Annals of Operations Research
https://doi.org/10.1007/s10479-025-06482-2

ORIGINAL RESEARCH

A fully parallel multi-objective genetic algorithm for optimization of flexible shop floor production performance and schedule stability under dynamic environments

Jia Luo[1,2,3] · Didier El Baz[4] · Rui Xue[1] · Jinglu Hu[3] · Lei Shi[5,6]

Received: 27 April 2023 / Accepted: 9 January 2025

Abstract
As the work environment changes dynamically in real-world manufacturing systems, the dynamic flexible job shop scheduling is an essential problem in operations research. Some works have taken rescheduling approaches to solve it as the multi-objective optimization problem. However, previous studies focus more on solution quality improvements while ignoring computation time. To get a quick response in the dynamic scenario, this paper develops a fully parallel Non-dominated Sorting Genetic Algorithm-II (NSGA-II) on GPUs and uses it to solve the multi-objective dynamic flexible job shop scheduling problem. The mathematical model is NP-hard which considers new arrival jobs and seeks a trade-off between shop efficiency and schedule stability. The proposed algorithm can be executed entirely on GPUs with minimal data exchange while parallel strategies are used to accelerate ranking and crowding mechanisms. Finally, numerical experiments are conducted. As our approach keeps the original structure of the conventional NSGA-II without sacrificing the solutions' quality, it gains better performance than other GPU-based parallel methods from four metrics. Moreover, a case study of a large-size instance is simulated at the end and displays the conflicting relationship between the two objectives.

Keywords Evolutionary computations · Parallel NSGA-II · GPU computing · Multi-objective optimization · Flexible job shop scheduling · Dynamic scheduling

1 Introduction

Scheduling is a critical activity in production management. The Flexible Job Shop Scheduling Problem (FJSSP) incorporates machine assignment and operation sequencing where machine assignment deals with choosing a capable one in the set of machines to proceed each operation and operation sequencing works for sequencing the assigned operations on the machines (Hu et al., 2024). The FJSSP is an NP-hard problem (Xu et al., 2024) and usually is modeled as a static problem in the operations research literature. However, the working environment changes dynamically in real-world manufacturing systems and rescheduling is used most commonly to manage unpredictable real-time events. An updated schedule may be totally

Extended author information available on the last page of the article

different from the original one. If at least one objective deals with schedule stability, the Dynamic Flexible Job Shop Scheduling Problem (DFJSSP) is considered as a multi-objective optimization problem. Some works (Akram et al., 2024; Baykasoğlu et al., 2020; Liu et al., 2024; Shen & Yao, 2015) have integrated rescheduling approaches with specific algorithms to solve the Multi-Objective Dynamic Flexible Job Shop Scheduling Problems (MODFJSSP). Unfortunately, they consider only solution quality improvements while ignoring computation time. As rescheduling is a short-term decision-making process that composes and updates the schedule according to the current state of the system and the overall system requirements (Baykasoğlu et al., 2020), a method proposing an adequate rescheduling plan in a short response time is greatly desired in this case.

Evolutionary algorithms are widely used for solving FJSSP (Li et al., 2022; Mahmud et al., 2022; Zhang et al., 2020) although the repeated fitness function segment makes the time cost to find adequate solutions increased when it is applied to solve complex and large problems. Multi-Objective Evolutionary Algorithms (MOEAs) keep the main structure of evolutionary algorithms while it is capable of obtaining a set of well trade-off solutions in a single run (Tran & Luong, 2024). Non-dominated Sorting Genetic Algorithm-II (NSGA-II) (Deb et al., 2002) is probably one of the most widely studied MOEAs. It builds on the principles of the Genetic Algorithm (GA) (Kacem & Dammak, 2021) but extends them to handle multiple objectives. Some works (Ahmadi et al., 2016; Luan et al., 2023) have utilized NSGA-II to solve MOFJSSP. However, MODFJSSP is more complex than FJSSP and NSGA-II is computationally expensive as it needs not only to execute the repeated fitness function segment but also to explore larger portions of the search space for seeking the entire Pareto front. Therefore, how to decrease the execution time when NSGA-II is applied to solve MODFJSSP deserves more attention while this issue has not been well addressed so far.

Parallelization is one of the most frequently used methods to save computational time for time-sensitive issues and Graphics Processing Units (GPUs) parallelization can generally achieve promising speed-ups over sequential implementations. Solving optimization problems with parallel evolutionary algorithms and GPU computing (Harada & Alba, 2020; Schryen, 2020) has a long tradition. On one hand, research on GPU-based evolutionary approaches for solving multi-objective dynamic shop scheduling problems (Luo et al., 2019, 2020) has won favor in recent years where multiple objectives are weighted into a single one in most cases. On the other hand, some literature (Agarwal et al., 2015; Aguilar-Rivera, 2020; Kim & Kim, 2024; Wong & Cui, 2013) has shown the potential of combining GPU accelerated code with the NSGA-II to solve optimization problems, but the complex problem of MODFJSSP has not been considered as far as our knowledge is concerned. Moreover, it is difficult to execute the NSGA-II completely on GPUs due to its ranking and crowding mechanisms (Luna & Alba, 2015) while overheads are increased when partial parallelization is implemented.

Consequently, the MODFJSSP is a complex and computationally intensive challenge, especially in dynamic manufacturing environments where real-time rescheduling is crucial. Traditional approaches like the NSGA-II algorithm, although effective, are hindered by high computational costs and inefficiencies when applied to large, complex problems. GPU parallelization offers a promising avenue to address these issues, yet fully integrating NSGA-II with GPU architecture remains a significant challenge due to the algorithm's complex ranking and crowding mechanisms. This research aims to solve the MODFJSSP by developing a fully parallel NSGA-II implementation on CUDA (Compute Unified Device Architecture), leveraging specialized data structures and a combined fitness function to optimize performance. Particularly, the contributions of this paper are summarized as follows:

1. A MODFJSSP model is studied which considers new arrival jobs and seeks a trade-off between shop efficiency and schedule stability.
2. The machine-idle-driven strategy is utilized to reschedule new arrival jobs where the machine resources are fully used.
3. Respecting the CUDA underlying architecture, three data structures are designed to move all NSGA-II components to be run on GPUs with minimal data exchange between the host and the device.
4. A dummy fitness function is built by combining two factors into one which enables the elitist preservation strategy of NSGA-II to be executed in parallel.
5. Numerical experiments are carried out and show that the proposed solving approach can solve the MODFJSSP efficiently with competitive results.

The remaining sections of this paper are organized as follows. Section 2 introduces related works. Section 3 describes the dynamic flexible job shop problem and the multi-objective optimization mathematical model. Section 4 presents the solving approach. Section 5 details the GPU-based fully parallel NSGA-II and its implementations. Section 6 illustrates numerical experiments and results analysis. Finally, Sect. 7 states conclusions and future works.

2 Related works

The FJSSP has received lots of attention from academia and industry for many years (Gao et al., 2019; Türkyılmaz et al., 2020; Xie et al., 2019) and the DFJSSP is one of the most important topics in this domain.

Zadeh et al. (2019) proposed an improved artificial bee colony algorithm to address the makespan minimization in the DFJSSP considering variable processing times. Luo (2020) utilized a deep Q-network and deep reinforcement learning to minimize the total tardiness in a DFJSSP which takes new job insertions into consideration. Li et al. (2021) studied a DFJSSP with four dynamic events where an MCTS-based algorithm and multiple continuous specified time windows were designed for minimizing the makespan. Although most researchers consider the DFJSSP as a single objective problem, there is a growing interest in MODFJSSP. Reddy et al. (2018) intended to minimize both makespan and total machine load variation in a DFJSSP under the condition of machines breakdown and solved this problem with an effective hybrid evolutionary algorithm. Zhang et al. (2022) designed a multitask multiobjective genetic programming for automated scheduling heuristic learning to minimize max-tardiness, mean-tardiness, max-flowtime and mean-flowtime in a DFJSSP where new arrival jobs were considered. Akram et al. (2024) established a novel black widow spider algorithm to address a DFJSSP with insertion of new jobs for optimization objectives: makespan, total energy consumption and schedule instability. Wang et al. (2021) presented an improved squirrel search algorithm to seek the balance between operational efficiency and system stability in a DFJSSP which took jobs arrival and departure, machines breakdown and recovery into account. In addition to traditional shop efficiency objectives, new objectives have received more concerns in recent years. Schedule stability is one of the most significant criteria since the deviation from the previous schedule has a great impact on the overall performance in dynamic scheduling. The JSSP is well known as an NP-hard combinatorial optimization problem. When more constraints and objectives are considered, the complexity of the problem increases. Therefore, the MODFJSSP is more complex than JSSP, FJSSP and DFJSSP.

Various solution techniques and approaches have been used to solve the MODFJSSP concerning both efficiency and stability objectives. Baykasoğlu et al. (2020) studied the MODFJSSP by the greedy randomized adaptive search where the lexicographic method was used to evaluate four objectives including the schedule stability. Fattahi et al. (2010) took the genetic algorithm to address the MODFJSSP where the makespan, the starting time deviation and the total deviation penalty were weighted linearly into a single combined criterion. Shen et al. (2015) developed a multi-objective evolutionary algorithm-based method for solving the MODFJSSP where three efficiency objectives and one stability objective were optimized based on the Pareto dominance. Although the lexicographic approach and the utility function method can solve multi-objective dynamic shop scheduling problems effectively, it is hard to have sufficient preference information before the solution process when real-world implementations are carried out. On the opposite, the Pareto optimal solution is more complex as it is a posteriori approach where a group of optimal solutions can be obtained (Minella et al., 2008) within various trade-offs among several conflicting objectives provided by the Pareto front.

Many Pareto-based MOEAs have been used to solve MOFJSSP and most of them adopt the ranking and crowding mechanisms (Luna & Alba, 2015). Since particle swarm optimization cannot be directly applied to MOFJSSP (Nouiri et al., 2018; Zarrouk et al., 2022), Shao et al. (2013) proposed a hybrid algorithm where the Pareto ranking and crowding distance method were incorporated to identify the fitness of particles. Kefalas et al. (2019) studied a tabu search-based memetic algorithm to solve a MOFJSSP while using the nondominated ranking and the crowding distance for parent selection for the next generation after merging the offspring with the current parent population. Xiong et al. (2012) kept the classic ranking mechanism but modified the crowding distance as a measure in decision space. The efficiency and stability of the developed algorithm were verified by experimental results on well-known MOFJSSP instances. The overall complexity of the algorithms addressed in the three above-mentioned references is $O(MN^3)$ where M is the number of objective functions and N is the number of individuals. The storage requirement is $O(N)$. Both of them are governed by the ranking and crowding mechanisms. Zhang et al. (2018) established an energy-efficient MOFJSSP model and used the NSGA-II to obtain an approximation of the Pareto front where the quality of solutions was ensured by the fast non-dominated sorting, the crowding distance and the elite-preserving mechanisms. The overall complexity of the NSGA-II is decreased to $O(MN^2)$ owing to the fast non-dominated sorting while the storage requirement is increased to $O(N^2)$. Most ranking and crowding mechanisms used in the literature are inherited from NSGA-II (Luna & Alba, 2015) and it has been the most studied algorithm in the domain of multi-objective scheduling since 2014 (Rahimi et al., 2022).

MODFJSSP is an important subtype of multi-objective scheduling problems and many researchers have used NSGA-II to solve MODFJSSP. Ahmadi et al. (2016) applied the NSGA-II to combine the improvement of makespan and stability in the MODFJSSP with random machine breakdown. Zhang et al. (2019) incorporate the NSGA-II into the genetic programming hyper-heuristics framework to achieve a trade-off between different objectives in the MODFJSSP where new jobs arrived stochastically. Li et al. (2020) took the NSGA-II and the scroll window technology to solve the MODFJSSP considering three distributions where normal order addition, urgent order insertion and machine failure were all discussed. Genetic programming and scroll window technology could enhance NSGA-II to solve dynamic problems. However, the overall complexity and the storage requirement of the algorithm are not improved due to the fast non-dominated sorting. Moreover, the computational burden of NSGA-II is $O(MN^2)$ in the worst case for all the approaches although some studies have tried to decrease it for some instances (Ortega et al., 2017). To sum up, both NSGA-II and

enhanced NSGA-II are computationally expensive. Therefore, although they are efficient to solve MOFJSSP, there is a big challenge to apply NSGA-II to large-size MODFJSSP problems as obtaining the renewed adequate scheduling with a quick response is highly desired under dynamic environments.

With the development of parallel computing, both real-life problems and benchmark problems have verified that parallelization on GPUs and CUDA can help NSGA-II to achieve great speedups. Wong et al. (2013) proposed a parallel NSGA-II to solve a real-life direct marketing problem where the whole algorithm was executed on GPUs except for the non-dominated selection procedure. Agarwal et al. (2015) addressed the scalability of the multi-robot coalition formation problem by the parallel NSGA-II on GPU architecture where the master–slave model is employed. The accuracy of NSGA-II does not get changed with GPU implementations in these two applications while the actual execution time is decreased. However, the overall complexity and the storage requirement of the algorithm are always kept the same as in Deb et al. (2002) since the fast non-dominated sorting is executed intactly on the CPU. Padurariu et al. (2014) investigated the benefits of GPU implementation for NSGA-II and evaluated its performance on benchmark problems where the elitist strategy and the crowding-distance computation were executed on the CPU. Gupta et al. (2015) proposed a GPU-based parallel NSGA-II implementation on benchmark problems with a major focus on non-dominated sorting as it was the most time-consuming step while other steps were relatively negligible. Instead of the fast non-dominated sorting, the traditional design is used in the two above-mentioned cases as it can be executed with a simpler data structure. However, the traditional non-dominated sorting performs $O(MN^3)$ comparisons even though this procedure can be accelerated by GPU parallelization. Moreover, most GPU implementations for the NSGA-II are concerned only with partial parallelization. Clearly, the best performance is obtained when transferring data to GPUs once at the beginning of the application where the calculation is performed as much as possible on the device and results are sent back only at the end. Partial parallelization generally results in frequent communication between CPU and GPUs overhead which may offset the effectiveness gains from GPU acceleration.

The MODFJSSP considering both shop efficiency and schedule stability has been studied in recent publications. However, how to obtain the updated scheduling within a short response time is rarely discussed and none of them, to the best of our knowledge, has considered using GPUs to save execution time from the Pareto solutions searching procedure. On the other hand, the NSGA-II is widely used for solving the multi-objective FJSSP while it is difficult to meet the time requirement in dynamic environments. Although GPU computing can be utilized, the speed-up is limited as it is hard to execute the entire NSGA-II on GPUs and frequent data exchange is inevitable in partial parallelization. Therefore, this paper focuses on filling these research gaps by developing a GPU-based fully parallel NSGA-II for solving the MODFJSSP considering both efficiency and effectiveness. This implementation is highly desired, particularly for large-scale manufacturing problems.

3 Problem description

3.1 MODFJSSP description

The FJSSP is a generalization of the Job Shop Scheduling Problem (JSSP) (Vallikavungal Devassia et al., 2018). The JSSP mainly deals with determining the best sequence for processing jobs where each job consists of a certain number of consecutive operations and each

operation is processed by a particular machine. The FJSSP could be treated as the multi-purpose machine JSSP where each operation is allowed to be processed on any machine among a set of available machines. According to whether all operations can be processed on all machines, the FJSSP can be classified into two categories (Kacem et al., 2002a): (1) total FJSSP that each operation can be processed on all machines; (2) partial FJSSP that at least one operation can be processed on a subset of machines. The total FJSSP can be considered as a special case of the partial FJSSP.

The MODFJSSP is a further development of the FJSSP where the working environment changes dynamically by unpredictable real-time events. In this paper, only the job arrival event which is the most frequent and common factor in the shop floor (Zhang et al., 2019), is taken into consideration. If the number of idle machines reaches a specific level and there are certain new arrival jobs, the rescheduling is triggered. The new arrival jobs should be processed sequentially and non-preemptively from the beginning of the rescheduling point with the remaining uncompleted operations of the original jobs. Processing time for any operation of all jobs on any available machine and the original schedule are known. To decrease disturbances, the MODFJSSP is constructed by using a multi-objective performance measure as the objective function. Particularly, the first objective deals with efficiency as measured by makespan and total tardiness while the second objective considers stability as measured by starting time deviation and processing machine deviation.

3.2 Mathematical model

To describe the MODFFJSSP model, we summarize the used notations in Table 1.

The mathematical model for the MODFJSSP at a specific rescheduling point is derived from the models presented in Demir and İşleyen (2013), Luo et al., (2019), where Luo et al., (2019) studies rescheduling in flexible flow shop scheduling, while (Demir & İşleyen, 2013) outlines the general mathematical models for FJSSP. In terms of both shop efficiency and schedule stability, the formalization is as follows:

Objective Function:

$$\min: \ \text{efficiency} = 2 \times TT + 5 \times C_{max} \tag{1}$$

$$\min: \ \text{stability} = TD + MD \tag{2}$$

Subject to:

$$TT = \sum_{j \in J \cup J'} T_j = \sum_{j \in J \cup J'} \left(\max \left(S'_{j(o_j-1)m} + \sum_{m \in K_{js}} P_{j(o_j-1)m} \times x_{j(o_j-1)m} - D_j, 0 \right) \right) \tag{3}$$

$$C_{\max} = \max_{j \in J \cup J'} C_j = \max_{j \in J \cup J'} \left(S'_{j(o_j-1)m} + \sum_{m \in K_{js}} P_{j(o_j-1)m} \times x_{j(o_j-1)m} \right) \tag{4}$$

$$TD = \sum_{j \in J} \sum_{s \in O_j} \sum_{m \in K_{js}} \left| S'_{jsm} - S_{jsm^b} \right| \times x_{jsm} \tag{5}$$

$$MD = \sum_{j \in J} \sum_{s \in O_j} \sum_{m \in K_{js}} \left| m - m^b \right| \times x_{jsm} \tag{6}$$

$$\sum_{m \in K_{js}} x_{jsm} = 1 \quad j \in J \cup J', \quad s \in O_j \tag{7}$$

$$S'_{j0m} \geq R_j \geq 0 \quad j \in J \cup J', \quad m \in K_{j0} \tag{8}$$

$$S'_{jsm} \geq S'_{j(s-1)m'} + \sum_{m' \in K_{j(s-1)}} P_{j(s-1)m'} \times x_{j(s-1)m'} j \in J \cup J', s \in O_j, m \in K_{js}, s > 0 \tag{9}$$

$$S'_{irm} \geq S'_{jsm} + \sum_{m \in K_{js}} P_{jsm} \times x_{jsm} j \in J \cup J', i \in J \cup J', j \neq i, s \in O_j, r \in O_i, m \in K_{ir} \tag{10}$$

$$S'_{jsm} \geq RS \quad j \in J \cup J', \quad s \in O_j, \quad m \in K_{js} \tag{11}$$

$$RS \geq S_{j0m^b} \quad j \in J \tag{12}$$

$$\sum_{m \in K} y_m(t) \geq \theta \quad t \in H \tag{13}$$

$$x_{jsm} = \begin{cases} 1 \text{ if operation } s \text{ of job } j \text{ is processed on machine } m \quad j \in J \cup J', \quad s \in O_j, \quad m \in K_{js} \\ 0 \text{ otherwise} \end{cases} \tag{14}$$

Table 1 Notations used in the mathematical model

	Notation	Description
Sets and indices	j, i	Job indices
	s, r	Operation indices
	m, m'	Machine index
	t	Time index
	J	Set of original jobs, $J = \{0, 1, 2, \ldots, n-1\}$
	J'	Set of new arrival jobs, $J' = \{0, 1, 2, \ldots, n'-1\}$
	O_j	Set of operations of job j, $O_j = \{0, 1, 2, \ldots, o_j - 1\}$
	K_{js}	Set of available machines for operation s of job j, $K_{js} = \{0, 1, 2, \ldots, k_{js} - 1\}$
	K	Set of total machines, $K = \{0, 1, 2, \ldots, k-1\}$
	H	Set of time periods, $H = \{1, 2, 3, \ldots, h\}$
Parameters	n	Number of original jobs
	n'	Number of new arrival jobs
	o_j	Number of operations of job j
	k_{js}	Number of available machines for operation s of job j
	k	Number of total machines
	h	Time horizon
	θ	Threshold number of idle machines triggering the rescheduling point
	M_m	Machine m, $m \in K$

Table 1 (continued)

	Notation	Description
	J_j	Job j, $j \in J \cup J'$
	$o_{j,s}$	Operation s of job j, $j \in J \cup J', s \in O_j$
	R_j	Release time of job j, $j \in J \cup J'$
	D_j	Due time of job j, $j \in J \cup J'$
	P_{jsm}	Processing time when operation s of job j is processed on machine m, $j \in J \cup J', s \in O_j, m \in K_{js}$
	m^b	Machine assigned to operation s of original job j before the *RS* is executed
	S_{jsm^b}	Start time of operation s of original job j on machine m^b before the *RS* is executed, $j \in J, s \in O_j$
	RS	Rescheduling point
Performance metrics	T_j	Tardiness of job$j$, $j \in J \cup J'$
	TT	Total tardiness
	C_j	Completion time of job j, $j \in J \cup J'$
	C_{max}	Completion time of the last job, i.e., the makespan
	TD	Total starting time deviation
	MD	Total processing machine deviation
	$LB(T)$	Lower bound for total tardiness
	$LB(C_{max})$	Lower bound for the makespan
	$LB(TD)$	Lower bound of total starting time deviation
	$LB(MD)$	Lower bound of total processing machine deviation
Continuous decision variables	S'_{jsm}	Start time of operation s of job j on machine m after the *RS* is executed, $j \in J \cup J', s \in O_j, m \in K_{js}$
Binary decision variables	x_{jsm}	Indicating whether operation s of job j is assigned to and processed on machine m, $j \in J \cup J', s \in O_j, m \in K_{js}$
	$y_m(t)$	Indicating whether machine m is idle at time t, $m \in K, t \in H$

$$y_m(t) = \begin{cases} 1 \text{ if machine } m \text{ is idle at time } t \quad m \in K, \quad t \in H \\ 0 \text{ otherwise} \end{cases} \tag{15}$$

The optimization objectives are presented as shop efficiency measured by (1) and schedule stability measured by (2). The shop efficiency consists of two factors. The first one is total tardiness defined by criterion (3) and the second one is makespan defined by criterion (4). The weights in (1) are suggested and verified by Fattahi and Fallahi (2010), Ishibuchi and Murata (1998), Rangsaritratsamee et al., (2004) as the variance of the makespan is much smaller than that of tardiness. The schedule stability is also described by two components. The first one is the total starting time deviation defined by criterion (5) and the second one is

the total processing machine deviation defined by criterion (6). Constraint (7) states that each operation can only be handled by one machine. Constraints (8) and (9) define the precedence among operations, with constraint (8) addressing the start time for the first operation and constraint (9) specifying the start times for subsequent operations. Constraint (10) describes the precedence dictated by machine sequencing, where an operation can only begin once the previous operation assigned to the same machine has been completed. Constraints (11) and (12) establish the rescheduling criteria, with constraint (11) ensuring that the updated start times of operations are no earlier than the rescheduling point, and constraint (12) requiring that the rescheduling point occur after the start of the first operation of all original jobs. Constraint (13) outlines that rescheduling is triggered when the number of idle machines exceeds the threshold. Finally, constraint (14) introduces the Boolean decision variable for machine assignment, and constraint (15) presents the Boolean decision variable for machine idleness.

The lower bound for TT is calculated as the worst-case tardiness among all jobs: $LB(TT) = \max_{j\in J\cup J'}\left(\max\left(0, \sum_{s\in O_j} \min_{m\in K_{js}} P_{jsm} - D_j\right)\right)$. The lower bound on $C_{\max}$ is calculated as the maximum value between the maximum total processing time for any single job and the maximum workload on any machine: $LB(C_{max}) = \max(\max_{j\in J\cup J'}(\sum_{s\in O_j} \min_{m\in K_{js}} P_{jsm}),\ \max_{m\in K}(\sum_{j\in J\cup J'} \sum_{s\in O_j, m\in K_{js}} P_{jsm}))$. Therefore, the lower bound for shop efficiency is computed as $2 \times LB(TT) + 5 \times LB(C_{max})$. The lower bound for TD is represented as the minimum starting time deviation across all machines for each operation s of original job j: $LB(TD) = \sum_{j\in J} \sum_{s\in O_j} \min_{m\in K_{js}} \left|S'_{jsm} - S_{jsm^b}\right|$. The lower bound for MD is represented as the minimum processing machine deviation across all machines for each operation s of original job j: $LB(MD) = \sum_{j\in J} \sum_{s\in O_j} \min_{m\in K_{js}} \left|m - m^b\right|$. Therefore, the lower bound for shop stability is computed as $LB(TD) + LB(MD)$. Minimizing TT involves scheduling jobs close to their due times, which can make it challenging to adhere to the original start times, increasing TD. Minimizing C_{max} typically requires optimizing the overall schedule to finish as soon as possible, which might lead to more deviations in machine assignments (higher MD). Therefore, there are significant conflicts between minimizing efficiency and stability. Efforts to optimize one objective often lead to increased values in the other. However, trade-offs exist where some level of efficiency improvement might be achieved with acceptable stability degradation, or vice versa.

The proposed mathematical model aims to balance shop efficiency and schedule stability, incorporating a rescheduling mechanism that adapts to dynamic conditions, such as new arrival jobs and machine idle times. The machine-idle-driven strategy is employed to reschedule new arrival jobs, ensuring the optimal utilization of available resources and enhancing overall performance. The MODFJSSP model combines the complexities of the JSSP with flexibility, new arrival jobs, machine idle times, and bi-objective optimization. The JSSP is NP-hard and difficult to solve (Lenstra et al., 1977). Flexibility refers to the possibility of assigning operations to any machine that can perform the required tasks, exponentially increasing the number of potential solutions. New arrival jobs and machine idle times add decision-making complexity, as they require continuous rescheduling once the rescheduling point is triggered. The bi-objectives further complicate the scheduling process by seeking a trade-off between shop efficiency and schedule stability simultaneously. Therefore, the proposed mathematical model is inherently more complex due to its multifaceted nature, which cannot be solved in polynomial time, and the complexity grows exponentially with the size of the problem instance.

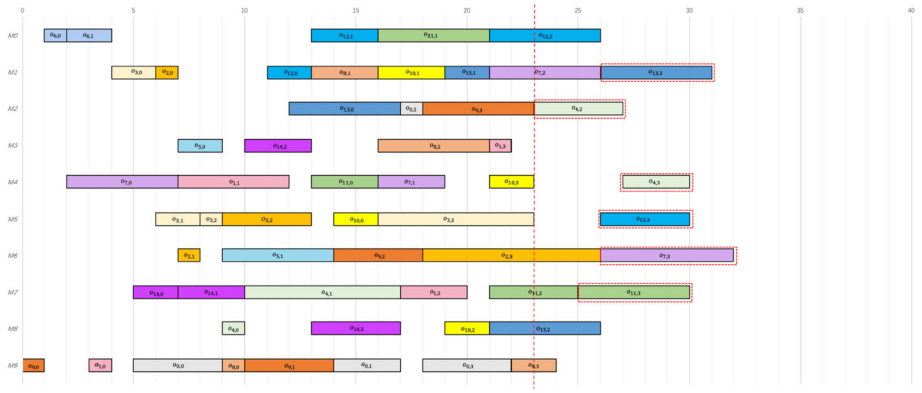

Fig. 1 Gantt chart of one of the best-found solutions for the original schedule in the illustrative example

3.3 Illustrative example

An example of the MODFFJSSP before the rescheduling is triggered is presented in Table 8 (Appendix A), encompassing a large-scale problem with 15 original jobs and 10 machines. This example extends the widely recognized FJSSP instance (I4) as described in Kacem et al., (2002b), providing a complex yet realistic scenario that enables thorough testing of various solution approaches. One of the best-found solutions for this original schedule is shown in the Gantt chart in Fig. 1. If rescheduling is triggered when there are at least 3 idle machines and certain new arrival jobs, the rescheduling point is marked by the red dashed line. Operations $o_{13,3}$, $o_{4,2}$, $o_{4,3}$,$o_{12,3}$, $o_{7,3}$, $o_{11,3}$ will be rescheduled from this red dashed line along with the new arrival jobs.

4 Solving approach

4.1 Process of the machine-idle-driven rescheduling strategy

To make full use of the machine resources, the machine-idle-driven strategy derived from the predictive–reactive rescheduling (Ouelhadj & Petrovic, 2009) is utilized to reschedule new arrival jobs, and the flowchart is presented in Fig. 2. At the initial time, operations are assigned to machines in order, following the original schedule. After all scheduled jobs have started to be processed, the production line would keep monitoring the machine utilization level and gathering new arrival jobs. Once the rescheduling is triggered, new arrival jobs need to be scheduled with the remaining operations of original jobs in a limited time while operations that are being executed are not terminated. The fully parallel NSGA-II is used to generate a set of non-dominated schedules considering both shop efficiency and schedule stability. Afterward, one schedule that fits the decision maker's preference is selected. Finally, the updated schedule is implemented until the next rescheduling point. This process continues until all jobs appearing on the shop floor have been finished.

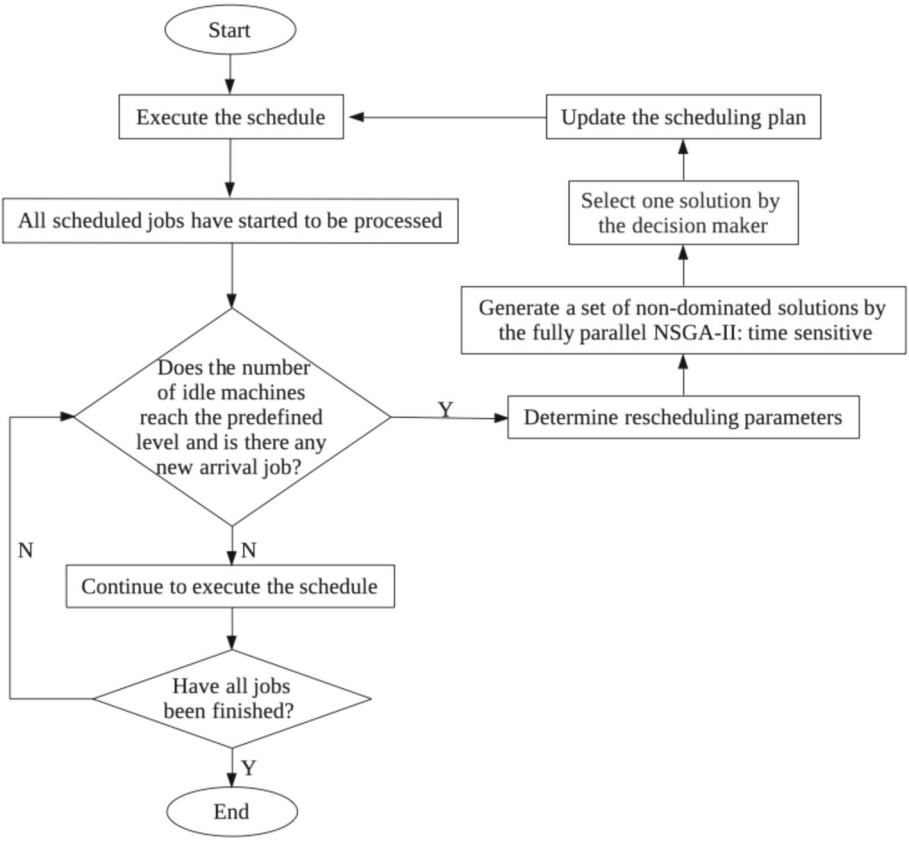

Fig. 2 Flow of the machine-idle-driven rescheduling for solving the MODFJSSP

4.2 Encoding scheme

The MODFJSSP is a combination of operation sequencing and machine assignment decisions (Zhang et al., 2018). Therefore, both the operation sequence vector and the machine assignment vector are used to represent the complete scheduling. The operation sequence vector utilizes the operation-based encoding where each job is represented by a natural number and each number is present as many times as the number of operations of the job it represents (May et al., 2015). The kth occurrence of a job number refers to the kth operation in the technological sequence of this job and the sequence of job numbers expresses the order where the operations of jobs are scheduled (Park et al., 2003). The machine assignment vector represents the assigned machine of each operation where the order is from the 0th operation of job 0 to the $(o_{n+n'-1}-1)$th operation of job $n+n'-1$. Moreover, the length of the two vectors is equal to $\sum_{j=0}^{n+n'-1} o_j$, including both original jobs and new arrival jobs. Operations whose starting times are earlier than the rescheduling point in the original schedule are treated as completed ones and will not be interrupted by the rescheduling. Their job numbers and machine numbers are taken directly from the original schedule where job numbers are placed at the beginning of the sequence and machine numbers are located at their original positions.

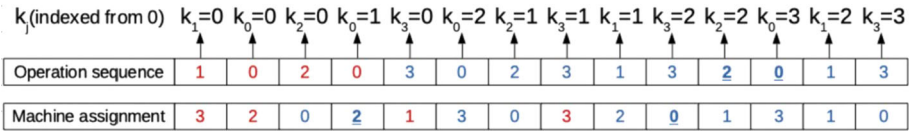

Fig. 3 Example of the two-vector representation

The remaining values are randomly initialized. An example is shown in Fig. 3 where job 0, job 1, job 2 are original jobs and job 3 is a new arrival job. Operation 2 of job 2 is scheduled on machine 0 before operation 3 of job 0 is scheduled on machine 2. Besides, the numbers in red represent the completed operations and machines used to process them while the rest in blue display operations and machines waiting to be dealt with after the rescheduling point. More details about the decoding procedure are discussed in Shen and Yao (2015).

4.3 NSGA-II and its full parallelization procedure

The NSGA-II (Deb et al., 2002) uses the ranking and crowding mechanisms to generate the approximated Pareto fronts. Each individual in the population has two attributes: nondomination rank and crowding distance. If two individuals have different nondomination ranks, the individual with the lower rank is preferred while if two individuals' ranks are equal, the one with higher crowding distance is better. To carry out the above-mentioned mechanisms, two operators are designed: fast non-dominated sorting and crowding distance computation. The fast non-dominated sorting operator assigns solutions to different Pareto fronts by a specific data structure that saves a domination count variable N_p and a set of dominated solutions S_p where p represents a specific individual in NSGA-II. After calculating N_p and S_p for every individual p, individuals whose $N_p = 0$ are assigned to the first nondominated front and their ranks are set as 1. For individuals in the first nondominated front, N_p of each member q in S_p is reduced by one. If N_p of any member q becomes zero, it is put in a separated list Q. All members in Q are assigned to the second non-dominated front and their ranks are set as 2. This procedure continues with individuals assigned to the last non-dominated front until all ranks are identified. On the other hand, the crowding distance computation operator first sorts individuals in the same non-dominated front according to each objective function value in ascending order. Afterward, the boundary individuals for each objective function are set to be an infinite value while other intermediates are assigned a value equal to the absolute normalized difference in the objective function values of two adjacent individuals. After conducting this calculation with all objective functions, the crowding distance value is calculated as the sum of these values corresponding to each objective.

The procedure of the fully parallel NSGA-II is expressed in Fig. 4 where g represents the current generation number, P_g denotes the parent population (a set of chromosomes at generation g), and Q_g displays the offspring population (a set of chromosomes obtained by NSGA-II operators at generation g). The communication between CPU and GPUs is minimized by running all NSGA-II components on GPUs. The parallel NSGA-II could always obtain a set of feasible solutions once the termination condition is reached and send it back to the CPU once at the end of the application. In MODFJSSP, the termination criterion is generally set as the timeframe given by the decision maker where the parallel design enables

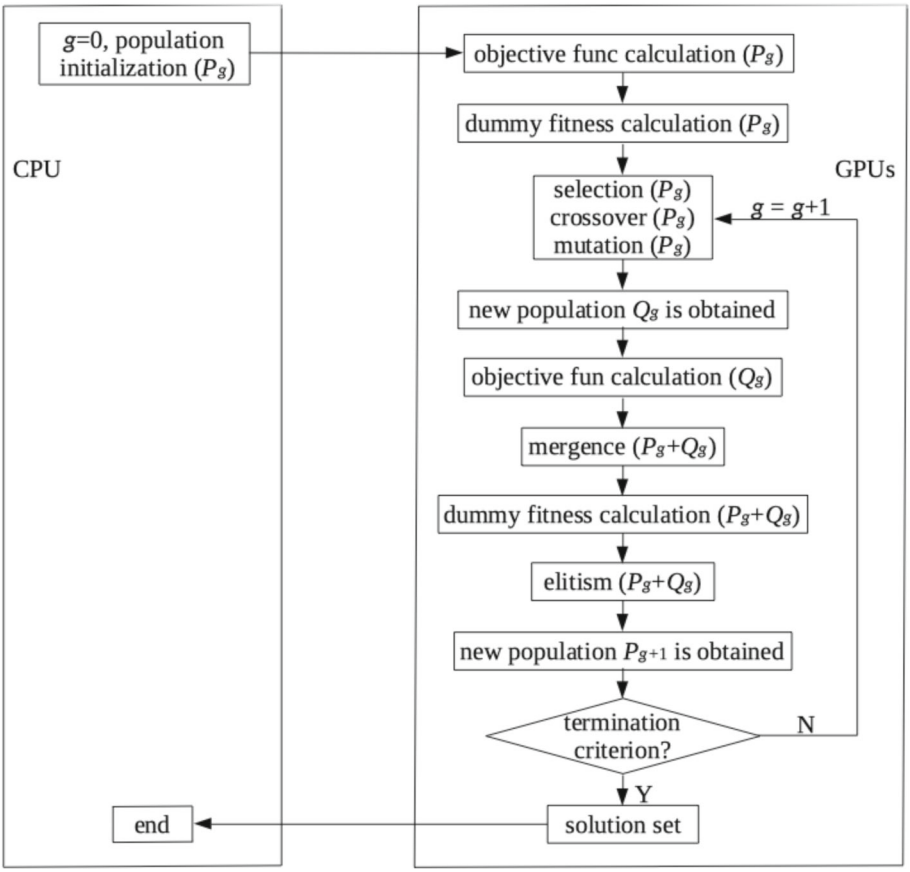

Fig. 4 Procedure of the fully parallel NSGA-II

the algorithm to be run with more generations and furthermore to have a larger chance to obtain better solutions. The dummy fitness function is built based on the fast non-dominated sorting and the crowding distance computation by combining two factors into one, which facilitates the elitist preservation strategy designed in Deb et al. (2002) to be executed in parallel.

5 Full parallelization model

For an easy presentation, we summarize the notations used in the full parallelization model in Table 2.

Table 2 Notations used in the full parallelization model

Notation	Description
ps, size	Number of individuals
*#	Ordinary variable that holds the value of its type
*	Pointer variable that holds the memory address of a vector
POP	Individual vector where each element consists of eight members
RANK	Non-domination rank matrix used to determine which Pareto front an individual belongs to where each element consists of four members
CROWD	Crowding distance matrix where each element consists of two members
*d_POP, *pop*	A pointer that references the *POP* data structure in device memory, referred to as *d_POP* in the launch parameters and as **pop* in the kernel functions. They are used to construct the vector *POP* on GPUs
*d_RANK, **rank*	A pointer that references an array of pointers to the *RANK* data structure in device memory, referred to as *d_RANK* in the launch parameters and as ***rank* in the kernel functions. They are used to construct the matrix *RANK* on GPUs
*d_CROWD, **crowd*	A pointer that references an array of pointers to the *CROWD* data structure in device memory, referred to as *d_CROWD* in the launch parameters and as ***crowd* in the kernel functions. They are used to construct the matrix *CROWD* on GPUs
***RANK*	The declaration version of the pointer that references an array of pointers to the *RANK* data structure in host memory
***d_RANK*	The declaration version of the pointer that references an array of pointers to the *RANK* data structure in device memory, specifically, the declaration forms of *d_RANK* and ***rank*
*h_R, **h_RANK,*d_R*	Buffer pointers used to construct the *RANK* data structure in device memory
om	Other members in the data structure except **sp*
threadIdx.x, blockIdx.x and blockDim.x	CUDA predefined variables
offset	Unique CUDA thread ID, calculated as *threadIdx.x* + *blockIdx.x * blockDim.x*
funcno	Objective function number
maxgen	Max generation of NSGA-II
max, e_max	Estimated maximum value of the k^{th} objective function
min, e_min	Estimated minimum value of the k^{th} objective function
base, b	Variable used to convert the crowding distance as an indicator where the smaller value is preferable

5.1 Data structures

Due to the importance of the memory hierarchy in NSGA-II, three data structures, *POP*, *RANK* and *CROWD*, are designed. *POP* is a $ps \times 1$ individual vector where each element consists of eight members: #chrom, #objvalue, #dummy (dummy fitness value), #index, #flag, #np (N_p), #dp, *sp. #chrom is a $\sum_{j=0}^{n+n'-1} o_j \times 2$ matrix storing a two-vector representation

chromosome and #objvalue is a 2×1 vector saving objective function values. #index is used to share information among *POP*, *RANK* and *CROWD* while #flag is an indicator working with #np to assign individuals to the next non-dominated front. *sp is a pointer to store the memory address of a $ps \times 1$ vector for S_p. The vector used to save S_p is sparse because the number of elements in S_p is unknown when the NSGA-II starts where #dp works as a counter for elements in S_p. *RANK* is a $ps \times ps$ non-domination matrix where each element consists of four members: #value, #index, #dp, *sp. *CROWD* is a $ps \times ps$ crowding distance matrix where each element consists of two members: #value, #index. Each row of *RANK* or *CROWD* could be considered as the separated list Q where information about individuals that are in row k (indexed from 0) present individuals in the k + 1th non-dominated front. Since the number of non-dominated fronts and the number of individuals in each non-dominated front are also unknown when the NSGA-II starts, *RANK* and *CROWD* are sparse as well. *POP*, *RANK* and *CROWD* are defined as static data structures. Static allocations have the drawback of oversizing *RANK*, *CROWD* and the vector used to save S_p to guarantee enough memory due to the unknown dominance patterns. However, the memory requirements can be greatly reduced if the number of fronts is a priori known (Ortega et al., 2017).

The GPU-based fully parallel NSGA-II is developed using the CUDA programming model where both the CPU and GPUs are used. In CUDA, the host refers to the CPU while the device refers to the GPUs (Harris, 2012). The host and device in CUDA have separate memory spaces and device pointers cannot be referenced in the host code. Therefore, three buffer pointers are designed to read from or write to the data structure *RANK* between the host and the device, where **h_R* and ***h_RANK* are host pointers, and **d_R* is a device pointer, as illustrated in Fig. 5. The solid arrows display the copy of addresses, where each arrow in blue or in orange points to the same $1 \times ps$ array on the device, each arrow in green or in red points to the same $ps \times 1$ array on the device. After non-dominated sorting, the values (without pointers) referenced in ***d_RANK* are copied to the values (without pointers) referenced in ***RANK*, as indicated by the pink hollow arrows. Moreover, the data structures of *POP* and *CROWD* on CPU and GPUs are designed in a similar way, which are omitted here due to space limitations.

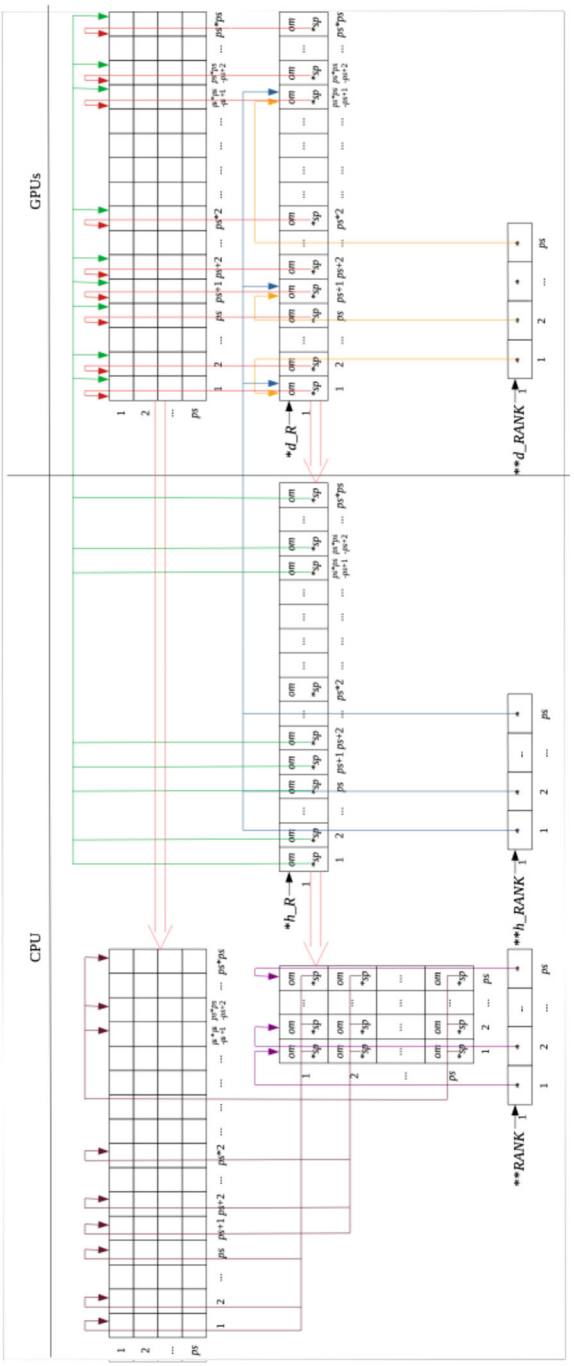

Fig. 5 Data structure of RANK and copying of addresses and values between host and device

5.2 Dummy fitness calculation

In CUDA, a function of code on the device is called as a kernel. It is launched by the host and executed by an array of threads in parallel. All threads run the same code and are grouped into blocks, which are equal cardinality subsets of threads while blocks are in turn logically arranged into a grid, which is a 1-, 2-, or 3-dimensional array of blocks (Boschetti et al., 2016). The implementation of the fully parallel NSGA-II starts with the host code as shown in Algorithm 1. Eight kernels (lines 4–15) are launched to obtain the dummy fitness value by the non-dominated sorting (lines 5–8) and the crowd distance computation (lines 9–13). The parameters within < < < > > > are the execution configuration, which specifies the number of blocks and the number of threads in one block that are used to execute the kernel. Each thread is designed to process one individual and identified by the unique global id as threadIdx.x + blockIdx.x * blockDim.x where threadIdx.x, blockIdx.x and blockDim.x are CUDA predefined variables.

After initialization, kernel fast_nondom_first_front (Algorithm 2) is launched to compute the first non-dominated front while N_p and S_p are saved. As the next step, kernel fast_nondom_other_front (Algorithm 3) is launched iteratively to identify all other fronts while redundant kernel launch may occur if the total number of non-dominated fronts is less than *ps*. Moreover, the reduction in Algorithm 3 is executed by the atomic operation (line 4) while the correctness of the parallel NSGA-II is guaranteed. These two kernels are used to compute elements stored in *RANK* on GPUs where elements in the same row are processed in parallel. Instead of sorting individuals in the same front according to each objective function value in ascending order, the device code sorts them in each *RANK* row on GPUs in decreasing order by the Bitonic Merge Sorting (Pharr & Fernando, 2005). As #value of *RANK* on GPUs is initialized as -1, all valid elements are saved at the left side of *RANK* on GPUs and the crowding distance is obtained by iteratively launching kernel crowd_distance (Algorithm 4). In order to have an overall merit for the non-dominated front and the crowding distance, kernel dummy_fitness (Algorithm 5) is launched at the end to calculate the dummy fitness value. After sorting elements in each *CROWD* row on GPUs by their crowding distance in decreasing order using the Bitonic Merge Sorting, the variable '*base*' is imported in Algorithm 5. It is computed as 10^k when *ps* is any value within the interval of all k-digits numbers and used to convert the crowding distance as an indicator where the smaller value is preferable (line 5). Finally, the fitness value of each individual is obtained as the sum of the non-dominated front and the converted crowding distance (line 6) where any individual with the smaller dummy fitness value has the priority to be saved by the NSGA-II operators. Elements stored in the same column are processed in parallel in these two kernels. Moreover, details about NSGA-II operators in the while loop are discussed in Sect. 5.3.

Algorithm 1: Host code of the fully parallel NSGA-II

```
1) allocate memory on the host and the device
2) initialize chromosome of each individual on the host
3) copy addresses and data from the host to the device
4) launch kernel to initialize POP, RANK and CROWD on the device where values stored in #objvalue
   of POP are calculated through values stored in #chrom of POP; #index of POP is initialized from 0
   to ps-1 in ascending order; #dummy, #flag, #np and #dp of POP are initialized to be 0; values in the
   vector, which is pointed by *sp of POP, are initialized to be -1; #value and #index of RANK are
   initialized to be -1; #dp of RANK is initialized to be 0; values in the vector, which is pointed by *sp
   of RANK, are initialized to be -1; #value and #index of CROWD are initialized to be -1
5) launch kernel fast_nondom_first _front<< < blocks, threads >> >( d_POP, d_RANK, ps )
6) for k ← 0 to ps-1 do
7)    launch kernel fast_nondom_other_front << < blocks, threads >> >( d_POP, d_RANK, k )
8) end for
9) for k ← 0 to funcno-1 do
10)   launch kernel to assign the k^th #objvalue of POP to #value of RANK by #index on the device
11)   launch kernel to sort RANK in each row by #value in decreasing order on the device
12)   launch kernel crowd_distance << < blocks, threads >> >(d_RANK, d_CROWD, k, ps, max, min)
13) end for
14) launch kernel to sort CROWD in each row by #value in decreasing order on the device
15) launch kernel dummy_fitness<< < blocks, threads >> >( d_RANK, d_CROWD, base, ps)
16) while g≤maxgen do
17)   launch kernels to execute NSGA-II operators on the device
18) end while
19) copy data from the device to the host
20) deallocate memory on the host and the device
```

Algorithm 2: Device code for identifying the first non-dominated front

```
1)  procedure fast_nondom_first _front (*pop, **rank, size)
2)    offset ← threadIdx.x + blockIdx.x * blockDim.x
3)    for k ← 0 to size-1 do
4)      if offset≠ k then
5)        dom ← check_dominance(pop[offset],pop[k])
6)        if dom = 1 then
7)          pop[offset].np ← pop[offset].np+1
8)        else
9)          pop[offset].sp[pop[offset].dp++] ← k
10)       end if
11)     end if
12)   end for
13)   synchronization
14)   if pop[offset].np=0 then
15)     pop[offset].flag ← 1
16)     rank[0][offset] ← pop[offset]
17)   end if
18) end procedure
```

Algorithm 3: Device code for identifying the remaining non-dominated fronts

```
1)  procedure fast_nondom_other _front (*pop, **rank, rank)
2)    offset ← threadIdx.x + blockIdx.x * blockDim.x
3)    for k ← 0 to rank[rank][offset].dp do
4)      pop[rank[rank][offset].sp[k]].np ← atomicSub( &pop[rank[rank][offset].sp[k]].np, 1 )
5)    end for
6)    synchronization
7)    if pop[offset].np=0 and pop[offset].flag=0 then
8)      pop[offset].flag ← 1
9)      rank[rank+1][offset] ← pop[offset]
10)   end if
11) end procedure
```

Algorithm 4: Device code for computing the crowding distance

```
1)  procedure crowd_distance (**rank, **crowd, func, size, e_max, e_min)
2)    offset ← threadIdx.x + blockIdx.x * blockDim.x
3)    for k ← 0 to size-1 do
4)      if rank[offset][k].index < 0 then
5)        tempindex ← k
6)        break
7)      end if
8)    end for
9)    for k ← 0 to tempindex-1 do
10)     if k = 0 and k = tempindex-1then
11)       rank[offset][k].value ←  ∞
12)     else
13)       rank[offset][k].value ← (rank[k−1].value−rank[offset][k+1].value) / (e_max−e_min)
14)     end if
15)     if func = 0 then
16)       crowd[offset][rank[offset][k].index].index ← rank[offset][k].index
17)       crowd[offset][rank[offset][k].index].value ← 0
18)     endif
19)     crowd[offset][rank[offset][k].index].value←crowd[offset][rank[offset][k].index].value + rank[offset][k].value
20)   end for
21) end procedure
```

Algorithm 5: Device code for computing the dummy fitness

```
1)  procedure dummy_fitness (*pop, **crowd, b, size)
2)    offset ← threadIdx.x + blockIdx.x * blockDim.x
3)    for k ← 0 to size-1 do
4)      if crowd[offset][k].index>=0 then
5)        crowd[offset][k].value ← (float)k/b
6)        pop[crowd[offset][k].index].dummy ← crowd[offset][k].value +offset+1
7)      end if
8)    end for
9)  end procedure
```

Before crossover	Parent 1	Operation sequence	= [1, 0, 2, 0, 3, 0, 2, 3, 1, 3, 2, 0, 1, 3]
		Machine assignment	= [3, 2, 0, 2, 1, 3, 0, 3, 2, 0, 1, 3, 1, 0]
	Parent 2	Operation sequence	= [1, 0, 2, 0, 2, 1, 3, 0, 3, 0, 2, 1, 3, 3]
		Machine assignment	= [3, 2, 1, 1, 1, 2, 3, 3, 2, 2, 0, 3, 0, 3]
		↓	
After crossover	Offspring 1	Operation sequence	= [1, 0, 2, 0, 1, 0, 2, 3, 3, 2, 1, 0, 3, 3]
		Machine assignment	= [3, 2, 0, 2, 1, 3, 3, 3, 2, 2, 0, 3, 0, 3]
	Offspring 2	Operation sequence	= [1, 0, 2, 0, 2, 3, 1, 0, 3, 0, 3, 2, 1, 3]
		Machine assignment	= [3, 2, 1, 1, 1, 2, 0, 3, 2, 0, 1, 3, 1, 0]

Fig. 6 Example of the crossover operation

5.3 NSGA-II operations on GPUs

5.3.1 Selection operator and elitist preservation strategy

With the data structures and the dummy fitness calculation presented in the previous subsections, the fully parallel NSGA-II can be considered as a verbatim port of the conventional NSGA-II. Therefore, operators of the conventional NSGA-II can be kept intact. At the beginning of each generation, the usual binary tournament selection, crossover and mutation operators are used to create an offspring population Q_g where the individual with the lowest dummy fitness value is selected for crossover and mutation. After merging the parent population P_g with the offspring population Q_g, all individuals are sorted by their dummy fitness values in ascending order using the Bitonic-Merge sort while the top ps individuals are saved as an elite group for the next generation.

5.3.2 Crossover operator

Firstly, the Fisher-Yates shuffle (Capodieci & Burgio, 2015) is utilized to pair two individuals on GPUs randomly. Afterward, the operation-based order crossover (Luo et al., 2020) is used for the operation sequence vector and works for genes representing uncompleted operations. On the other hand, the traditional single-point crossover is applied to the machine assignment vector. The full procedure for an example is shown in Fig. 6 where job 0, job 1, job 2 are original jobs and job 3 is a new arrival job. The integers highlighted in yellow indicate completed operations and machines used to process them while the integers in red mark the loci of randomly chosen operations in the operation sequence vector and the single crossover point in the machine assignment vector. Since the vectors are not related to each other, the two crossover operators are implemented independently.

5.3.3 Mutation operator

The mutation operation only executes with genes representing uncompleted operations and machines used to process them. The swap mutation is applied for the operation sequence vector where different arbitrary genes are chosen and exchanged values. Concerning the machine assignment vector, $n + n'$ genes are substituted by randomly generated values within the range, aside from the original ones. Following the above example, this procedure is illustrated in Fig. 7, where genes in green display the execution of the mutation operation.

Before mutation	Operation sequence	= [1, 0, 2, 0, 1, 0, 2, 3, 3, 2, 1, 0, 3, 3]
	Machine assignment	= [3, 2, 0, 2, 1, 3, 3, 3, 2, 2, 0, 3, 0, 3]
	↓	
After mutation	Operation sequence	= [1, 0, 2, 0, 1, 2, 2, 3, 3, 0, 1, 0, 3, 3]
	Machine assignment	= [3, 2, 2, 1, 1, 3, 3, 3, 2, 0, 0, 1, 0, 3]

Fig. 7 Example of the mutation operation

5.4 Insights on full parallelization

It is obvious that the fully parallel NSGA-II keeps the original structure of the conventional NSGA-II. The time complexity of the fast non-dominated sorting in sequential is $O(MN^2)$. However, if the number of GPU threads is larger than the number of individuals, the actual time complexity is reduced to $O(MN)$ as the domination check executed in the first front can be implemented in parallel. In exchange, the storage requirement is increased to $O(MN^3)$ as individuals on all fronts have to keep a local copy of S_p. The complexity of the crowding distance is governed by the sorting algorithm as it requires to sort the population according to each objective function value. Instead of the sorting algorithm used in the sequential NSGA-II whose computational complexity is $O(MNlogN)$, the Bitonic-Merge sort is utilized in the parallel version. Although the Bitonic-Merge sort suffers from $O(MNlog^2N)$ (Pharr & Fernando, 2005), the actual time complexity can be decreased to $O(Mlog^2N)$ under the condition that the number of GPU threads is larger than the number of individuals. Therefore, the computational burden of the fully parallel NSGA-II is the fast non-dominated sorting. If we have enough GPU threads, the overall time complexity is reduced from $O(MN^2)$ to $O(MN)$.

6 Computational experiments

Test 1 checks the efficiency and the effectiveness of the fully parallel NSGA-II on GPUs by solving nine MODFJSSP instances while test 2 evaluates the performance of MODFJSSP by a case study. The GPU code implementation is carried out using CUDA 9.2 on NVIDIA TITAN X (Pascal) with 3584 single-precision CUDA cores. All programs are written in C, except for the GPU kernels in CUDA C.

6.1 Evaluation

6.1.1 Instances description

Firstly, we have extended six representative FJSSP instances (2 sets of 10-job instances, 2 sets of 15-job instances, and 2 sets of 20-job instances) (Brandimarte, 1993) by fixing dynamic factors as shown in Table 3. Second, three DFJSSP instances derived from Nguyen et al. (2018) are also considered where PF $f\%$ implies that $f\%$ of the total number of machines in the shop are available to process an operation. Finally, the release time, the due time and the number of idle machines triggering the rescheduling point are set up by simulation following the rules as defined in Table 4. Furthermore, all instance

Table 3 Benchmark problems

Source	Problem	n	n'	k	k_{js}
Brandimarte (1993)	MK01	10	6	6	3
	MK02	10	5	6	6
	MK04	15	8	8	3
	MK05	15	1	4	2
	MK07	20	6	5	5
	MK08	20	2	10	2
Nguyen et al. (2018)	PF20%	10	Till *RS*, new jobs arrive following the Poisson distribution	10	$n\times 20\% = 2$
	PF50%	10		10	$n\times 50\% = 5$
	PF100%	10		10	$n\times 100\% = 10$

Table 4 Experimental relative data

R_j	$U[0, \overline{P}]$, where $\overline{P} = \sum_{j\in J\cup J'}\left(\sum_{s\in O_j}\left(\sum_{m\in K_{js}} P_{jsm}/k_{js}\right)/o_j\right)$
D_j	$R_j + \overline{P}_j \times (1+\sigma)$, where $\sigma = \mathrm{U}[0, 2]$ and $\overline{P}_j = \sum_{s\in O_j}\left(\sum_{m\in K_{js}} P_{jsm}/k_{js}\right)$
θ	*Discrete* $U(0, T)$
$\left\|m - m^b\right\|$	1, if $m \neq m^b$

data can be downloaded from https://www.dropbox.com/scl/fo/je7yu1f0uhrt60x1hd52l/AMaEkEI2JJXvdlSP6a2LJQs?rlkey=01407hsysw67tuujd8qtzhvzt&st=t7yll789&dl=0.

6.1.2 Performance measures

The performance of multi-objective algorithms is frequently evaluated by performance metrics (Audet et al., 2021). Since the true Pareto fronts for the tested MODFJSSP instances are unknown in advance, four popular metrics (Ahmadi et al., 2016) are used to evaluate their performance. Their explanations are expressed as follows:

- Number of Pareto Solutions (NPS): Solution quality metric where the higher value implies that decision-makers have access to more alternative solutions.
- Spacing: Solution quality metric where the lower value indicates that the consistency of spacing among non-dominated solutions is higher.
- Modified Mean Ideal Distance (MMID): Solution quality metric where the lower value presents that the algorithm has better convergence performance. It is the normalized version of the Mean Ideal Distance.
- Time: Execution time metric where the lower value expresses that the algorithm's running time is shorter.

Table 5 Parameter tuning results

Crossover rate	Mutation rate	NPS	Spacing	MMID	Time
0.9000	0.0500	26.5000	5.7016	**0.6822**	219.1044
0.9000	0.1000	**26.6667**	**3.3916**	0.6918	**218.5694**
0.9000	0.1500	24.8667	5.2916	0.6996	224.7204
0.9000	0.2000	25.6667	4.1126	0.6967	225.6631
0.9000	0.2500	**26.6667**	3.9832	0.7065	225.5075

6.1.3 Parameter tuning

Considering the existing experiences, the most appropriate crossover rate ranges between 0.75 and 0.9 (Schaffer, 1989) and the mutation rate should be much lower than the crossover rate (Cabrera et al. 2002). The crossover rate and the mutation rate were set as 0.9 and 1/*funcno* respectively in NSGA-II (Deb et al., 2002) for real-coded GAs. Therefore, we would like to keep the crossover rate and the mutation rate of the fully parallel NSGA-II as 0.9 and 0.5 respectively since there are only two decision variables in MODFJSSP. However, as 0.5 is not much lower than 0.9, we further apply the fully parallel NSGA-II on MK01 with five groups of mutation rates as shown in Table 5. Concerning the average results of 30 iterations, we could find parameters do not have a great impact on the algorithm performance if they are kept within the acknowledged range. Since the proposed algorithm has achieved the best result on 3 metrics when mutation rate $= 0.1$, the crossover rate and the mutation rate are finally set as 0.9 and 0.1 respectively. For the fairness of comparison, we keep the same values for all algorithms in the following tests. Moreover, the population size is kept as 256 while the island size for the multi-objective hybrid GA is specified as 64 (Luo et al., 2019). Finally, the termination criterion is fixed as 500 generations rather than a reasonable time budget to have an easier comparison among different algorithms. The criterion was chosen after verifying no significant improvement in the Pareto front, indicating that convergence has been achieved.

6.1.4 Computational results

The proposed algorithm is compared with two GPU-based parallel Genetic Algorithms (GAs): the cellular GA and the hybrid GA. The cellular GA (Alba & Dorronsoro, 2009) maps individuals in a grid environment where two-parent individuals are selected from a smaller neighborhood area and recombined to generate a new one. Afterward, this new individual undertakes the mutation and replaces the original individual if its solution is better. The hybrid GA (Luo et al., 2019) is designed with a fine-grained GA at the lower level and an island GA at the upper level. The selection, crossover, and mutation are executed at the lower level while an elitism-based replacement after every generation inside the island and a migration among islands after every ten generations are carried out at the upper level. These two GPU-based parallel GAs cannot be used directly for seeking the Pareto front. Therefore, the data structures designed in Sect. 5.1 and the dummy fitness defined in Sect. 5.2 are implemented while the rest is kept intact.

We examine the three GPU-based parallel algorithms for 150 independent runs. The results are shown in Table 6 where each row represents an instance and each column indicates the

Table 6 Efficiency and effectiveness comparison

	Fully parallel NSGA-II				Multi-objective cellular GA				Multi-objective hybrid GA			
	NPS	Spacing	MMID	Time	NPS	Spacing	MMID	Time	NPS	Spacing	MMID	Time
MK01	**24.4000**	**4.4718**	**0.6824**	**212.0428**	19.5800	7.5985	0.7068	213.8040	16.0067	7.2141	0.7453	291.6103
MK02	**38.1600**	**2.9634**	**0.6766**	**219.7955**	23.7267	6.3053	0.7022	226.2363	20.4733	5.3802	0.7441	313.4725
MK04	**14.4400**	**8.6656**	**0.7005**	208.0219	13.2400	10.7433	0.7213	**203.9253**	12.0867	10.1062	0.7549	284.9302
MK05	**109.4600**	**5.6546**	**0.6636**	**286.4334**	68.8600	8.6955	0.6810	289.8694	32.1467	15.1342	0.7641	390.6846
MK07	**9.5800**	**17.6725**	0.7399	**247.7202**	8.1933	26.9988	0.7366	251.1218	8.9267	19.1916	**0.7173**	346.9567
MK08	**17.4400**	**16.5119**	**0.6720**	202.7941	17.0600	18.7785	0.6723	**180.7788**	15.1600	19.7538	0.6850	273.3985
PF20%	**11.9800**	**9.1752**	**0.6883**	175.4895	10.7267	11.3644	0.6965	**170.6742**	10.6400	9.8904	0.7241	220.9693
PF50%	**18.0533**	**4.8293**	**0.6952**	191.6625	13.5867	8.8425	0.7375	**182.7641**	14.2800	7.3816	0.7419	233.3737
PF100%	**14.2667**	**8.6452**	**0.7011**	**190.1555**	13.5133	11.9692	0.7056	195.9963	12.5934	8.6945	0.7029	238.6980

average value for one metric. The best values are highlighted in bold. Overall, the proposed algorithm achieves the best performance in most instances from all metrics. Concerning the solution quality, the fully parallel NSGA-II obtains the best results on NPS and Spacing. It also overcomes the other two algorithms on MMID for most instances while the difference between the proposed method and the multi-objective hybrid GA in instance MK07 is insignificant. Consequently, it is important to keep the original structure of NSGA-II. Although other parallel evolutionary algorithms can be modified to solve multi-objective problems, it is hard for them to keep the solution quality as good as the NSGA-II. In terms of execution time, the fully parallel NSGA-II works most efficiently in 5 instances out of 9. Since the cellular model is designed particularly for the 2D environment (Dorronsoro & Bouvry, 2013), the execution time of multi-objective cellular GA is not so different from the proposed method and even gets the lowest value for instances MK4, MK8, PF20% and PF50%. The multi-objective hybrid GA needs a longer execution time than the other two algorithms as it has to implement more times the ranking and crowding mechanisms for carrying out the migration.

Moreover, the Wilcoxon signed ranks test (Derrac et al., 2011) is utilized to investigate to what extent the fully parallel NSGA-II differs from the other two algorithms on overall performance. The significance level is set as 0.1 while the R−, R+ and p values are computed by SPSS (https://www.ibm.com/analytics/spss-statistics-software) and displayed in Table 7. We can find that the proposed method overcomes the multi-objective cellular GA on NPS for instances MK01, MK02, MK05, MK7, PF20%, PF50% on Spacing for all instances, on MMID for instances MK1, MK2, MK4, MK5, PF50%, on Time for instances MK01, MK2, MK5, MK7, PF100%. Meanwhile, it shows an improvement over the multi-objective hybrid GA on NPS for instances MK01, MK02, MK04, MK5, MK8, PF20%, PF50%, PF100%, on Spacing for all instances, on MMID for instances MK01, MK02, MK04, MK05, MK08, PF20%, PF50%, on Time for all instances.

6.2 Case study

6.2.1 Case data

The illustrative example in Sect. 3.3 is expanded to include dynamic factors as a case study. In addition to the original fifteen jobs, five new arrival jobs are introduced. The processing times for each operation of these new jobs on the available machines are generated using a discrete uniform distribution over the range [0, 10]. Their release and due times are determined according to the rules specified in Table 4. Further details can be found in Table 9 (Appendix A).

The original schedule is established as one of the best-found solutions shown in Fig. 1, where rescheduling is triggered when there are at least three idle machines after all scheduled jobs have begun processing. The machine assigned to operation $o_{j,s}$ of original job j and the start time of operation $o_{j,s}$ on that machine prior to the rescheduling point are fixed, as illustrated in the Gantt chart. Operations $o_{13,3}$, $o_{4,2}$, $o_{4,3}$,$o_{12,3}$, $o_{7,3}$, $o_{11,3}$ will be rescheduled from the rescheduling point, along with the five new arrival jobs, with a fixed time cost of 1 for transferring them to a different machine.

Table 7 Wilcoxon signed-rank test results

		NPS			Spacing			MMID			Time		
		R^-	R^+	p value	R^-	R^+	p value	R^-	R^+	p value	R^-	R^+	p value
Fully parallel NSGA-II vs. multi-objective cellular GA	MK01	7353.5000	3231.5000	0.0000	3706.0000	7619.0000	0.0000	44,441.0000	6884.0000	0.0220	3180.0000	8145.0000	0.0000
	MK02	8866.5000	1718.5000	0.0000	2240.0000	9085.0000	0.0000	4078.0000	7247.0000	0.0030	763.0000	10,562.0000	0.0000
	MK04	5718.5000	4434.5000	> 0.1000	4638.0000	6687.0000	0.0550	4753.0000	6572.0000	0.0880	9933.0000	1392.0000	0.0000
	MK05	10,592.0000	583.0000	0.0000	1999.0000	9326.0000	0.0000	4178.0000	7147.0000	0.0050	3970.0000	7355.0000	0.0010
	MK07	6102.0000	3351.0000	0.0030	4108.0000	7217.0000	0.0040	5745.0000	5580.0000	> 0.1000	2573.0000	8752.0000	0.0000
	MK08	5619.0000	4821.0000	> 0.1000	4623.0000	6702.0000	0.0510	5491.0000	5834.0000	> 0.1000	11,325.0000	0.0000	0.0000
	PF20%	5931.0000	3660.0000	0.0160	4390.0000	6935.0000	0.0170	5346.0000	5979.0000	> 0.1000	9234.0000	2091.0000	0.0000
	PF50%	6966.0000	3045.0000	0.0000	2913.0000	8412.0000	0.0000	3816.0000	7509.0000	0.0010	9841.0000	1484.0000	0.0000
	PF100%	5842.5000	5183.5000	> 0.1000	4038.0000	7287.0000	0.0020	5252.0000	6073.0000	> 0.1000	2033.0000	9292.0000	0.0000
Fully parallel NSGA-II vs. multi-objective hybrid GA	MK01	9779.5000	1098.5000	0.0000	2969.0000	8356.0000	0.0000	2453.0000	8872.0000	0.0000	0.0000	11,325.0000	0.0000
	MK02	10,440.5000	734.5000	0.0000	1299.0000	10,026.0000	0.0000	1565.0000	9760.0000	0.0000	0.0000	11,325.0000	0.0000
	MK04	6630.0000	3523.0000	0.0020	4651.0000	6674.0000	0.0580	3633.0000	7692.0000	0.0000	0.0000	11,325.0000	0.0000
	MK05	11,175.0000	0.0000	0.0000	572.0000	10,753.0000	0.0000	303.0000	11,022.0000	0.0000	0.0000	11,325.0000	0.0000
	MK07	5048.5000	3729.5000	> 0.1000	4691.0000	6634.0000	0.0680	6556.0000	4769.0000	0.0940	0.0000	11,325.0000	0.0000
	MK08	7313.0000	3127.0000	0.0000	4094.0000	7231.0000	0.0030	4695.0000	6630.0000	0.0690	0.0000	11,325.0000	0.0000
	PF20%	6305.0000	3848.0000	0.0120	4643.0000	6682.0000	0.0560	3767.0000	7558.0000	0.0000	145.0000	11,180.0000	0.0000
	PF50%	6940.5000	3499.5000	0.0010	3074.0000	8251.0000	0.0000	3096.0000	8229.0000	0.0000	0.0000	11,325.0000	0.0000
	PF100%	6222.0000	3931.0000	0.0200	4757.0000	6568.0000	0.0890	5379.0000	5946.0000	> 0.1000	0.0000	11,325.0000	0.0000

R^-: value of the metric got by the fully parallel NSGA-II > value of the metric got by the multi-objective cellular GA (multi-objective hybrid GA)

R^+: value of the metric got by the fully parallel NSGA-II < value of the metric got by the multi-objective cellular GA (multi-objective hybrid GA)

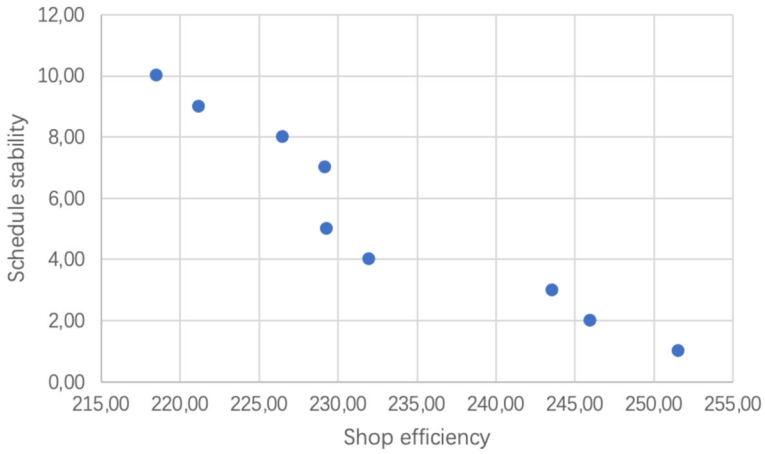

Fig. 8 Best-found Pareto solutions of the case study problem generated by the fully parallel NSGA-II on GPUs

6.2.2 Managerial insights

The Pareto solutions for the case study problem are displayed in Fig. 8. It is the best-found result of 150 independent runs of the fully parallel NSGA-II on GPUs while all parameters are kept the same as in Sect. 6.1.3. We can easily observe the inverse relationship between schedule stability and shop efficiency. As shop efficiency increases, schedule stability decreases, indicating a trade-off between the two metrics. For instance, the highest shop efficiency of 251.54 is associated with the lowest schedule stability of 1.00, while the lowest shop efficiency of 218.48 corresponds to the highest stability of 10.00. This trend suggests that efforts to boost efficiency may compromise stability, resulting in potential disruptions in operations. The diminishing returns evident in the increments of both metrics underscore the need for strategic decision-making. Decision-makers should strive to achieve a balance that optimizes performance without sacrificing operational stability, utilizing continuous monitoring to effectively adjust strategies and sustain this essential equilibrium.

The above-mentioned trade-off can be verified with more detail by the Gantt charts presented in Figs. 9 and 10. Figure 9 shows the solution located at the bottom-right of the Pareto front in Fig. 8, while Fig. 10 illustrates the solution at the top-left. Compared to the original schedule in Fig. 1, only $o_{13,3}$ is moved from M_1 to M_0 in Fig. 9. Although its starting time is kept as original, it takes more processing time in the update schedule. Most new arrival jobs are scheduled after completing operations of the original schedule on each machine. However, $o_{19,0}$, $o_{19,1}$ are two exceptions as their processing time on M_5 are as short as the idle interval between $o_{3,3}$ and $o_{12,3}$. On the opposite, many operations of new arrival jobs are scheduled before the completion of operations of original jobs in Fig. 10. Although the values of total tardiness and makespan are decreased in this case, certain operations of original jobs must change processing machines or/and starting time while only $o_{4,2}$ and $o_{4,3}$ are kept as in the original schedule.

The updated schedule in Fig. 9 minimizes disruption and ensures smoother operations, which can be particularly beneficial in environments where consistency and predictability are crucial. Decision-makers should consider this strategy when stability is a priority, as it

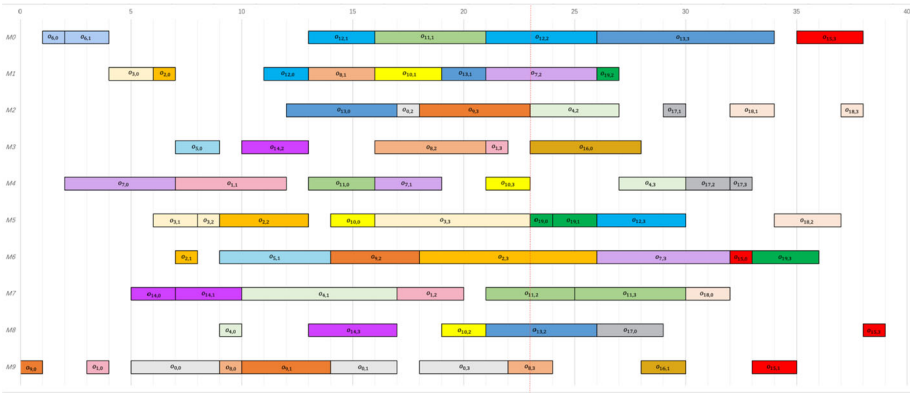

Fig. 9 Gantt chart of the solution from the Pareto front of the case study problem with the minimum value of schedule stability

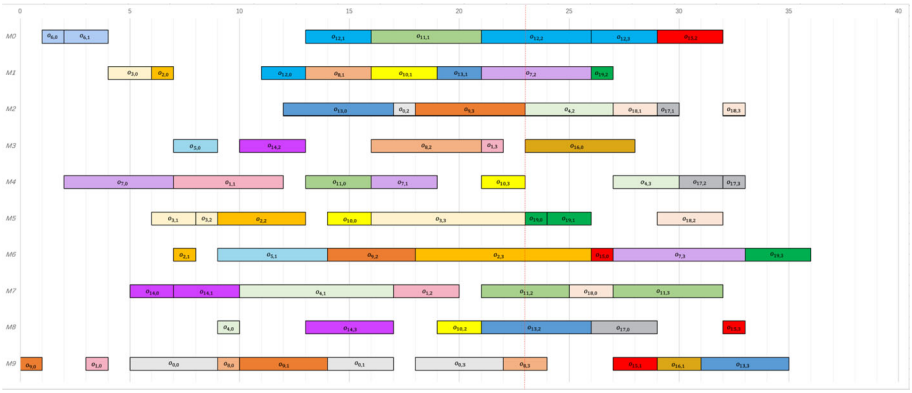

Fig. 10 Gantt chart of the solution from the Pareto front of the case study problem with the minimum value of shop efficiency

reduces the risk of operational bottlenecks. Conversely, Fig. 10 shows a more efficiency-driven approach, leading to reduced total tardiness and makespan. This strategy is more appropriate in highly competitive environments where meeting deadlines and maximizing throughput are critical, even if it introduces instability. In addition to these two extreme examples, decision-makers are encouraged to explore other potential solutions along the Pareto front presented in Fig. 8 that align with their specific preferences and operational priorities. The balance between stability and efficiency is crucial for making informed, context-driven decisions. This approach allows decision-makers to achieve an optimal balance, tailored to the unique demands and constraints of their organization.

7 Conclusions and future works

In this paper, we have first studied a multi-objective dynamic flexible job shop scheduling model. The machine-idle-driven strategy was used to reschedule new arrival jobs while both shop efficiency and schedule stability are considered. To reach a quick response in the dynamic scenario, a GPU-based fully parallel NSGA-II was proposed with respect to the original structure to keep the solutions' quality. Three data structures, *POP*, *RANK* and *CROWD* were designed particularly to make the memory hierarchy in NSGA-II compatible with the CUDA programming model. With the dummy fitness calculation, the proposed algorithm could be executed entirely on GPUs with minimal data exchange between the host and the device. The efficiency and effectiveness of our approach were verified by comparison with the other two GPU-based parallel methods. Numerical experiments showed the fully parallel NSGA-II gained better performance than the multi-objective cellular GA and the multi-objective hybrid GA in most cases from four metrics: NPS, Spacing, MMID and Time. Finally, a large-size multi-objective dynamic flexible job shop scheduling instance was simulated. It displayed the conflicting relationship between shop efficiency and schedule stability. Therefore, decision-makers were suggested to consider all optional solutions on the Pareto front on their preferences.

In the future, the internal relationship between schedule stability components will be analyzed first. Although various weights for makespan and total tardiness have been studied, a few works have discussed the weights for total starting time deviation and total processing machine deviation. Second, experiments with data from real-world implementations are planned to be carried out. Since most experiments in this domain are conducted with simulation data, real data might bring more interesting managerial insights. Third, the data structures designed to enable all NSGA-II components to be run on GPUs will be improved. As the sparse structures cannot make optimal usage of the underlying hardware, techniques to convert them into dense structures deserve further discussion. Fourth, FPGA-based approaches will be considered. Owing to the inherent parallelism of FPGAs, FPGA-based approaches and some codesign approaches are also expected to have good performance. Finally, we would also like to study parallel designs for other multi-objective evolutionary algorithms. It is hard to execute NSGA-II completely on GPUs due to its ranking and crowding mechanisms. Some algorithms, like MOEA/D Zhang and Li (2007), working with different fitness assignments and diversity preservation strategies may be more suitable for GPU computing.

Appendix A: Case data

See Tables 8 and 9.

Table 8 The case data of the original jobs

J_j	$o_{j,s}$	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	R_j	D_j
J_0	$o_{0,0}$	1	4	6	9	3	5	2	8	9	4	5	29.63
	$o_{0,1}$	1	1	3	4	8	10	4	11	4	3		
	$o_{0,2}$	2	5	1	5	6	9	5	10	3	2		
	$o_{0,3}$	10	4	5	9	8	4	15	8	4	4		
J_1	$o_{1,0}$	4	8	7	1	9	6	1	10	7	1	3	31.31
	$o_{1,1}$	6	11	2	7	5	3	5	14	9	2		
	$o_{1,2}$	8	5	8	9	4	3	5	3	8	1		
	$o_{1,3}$	9	3	6	1	2	6	4	1	7	2		
J_2	$o_{2,0}$	7	1	8	5	4	9	1	2	3	4	6	19.04
	$o_{2,1}$	5	10	6	4	9	5	1	7	1	6		
	$o_{2,2}$	4	2	3	8	7	4	6	9	8	4		
	$o_{2,3}$	7	3	12	1	6	5	8	3	5	2		
J_3	$o_{3,0}$	6	2	5	4	1	2	3	6	5	4	4	23.63
	$o_{3,1}$	8	5	7	4	1	2	36	5	8	5		
	$o_{3,2}$	9	6	2	4	5	1	3	6	5	2		
	$o_{3,3}$	11	4	5	6	2	7	5	4	2	1		
J_4	$o_{4,0}$	6	9	2	3	5	8	7	4	1	2	9	37.04
	$o_{4,1}$	5	4	6	3	5	2	28	7	4	5		
	$o_{4,2}$	6	2	4	3	6	5	2	4	7	9		
	$o_{4,3}$	6	5	4	2	3	2	5	4	7	5		
J_5	$o_{5,0}$	4	1	3	2	6	9	8	5	4	2	7	12.24
	$o_{5,1}$	1	3	6	5	4	7	5	4	6	5		
J_6	$o_{6,0}$	1	4	2	5	3	6	9	8	5	4	1	5.17
	$o_{6,1}$	2	1	4	5	2	3	5	4	2	5		
J_7	$o_{7,0}$	2	3	6	2	5	4	1	5	8	7	2	32.29
	$o_{7,1}$	4	5	6	2	3	5	4	1	2	5		
	$o_{7,2}$	3	5	4	2	5	49	8	5	4	5		
	$o_{7,3}$	1	2	36	5	2	3	6	4	11	2		
J_8	$o_{8,0}$	6	3	2	22	44	11	10	23	5	1	8	66.95
	$o_{8,1}$	2	3	2	12	15	10	12	14	18	16		
	$o_{8,2}$	20	17	12	5	9	6	4	7	5	6		
	$o_{8,3}$	9	8	7	4	5	8	7	4	56	2		
J_9	$o_{9,0}$	5	8	7	4	56	3	2	5	4	1	0	18.62
	$o_{9,1}$	2	5	6	9	8	5	4	2	5	4		
	$o_{9,2}$	6	3	2	5	4	7	4	5	2	1		
	$o_{9,3}$	3	2	5	6	5	8	7	4	5	2		
J_{10}	$o_{10,0}$	1	2	3	6	5	2	1	4	2	1	14	36.67
	$o_{10,1}$	2	3	6	3	2	1	4	10	12	1		
	$o_{10,2}$	3	6	2	5	8	4	6	3	2	5		
	$o_{10,3}$	4	1	45	6	2	4	1	25	2	4		
J_{11}	$o_{11,0}$	9	8	5	6	3	6	5	2	4	2	13	51.39

Table 8 (continued)

J_j	$o_{j,s}$	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	R_j	D_j
	$o_{11,1}$	5	8	9	5	4	75	63	6	5	21		
	$o_{11,2}$	12	5	4	6	3	2	5	4	2	5		
	$o_{11,3}$	8	7	9	5	6	3	2	5	8	4		
J_{12}	$o_{12,0}$	4	2	5	6	8	5	6	4	6	2	11	29.65
	$o_{12,1}$	3	5	4	7	5	8	6	6	3	2		
	$o_{12,2}$	5	4	5	8	5	4	6	5	4	2		
	$o_{12,3}$	3	2	5	6	5	4	8	5	6	4		
J_{13}	$o_{13,0}$	2	3	5	4	6	5	4	85	4	5	12	44.69
	$o_{13,1}$	6	2	4	5	8	6	5	4	2	6		
	$o_{13,2}$	3	25	4	8	5	6	3	2	5	4		
	$o_{13,3}$	8	5	6	4	2	3	6	8	5	4		
J_{14}	$o_{14,0}$	2	5	6	8	5	6	3	2	5	4	5	17.75
	$o_{14,1}$	5	6	2	5	4	2	5	3	2	5		
	$o_{14,2}$	4	5	2	3	5	2	8	4	7	5		
	$o_{14,3}$	6	2	11	14	2	3	6	5	4	8		

The processing time of operation $o_{j,s}$ on machine M_m and the release time R_j are collected from I4 (Kacem et al., 2002b).
The due time D_j is generated following the rule as defined in Table 4.

Table 9 The case data of the new arrival jobs

J_j	$o_{j,s}$	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	R_j	D_j
J_{15}	$o_{15,0}$	9	7	1	4	6	6	1	8	7	7	15.29	33.61
	$o_{15,1}$	5	9	3	5	2	6	7	5	4	2		
	$o_{15,2}$	3	5	7	9	1	9	8	1	9	3		
	$o_{15,3}$	8	3	3	6	8	9	9	3	1	10		
J_{16}	$o_{16,0}$	5	8	7	5	3	5	2	8	4	2	15.29	29.89
	$o_{16,1}$	8	4	7	9	10	2	9	6	6	2		
J_{17}	$o_{17,0}$	7	8	6	5	4	3	2	3	3	9	15.29	32.71
	$o_{17,1}$	5	6	1	10	4	2	8	3	6	2		
	$o_{17,2}$	1	6	5	10	2	2	1	1	4	4		
	$o_{17,3}$	2	2	4	1	1	8	8	7	8	4		
J_{18}	$o_{18,0}$	9	5	7	7	4	3	7	2	1	10	15.29	28.97
	$o_{18,1}$	2	8	2	1	6	7	1	6	7	1		
	$o_{18,2}$	4	7	6	9	7	3	2	8	10	1		

Table 9 (continued)

J_j	$o_{j,s}$	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	R_j	D_j
	$o_{18,3}$	9	9	1	10	9	6	6	8	2	8		
J_{19}	$o_{19,0}$	1	6	9	6	6	1	1	10	5	10	15.29	43.86
	$o_{19,1}$	8	10	5	6	5	2	10	8	3	2		
	$o_{19,2}$	8	1	7	8	9	1	10	8	5	9		
	$o_{19,3}$	6	7	2	5	1	1	3	8	4	10		

The processing time of operation $o_{j,s}$ on machine M_m, the release time R_j and the due time D_j are generated following the rule as defined in Table 4.

Funding This work is supported by the National Natural Science Foundation of China (Grant No. 72104016 and Grant No. 72304025), the Natural Science Foundation of Chongqing, China (Grant No. CSTB2023NSCQ-MSX0391), Beijing Natural Science Foundation (Grant No. 9242003), the R&D Program of Beijing Municipal Education Commission (Grant No. SM202110005011 and Grant No. SM202010005004), the Japan Society for the Promotion of Science (Grant No. P19800), the Funding of Centre International de Mathematiques et d'Informatique de Toulouse (Grant No. CIMI-ANR-ll-LABX-0040-LABX-2011) and the Foundation of Key Laboratory of Education Informatization for Nationalities (Yunnan Normal University), the Ministry of Education (Grant No. EIN2024C006).

Declaration

Conflict of interest The authors declare no potential conflicts of interest concerning the research, authorship, and/or publication of this article.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Agarwal, M., Agrawal, N., Sharma, S., Vig, L., & Kumar, N. (2015). Parallel multi-objective multi-robot coalition formation. *Expert Systems with Applications, 42*(21), 7797–7811.

Aguilar-Rivera, A. (2020). A GPU fully vectorized approach to accelerate performance of NSGA-2 based on stochastic non-domination sorting and grid-crowding. *Applied Soft Computing, 88*, 106047.

Ahmadi, E., Zandieh, M., Farrokh, M., & Emami, S. M. (2016). A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms. *Computers & Operations Research, 73*, 56–66.

Akram, K., Bhutta, M. U., Butt, S. I., Jaffery, S. H. I., Khan, M., Khan, A. Z., & Faraz, Z. (2024). A Pareto-optimality based black widow spider algorithm for energy efficient flexible job shop scheduling problem considering new job insertion. *Applied Soft Computing, 164*, 111937.

Alba, E., & Dorronsoro, B. (2009). *Cellular genetic algorithms*. New York: Springer.

Audet, C., Bigeon, J., Cartier, D., Le Digabel, S., & Salomon, L. (2021). Performance indicators in multiobjective optimization. *European Journal of Operational Research, 292*(2), 397–422.

Baykasoğlu, A., Madenoğlu, F. S., & Hamzadayı, A. (2020). Greedy randomized adaptive search for dynamic flexible job-shop scheduling. *Journal of Manufacturing Systems, 56*, 425–451.

Boschetti, M. A., Maniezzo, V., & Strappaveccia, F. (2016). Using GPU computing for solving the two-dimensional guillotine cutting problem. *INFORMS Journal on Computing, 28*(3), 540–552.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research, 41*(3), 157–183.

Cabrera, J. A., Simon, A., & Prado, M. (2002). Optimal synthesis of mechanisms with genetic algorithms. *Mechanism and Machine Theory, 37*(10), 1165–1177.

Capodieci, N., & Burgio, P. (2015). Efficient implementation of genetic algorithms on gp-gpu with scheduled persistent cuda threads. In: *2015 Seventh International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)* (pp. 6–12). IEEE.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197.

Demir, Y., & İşleyen, S. K. (2013). Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling, 37*(3), 977–988.

Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation, 1*(1), 3–18.

Dorronsoro, B., & Bouvry, P. (2013). Cellular genetic algorithms without additional parameters. *The Journal of Supercomputing, 63*(3), 816–835.

Fattahi, P., & Fallahi, A. (2010). Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability. *CIRP Journal of Manufacturing Science and Technology, 2*(2), 114–123.

Gao, K., Cao, Z., Zhang, L., Chen, Z., Han, Y., & Pan, Q. (2019). A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. *IEEE/CAA Journal of Automatica Sinica, 6*(4), 904–916.

Gupta, S., & Tan, G. (2015). A scalable parallel implementation of evolutionary algorithms for multi-objective optimization on GPUs. In: *2015 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1567–1574). IEEE.

Harada, T., & Alba, E. (2020). Parallel genetic algorithms: A useful survey. *ACM Computing Surveys (CSUR), 53*(4), 1–39.

Harris, M. (2012). An easy introduction to CUDA C and C++, https://devblogs.nvidia.com/parallelforall/easy-introduction-cuda-c-and-c/ .

https://www.dropbox.com/scl/fo/je7yu1f0uhrt60x1hd52l/AMaEkEI2JJXvdlSP6a2LJQs?rlkey=01407hsysw67tuujd8qtzhvzt&st=t7yll789&dl=0

https://www.ibm.com/analytics/spss-statistics-software

Hu, Y., Zhang, L., Zhang, Z., Li, Z., & Tang, Q. (2024). Matheuristic and learning-oriented multi-objective artificial bee colony algorithm for energy-aware flexible assembly job shop scheduling problem. *Engineering Applications of Artificial Intelligence, 133*, 108634.

Ishibuchi, H., & Murata, T. (1998). A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews), 28*(3), 392–403.

Kacem, A., & Dammak, A. (2021). Multi-objective scheduling on two dedicated processors. *TOP, 29*(3), 694–721.

Kacem, I., Hammadi, S., & Borne, P. (2002a). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 32*(1), 1–13.

Kacem, I., Hammadi, S., & Borne, P. (2002b). Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation, 60*(3–5), 245–276.

Kefalas, M., Limmer, S., Apostolidis, A., Olhofer, M., Emmerich, M., & Bäck, T. (2019). A tabu search-based memetic algorithm for the multi-objective flexible job shop scheduling problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (pp. 1254–1262).

Kim, D., & Kim, J. (2024). GPU-accelerated non-dominated sorting genetic algorithm III for maximizing protein production. *Electronic Research Archive, 32*(4), 2514–2540.

Lenstra, J. K., Kan, A. R., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics, 1*, 343–362.

Li, Y., & Wang, J. (2020). Multi-objective dynamic scheduling model of flexible job shop based on nsgaii algorithm and scroll window technology. In: *International Conference on Swarm Intelligence* (pp. 435–444). Springer, Cham.

Li, K., Deng, Q., Zhang, L., Fan, Q., Gong, G., & Ding, S. (2021). An effective MCTS-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem. *Computers & Industrial Engineering, 155*, 107211.

Li, R., Gong, W., & Lu, C. (2022). Self-adaptive multi-objective evolutionary algorithm for flexible job shop scheduling with fuzzy processing time. *Computers & Industrial Engineering, 168*, 108099.

Liu, J., Sun, B., Li, G., & Chen, Y. (2024). Multi-objective adaptive large neighbourhood search algorithm for dynamic flexible job shop schedule problem with transportation resource. *Engineering Applications of Artificial Intelligence, 132*, 107917.

Luan, F., Zhao, H., Liu, S. Q., He, Y., & Tang, B. (2023). Enhanced NSGA-II for multi-objective energy-saving flexible job shop scheduling. *Sustainable Computing: Informatics and Systems, 39*, 100901.

Luna, F., & Alba, E. (2015). Parallel multiobjective evolutionary algorithms. In: *Springer Handbook of Computational Intelligence* (pp. 1017–1031). Springer, Berlin, Heidelberg.

Luo, J., El Baz, D., Xue, R., & Hu, J. (2020). Solving the dynamic energy aware job shop scheduling problem with the heterogeneous parallel genetic algorithm. *Future Generation Computer Systems, 108*, 119–134.

Luo, J., Fujimura, S., El Baz, D., & Plazolles, B. (2019). GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem. *Journal of Parallel and Distributed Computing, 133*, 244–257.

Luo, S. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing, 91*, 106208.

Mahmud, S., Chakrabortty, R. K., Abbasi, A., & Ryan, M. J. (2022). Swarm intelligent based metaheuristics for a bi-objective flexible job shop integrated supply chain scheduling problems. *Applied Soft Computing, 121*, 108794.

May, G., Stahl, B., Taisch, M., & Prabhu, V. (2015). Multi-objective genetic algorithm for energy-efficient job shop scheduling. *International Journal of Production Research, 53*(23), 7071–7089.

Minella, G., Ruiz, R., & Ciavotta, M. (2008). A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing, 20*(3), 451–471.

Nguyen, S., Zhang, M., Alahakoon, D., & Tan, K. C. (2018). Visualizing the evolution of computer programs for genetic programming [research frontier]. *IEEE Computational Intelligence Magazine, 13*(4), 77–94.

Nouiri, M., Bekrar, A., Jemai, A., Niar, S., & Ammari, A. C. (2018). An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. *Journal of Intelligent Manufacturing, 29*(3), 603–615.

Ortega, G., Filatovas, E., Garzon, E. M., & Casado, L. G. (2017). Non-dominated sorting procedure for Pareto dominance ranking on multicore CPU and/or GPU. *Journal of Global Optimization, 69*(3), 607–627.

Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling, 12*(4), 417–431.

Padurariu, F. R., & Marinescu, C. (2014). NSGA-II: Implementation and performance metrics extraction for CPU and GPU. In: *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (pp. 494–499). IEEE.

Park, B. J., Choi, H. R., & Kim, H. S. (2003). A hybrid genetic algorithm for the job shop scheduling problems. *Computers & Industrial Engineering, 45*(4), 597–613.

Pharr, M., & Fernando, R. (2005). *Gpu gems 2: for high-performance graphics and general-purpose computation*. Boston: Addison-Wesley Professional.

Rahimi, I., Gandomi, A. H., Deb, K., Chen, F., & Nikoo, M. R. (2022). Scheduling by NSGA-II: Review and bibliometric analysis. *Processes, 10*(1), 98.

Rangsaritratsamee, R., Ferrell, W. G., Jr., & Kurz, M. B. (2004). Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers & Industrial Engineering, 46*(1), 1–15.

Reddy, M. S., Ratnam, C., Rajyalakshmi, G., & Manupati, V. K. (2018). An effective hybrid multi objective evolutionary algorithm for solving real time event in flexible job shop scheduling problem. *Measurement, 114*, 78–90.

Schaffer, J. D. A. R. (1989). *A study of control parameters affecting online performance of genetic algorithms for function optimization*. California: San Mateo.

Schryen, G. (2020). Parallel computational optimization in operations research: A new integrative framework, literature review and research directions. *European Journal of Operational Research, 287*(1), 1–18.

Shao, X., Liu, W., Liu, Q., & Zhang, C. (2013). Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology, 67*(9), 2885–2901.

Shen, X. N., & Yao, X. (2015). Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems. *Information Sciences, 298*, 198–224.

Tran, B. T., & Luong, N. H. (2024, July). On the investigation of multimodal evolutionary algorithms using search trajectory networks. In: *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 32–40).

Türkyılmaz, A., Şenvar, Ö., Ünal, İ, & Bulkan, S. (2020). A research survey: Heuristic approaches for solving multi objective flexible job shop problems. *Journal of Intelligent Manufacturing, 31*(8), 1949–1983.

Vallikavungal Devassia, J., Salazar-Aguilar, M. A., & Boyer, V. (2018). Flexible job-shop scheduling problem with resource recovery constraints. *International Journal of Production Research, 56*(9), 3326–3343.

Wang, Y., & Han, J. (2021). A FJSSP method based on dynamic multi-objective squirrel search algorithm. *International Journal of Antennas and Propagation, 2021*(1), 6062689.

Wong, M. L., & Cui, G. (2013). Data mining using parallel multi-objective evolutionary algorithms on graphics processing units. In: *Massively Parallel Evolutionary Computation on GPGPUs* (pp. 287–307). Springer, Berlin, Heidelberg.

Xie, J., Gao, L., Peng, K., Li, X., & Li, H. (2019). Review on flexible job shop scheduling. *IET Collaborative Intelligent Manufacturing, 1*(3), 67–77.

Xiong, J., Tan, X., Yang, K. W., Xing, L. N., & Chen, Y. W. (2012). A hybrid multiobjective evolutionary approach for flexible job-shop scheduling problems. *Mathematical Problems in Engineering, 2012*(1), 478981.

Xu, Y., Zhang, M., Yang, M., & Wang, D. (2024). Hybrid quantum particle swarm optimization and variable neighborhood search for flexible job-shop scheduling problem. *Journal of Manufacturing Systems, 73*, 334–348.

Zadeh, M. S., Katebi, Y., & Doniavi, A. (2019). A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times. *International Journal of Production Research, 57*(10), 3020–3035.

Zarrouk, R., Daoud, W. B., Mahfoudhi, S., & Jemai, A. (2022). Embedded PSO for solving FJSP on embedded environment (Industry 4.0 Era). *Applied Sciences, 12*(6), 2829.

Zhang, F., Mei, Y., & Zhang, M. (2019). Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics. In: *2019 IEEE Congress on Evolutionary Computation* (CEC) (pp. 1366–1373). IEEE.

Zhang, F., Mei, Y., Nguyen, S., & Zhang, M. (2022). Multitask multiobjective genetic programming for automated scheduling heuristic learning in dynamic flexible job-shop scheduling. *IEEE Transactions on Cybernetics, 53*(7), 4473–4486.

Zhang, G., Hu, Y., Sun, J., & Zhang, W. (2020). An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. *Swarm and Evolutionary Computation, 54*, 100664.

Zhang, Q., & Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation, 11*(6), 712–731.

Zhang, Z., Wu, L., Peng, T., & Jia, S. (2018). An improved scheduling approach for minimizing total energy consumption and makespan in a flexible job shop environment. *Sustainability, 11*(1), 179.

Authors and Affiliations

Jia Luo[1,2,3] · Didier El Baz[4] · Rui Xue[1] · Jinglu Hu[3] · Lei Shi[5,6]

✉ Rui Xue
xue.rui.bjut@hotmail.com

Jia Luo
jia.luo1125@qq.com

Didier El Baz
didier.el-baz@laas.fr

Jinglu Hu
jinglu@waseda.jp

Lei Shi
leiky_shi@cuc.edu.cn

[1] College of Economics and Management, Beijing University of Technology, No. 100 Ping Le Yuan, Chaoyang District, Beijing 100124, China

[2] Chongqing Research Institute, Beijing University of Technology, Chongqing 401121, China

[3] Graduate School of Information, Production and Systems, Waseda University, 2-7 Hibikino, Wakamatsu Ward, Kitakyushu, Fukuoka 808-0135, Japan

[4] LAAS-CNRS, Université de Toulouse, CNRS, 7 Avenue du Colonel Roche, 31031 Toulouse, France

[5] State Key Laboratory of Media Convergence and Communication, Communication University of China, Beijing 100024, China

[6] Key Laboratory of Education Informatization for Nationalities (Yunnan Normal University), Ministry of Education, Kunming 650092, China