	562.8	562.8	585.6	541.1	682.5	25.27	17.53	17.53	14.20	20.72
	5	806.8	853.4	870.5	864.3	860.6	969.2	16.76	11.95	10.19	10.82	11.21
60	2	612.4	681.2	691.4	713.8	681.2	815.7	24.92	16.49	15.24	12.49	16.49
	3	984.8	1158.4	1145.8	1197.6	1128.0	1350.4	27.08	14.22	15.15	11.32	16.47
	5	1132.2	1420.2	1412.0	1470.8	1490.6	1654.8	31.58	14.17	14.67	11.12	9.92
80	2	512.0	921.0	717.7	947.0	947.0	1102.8	53.57	16.49	34.93	14.13	14.13
	3	1014.2	1290.2	1363.4	1382.7	1412.6	1583.8	35.96	18.54	13.92	12.70	10.81
	5	1457.7	1950.9	1958.2	1927.3	1919.4	2365.8	38.38	17.54	17.23	18.53	8.87
100	2	594.4	1095.9	955.5	922.6	1020.6	1323.2	55.08	17.18	27.79	30.27	22.87
	3	907.3	1561.5	1583.8	1511.0	1556.6	1899.7	52.24	17.80	16.63	20.46	18.06
	5	1673.5	2539.5	2300.1	2512.6	2271.3	3302.8	49.33	23.11	30.36	23.93	31.23

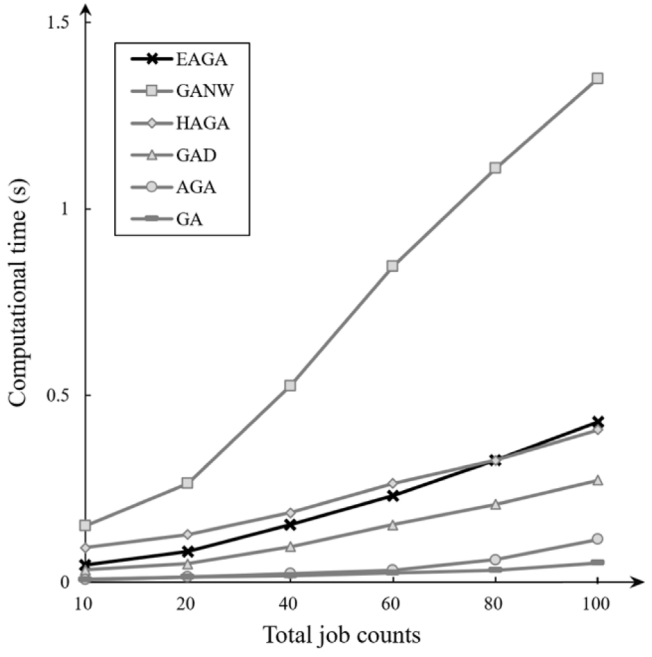


Fig. 5. Computational time of each algorithm under difference total job counts.

mentioned above, we could explore the performance of the EAGA applied to large-scale datasets.

Data availability

Test data are randomly-generated using a set of rules defined in the manuscript.

Acknowledgments

This work was supported by the National Social Science Fund of China [grant number 21AGL015].

References

Baardman, L., Cohen, M., Panchamgam, K., Perakis, G., Segev, D., 2019. Scheduling promotion vehicles to boost profits. Manage. Sci. 65 (1), 50–70.
Bodur, M., Luedtke, J.R., 2017. Mixed-integer rounding enhanced benders decomposition for multiclass service-system staffing and scheduling with arrival rate uncertainty. Manage. Sci. 63 (7), 2049–2395.
Coffman, Jr., E.G., Yannakakis, M., Magazine, M.J., Santos, C., 1990. Batch sizing and job sequencing on a single machine. Ann. Oper. Res. 26, 135–147.
Dobson, G., Karmarkar, U.S., Rummel, J.L., 1987. Batching to minimize flow times on one machine. Manage. Sci. 33 (6), 784–799.
Drozdowski, M., Jaehn, F., Paszkowski, R., 2017. Scheduling position-dependent maintenance operations. Oper. Res. 65 (6), 1657–1677.
Du, J., Leung, J.Y.T., 2017. Minimizing total tardiness on one machine is NP-hard. Math. Oper. Res. 15 (3), 483–495, http://www.jstor.org/stable/3689992.
Emadikhiav, M., Bergman, D., Day, R., 2020. Consistent routing and scheduling with simultaneous pickups and deliveries. Prod. Oper. Manage. 29 (8), 1937–1955.
Gao, K., Yang, F., Zhou, M., Suganthan, P.N., 2018. Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm. IEEE Trans. Cybern. 49 (5), 1944–1955.
Gmys, J., Mezmaz, M., Melab, N., Tuyttens, D., 2020. A computationally efficient branch-and-bound algorithm for the permutation flow-shop scheduling problem. European J. Oper. Res. 284 (3), 814–833.
Graves, S., 1979. Note—On the deterministic demand multi-product single-machine lot scheduling problem. Manage. Sci. 25 (3), 276–280.
Graves, S., 1981. A review of production scheduling. Oper. Res. 29 (4), 646–675.
Guido, R., Solina, V., Conforti, D., 2017. Offline patient admission scheduling problems. In: International Conference on Optimization and Decision Science, Vol. 2017. Springer, Cham, pp. 129–137.
Hall, N.G., Potts, C., 2003. Supply chain scheduling: Batching and delivery. Oper. Res. 51 (4), 566–584.
Hall, N.G., Potts, C., 2004. Rescheduling for new orders. Oper. Res. 52 (3), 440–453.
Helo, P., Phuong, D., Hao, Y., 2019. Cloud manufacturing–scheduling as a service for sheet metal manufacturing. Comput. Oper. Res. 110, 208–219.
Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA.
Hoogeveen, H., Lenté, C., T'kindt, V., 2012. Rescheduling for new orders on a single machine with setup times. European J. Oper. Res. 233 (1), 40–46.
Kilpatrick, J., 2022. Supply chain implications of the Russia-Ukraine conflict. Manage. Sci. 56 (9), 1551–1567, Deloitte Insights. https://www2.deloitte.com/us/en/insights/focus/supply-chain/supply-chain-war-russia-ukraine.html. Last checked June 24, 2022.
Kim, S.H., Cohen, M., Netessine, S., Veeraraghavan, S., 2010. Contracting for infrequent restoration and recovery of mission-critical systems. Manage. Sci. 56 (9), 1551–1567.
Kowalczyk, D., Leus, R., 2017. An exact algorithm for parallel machine scheduling with conflicts. J. Sched. 20, 355–372.
Levi, R., Magnanti, T., Shaposhnik, Y., 2019. Scheduling with testing. Manage. Sci. 65 (2), 776–793.

Li, F., Zhang, Y., Wei, H., Lai, X., 2019. Integrated problem of soaking pit heating and hot rolling scheduling in steel plants. Comput. Oper. Res. 108, 238–246.
Liu, A., Fowler, J., Pfund, M., 2016. Dynamic coordinated scheduling in the supply chain considering flexible routes. Int. J. Prod. Res. 54 (1), 322–335.
Liu, Z., Ro, Y.K., 2014. Rescheduling for machine disruption to minimize makespan and maximum lateness. J. Sched. 17, 339–352.
Magableh, G.M., 2021. Supply chains and the COVID-19 pandemic: A comprehensive framework. Eur. Manag. Rev. 18 (3), 363–382.
Marinova, G.I., Bitri, A.K., 2021. Challenges and opportunities for semiconductor and electronic design automation industry in post-Covid-19 years. In: IOP Conference Series: Materials Science and Engineering, Vol. 1208, No. 1. 012036.
Naddef, D., Santos, C., 1987. One-pass batching algorithms for the one machine problem. Discrete Appl. Math. 21, 133–145.
Portougal, V., Robb, D., 2000. Production scheduling theory: Just where is it applicable? INFORMS J. Appl. Anal. 30 (6), 64–76.
Samarghandi, H., 2019. Solving the no-wait job shop scheduling problem with due date constraints: A problem transformation approach. Comput. Ind. Eng. 136, 635–662.
Santos, C., Magazine, M.J., 1985. Batching in single operation manufacturing systems. Oper. Res. Lett. 4, 99–103.
Selvarajah, E., Steiner, G., 2009. Approximation algorithms for the supplier's supply chain scheduling problem to minimize delivery and inventory holding costs. Oper. Res. 57 (2), 426–438.
Sheffi, Y., 2005. The Resilient Enterprise: Overcoming Vulnerability for Competitive Advantage. MIT Press, Cambridge, MA.
Sheffi, Y., 2021. What everyone gets wrong about the never-ending COVID-19 supply chain crisis. MIT Sloan Manag. Rev. 63 (1), 1–5.
Soares, L., Carvalho, M.A.M., 2020. Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints. European J. Oper. Res. 285 (3), 955–964.
Steiner, G., Zhang, R., 2011. Revised delivery-time quotation in scheduling with tardiness penalties. Oper. Res. 59 (6), 1504–1511.
Sun, X.T., Chung, S.H., Chan, F.T.S., Wang, Z., 2020. The impact of liner shipping unreliability on the production–distribution scheduling of a decentralized manufacturing system. Transp. Res. E 114, 242–269.
Wang, L., Tang, D., 2011. An improved adaptive genetic algorithm based on hormone modulation mechanism for job-shop scheduling problem. Expert Syst. Appl. 38, 7243–7250.
Wang, L., Wang, S., Liu, M., 2013. A Pareto-based estimation of distribution algorithm for the multi-objective flexible job-shop scheduling problem. Int. J. Prod. Res. 51 (12), 3574–3592.
Yin, Y., Cheng, T.C.E., Wang, D.J., 2016. Rescheduling on iden-tical parallel machines with machine disruptions to minimize total completion time. European J. Oper. Res. 252 (3), 737–749.
Yu, C., Semeraro, Q., Matta, A., 2018. A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility. Comput. Oper. Res. 100, 211–229.
Zhang, S., Tang, F., Li, X., Liu, J., Zhang, B., 2021. A hybrid multi-objective approach for real-time flexible production scheduling and rescheduling under dynamic environment in Industry 4.0 context. Comput. Oper. Res. 132, 105267.
Zhang, H., Xie, J., Ge, J., Zhang, Z., Zong, B., 2019. A hybrid adaptively genetic algorithm for task scheduling problem in the phased array radar. European J. Oper. Res. 272 (3), 868–878.
Zhou, S., Li, X., Chen, H., Guo, C., 2016. Minimizing makespan in a no-wait flowshop with two batch processing machines using estimation of distribution algorithm. Int. J. Prod. Res. 54 (16), 4919–4937.