Hybrid Flow Shop with Limited Transportation Scheduling Problem: A Comparison Between Genetics Algorithm, and a Novel Recursive Local Search Heuristic

Arash Amirteimoori

The London School of Economics and Political Science Department of Mathematics

Reza Kia (✉ reza.kia@gutech.edu.om)

German university of technology

Reza Tavakkoli-Moghaddam

University of Tehran College of Engineering

Research Article

Keywords: Scheduling, Hybrid flow shop, Limited transportation, Makespan, MILP, Recursive Local Search Heuristic, Genetic algorithm, Statistical tests

Posted Date: August 9th, 2021

DOI: <https://doi.org/10.21203/rs.3.rs-687470/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Hybrid flow shop with limited transportation scheduling problem: A comparison between Genetic algorithm, and a novel Recursive Local Search Heuristic

Arash Amirteimoori^a, Reza Kia^{b*}, Reza Tavakkoli-Moghaddam^c

^aDepartment of Mathematics, London School of Economics
and Political Science (LSE), London, UK

^bDepartment of Logistics, Tourism and Service Management,
German University of Technology in Oman (GUtech), Muscat, Oman

^cDepartment of Industrial Engineering, University of Tehran, Tehran, Iran

*Corresponding author. Tel.: +96895182391;
E-mail: reza.kia@gutech.edu.om

Abstract

In this paper, concurrent scheduling of jobs and transportation in a hybrid flow shop system is studied, where multiple jobs, transporters, and stages with parallel unrelated machines are considered. In addition to the mentioned technical features, jobs are able to omit one or more stages, and may not be executable by all the machines, and similarly, transportable by all the transporters. Unlike most studies in the literature, the transport resource is finite and needs to be simultaneously scheduled with the jobs. Initially, a new mixed integer linear programming (MILP) model is proposed to minimize the makespan. Then, a novel Recursive Local Search Heuristic (RLSH) is proposed to tackle the large-sized instances, which otherwise could not be solved via MILP solver (Gurobi) in a reasonable time. RLSH is also compared against Genetic Algorithm (GA) on a set of numerical examples generated from the uniform distribution. As the computational results demonstrate, it is concluded that RLSH is extremely efficacious dealing with the problem and outperforms GA in the objective value quality. Finally, using two well-known statistical tests: Wald and analysis of variance(ANOVA), we assess the performance of the suggested approaches.

Keywords— Scheduling, Hybrid flow shop, Limited transportation, Makespan, MILP, Recursive Local Search Heuristic, Genetic algorithm, Statistical tests

1 Introduction

Flow shop (FS), as one of the most applied production systems has been intensively studied over the last decade since it is considerably flexible and cost-efficient. Classically, FS consists of a number of stages each with one machine, where all the jobs should be processed on.

This form of FS is not practically able to satisfy the industries where mass-production is intended, or not all the jobs have the same production plan. Hence, some additional complementary features are to be taken into account to cover the mentioned requirements.

One characteristic is to consider the unrelated parallel machines on each stage to increase the process speed.

Other flexible attribute is the jobs stage omission, which permits the jobs to omit the stages if required. The transporter eligibility is also another technical characteristic which restricts a job to a certain set of carriers for some technical reasons (e.g., size or vulnerability).

A flow shop system with such flexible properties is called Hybrid Flow Shop Problem (HFSP), or sometimes called as Flexible Flow Shop System (FFSP), which covers more technicalities than the classic flow shop. Figure 1 demonstrates the general scheme of HFSP problem.

Figure 1. Hybrid flow shop with transportation

Generally, HFSP is NP-hard even if only two stages exist, where two parallel machines exist on the first stage, and one on the second stage [12]. Thus, it is not practically possible to optimally solve the problem in a reasonable running time if the size of the problem grows larger.

Traditionally, HFSP assumes the infinite transportation resource, which according to the literature, refers to the unlimited number of transporters. In other words, once a transportation is needed, it is assumed that carriers are always available to handle the transportation operation.

In the finite transportation assumed in this study, there might not always be an available carrier to deal with transportation operation, and the job should wait until one of the carriers is ready. Thus, it is also needed to schedule transportation resource concurrently with the job process.

Along with the manufacturing technicalities aforementioned, the input and output buffers are unlimited. Therefore, no capacity limitation exists on the number of jobs queuing at the stage for process, and for transportation from the current stage to the next one.

To tackle the problem, three approaches are employed: a new Mixed integer linear

programming (MILP), Genetic Algorithms (GA), and a novel Recursive Local Search Heuristic (RLSH).

The results produced by the methods are compared against one another on a set of numerical instances with setup time, process time, and travel time randomly generated from the uniform distribution. Since the problem is computationally complex, as long as Gurobi is able to deal with the problems in a reasonable running time, the comparison of all the approaches are continued. Otherwise, MILP is excluded from comparison. Eventually, analyzing the objective values and the computation times using statistical tests, the best approach is specified.

The rest of this study is structured as follows: Section 2 goes over the literature of flow shop scheduling problems with transportation. Section 3 presents the problem statement and solution approaches. In Section 4, the computational results are analyzed. Finally, in Section 5 some conclusions are provided.

2 Literature Review

HFSP problem is one the most computationally complicated problems in terms of solving to optimality. As Gupta [10] proved in his study, even in the two-stage problem with one machine at first stage and two machines at the second stage the problem is NP-Hard. Therefore, the combinatorial optimization is the most computationally reasonable approach for the large-sized instances. Below, we present the most relevant studies employing combinatorial optimization to deal with HFSP problem.

Maggu and Das [18] and Maggu et al. [17] studied scheduling problems with unlimited transportation resource while ignoring the capacity of the carriers to minimize makespan. The transportation times are dependent on the jobs and the buffers are unrestricted. It is also proved that the problem is polynomially solvable.

Kise et al. [14] studied two-machine flow-shop scheduling problem with transporters, where one job could be allocated to each transporter at each time. They also validated that even in the case of two machines, the problem is NP-complete.

Stevens and Gemmill [20] scrutinized a two-machine flow-shop with transportation minimizing the job waiting time. The buffer storage of the second machine is considered unlimited, whereas the first one has no buffer, and the carrier should immediately return after it reaches the second machine. As the solution approaches, two heuristics developed are practically able to generate fine solutions, but not optimal in all cases.

Bank and Werner [1] considered a flexible flow shop scheduling problem with un-

related parallel machines. The objective function is distributing the jobs to the machines and to schedule the jobs assigned to each machine so that the weighted sum of linear earliness and tardiness penalties is minimal. To deal with the problem, some structural properties useful to search for an approximate solution is employed. Additionally, various constructive and iterative heuristic algorithms are suggested and compared on a set of problems.

Brucker et al. [3] investigated flow shop scheduling problems with unlimited transportation and constructed an algorithm producing optimal solution if the process time of the jobs is set to 1.

Lee and Strusevich [15] examined the two-machine flow shop scheduling problems with an uncapacitated inter-stage transporter. To tackle the makespan minimization problem, they utilized an approximation heuristic given that there exists an optimal schedule with exactly two shipments.

Naderi et al. [19] exploited an improved simulated annealing to minimize makespan in FFS with sequence-dependent setup and transfer times, where the transfer times produced are uniformly distributed. Comparative tests support the robustness of the algorithm.

Fattahi et al. [8] investigated a HFSP with assembly operations, in which a number of the similar jobs are considered, and makespan is the objective function. Due to NP-hardness of the problem, a hierarchical branch and bound algorithm is presented. They also suggested some lower and upper bounds to augment the effectiveness of the developed algorithm.

Kheirandish et al. [13] studied a HFSP with two stages, wherein the jobs after process are assembled. In the first stage, the jobs are executed, and transferred to the second stage for assembly operation. A MILP model is developed to minimize the

makespan and Lingo is employed to tackle the problem. Since the model is NP-hard, an artificial bee colony algorithm in addition to GA are proposed to obtain near-optimal solutions in a reasonable time. Additionally, the efficiency of the solution approaches is investigated on different instances of the problem.

Gheisariha et al. [9] developed an promoted multi-objective harmony search (EMOHS) heuristic and a Gaussian mutation to address the flexible flow shop scheduling problems with sequence-based setup time, transfer time, and probable rework. A constructive heuristic is designed to construct the initial solution, and clustering is exercised to improve the solution. The suggested algorithm uses response surface methodology to simultaneously minimize both makespan and mean tardiness.

Yuan et al. [22] examined a two-machine flow shop group scheduling problem with job-related blocking and transportation times. To minimize makespan, a MILP model is formulated. Since the problem is strongly NP-hard, a co-evolutionary genetic algorithm (CGA) is proposed to tackle the problem, and a block-mining-based artificial chromosome construction mechanism is developed to accelerate the convergence process. The computational results reveal that CGA is effective dealing with the problem.

In the study carried out by Xin et al. [21], a permutation flow shop scheduling problem considering sequence-dependent setup time and a new conveyor speed control energy-saving mechanism is investigated. The objective is to discover the optimal sequence of jobs and conveyor belt speed scheme between any two nodes. A MILP model is developed to minimize both the makespan and total energy consumption. To address this bi-objective model, an enhanced discrete whale swarm optimization (IDWSO) is taken into account that combines differential evolution, the promoted search, and job-swapped mutation to augment the performance.

Lei et al. [16] studied FFSP with transport waiting times (FFSPDW), where AGVs handle the transportation operations. In order for minimizing makespan considering the dynamic waiting times, they suggested a waiting time calculation method to evaluate waiting time and makespan. This method takes two situations into account: infinite buffer capacity and zero buffer capacity. To solve the problem, a memetic algorithm integrated with waiting time calculation method is developed. Finally, they verified the algorithm parameters via ANOVA. Computational results support that the algorithm is able to deal with the problem in short computation time.

Cai et al. [4] examined the distributed permutation flow shop scheduling problem (DPFSSP) with transportation and eligibility constraints, where three objectives are considered: maximum completion time, maximum lateness and total costs (transportation and setup costs). First, they introduce the mathematical model and constructive heuristics for single objective. Then, an improved Non-dominated Sorting Genetic Algorithm II (NSGA-II) for the multi-objective DPFSSP to search for the Pareto optimal solutions is suggested.

Yuan et al. [23] employed a flow shop group scheduling problem, wherein both sequence-dependent setup time among the groups and round-trip transportation time among the machines are examined. To tackle the makespan objective function, a mixed integer linear programming model is suggested. Thereafter, an effective co-evolutionary discrete differential evolution algorithm (CDDEA) are developed. In the CDDEA approach, a number of problem-specific heuristic rules are suggested to generate the initial population. A new discrete differential evolution strategy and a cooperative-oriented optimization method are proposed to synergistically evolve

both the sequence of jobs in each group and the sequence of groups. Additionally, two lower bounds are considered to assess the CDDEA's solution quality. The results prove that the suggested CDDEA is efficient addressing the studied problem.

Batur et al. [2] reviewed a hybrid flow shop with one machine at the first and two at the second, in which a robot is assumed for transferring the jobs and the objective function is to determine the best cycle time. The buffer capacities between the stages and the robot travel time are respectively presumed unrestricted and constant. Employing simulated Annealing based heuristic, they tackled the problem, the results of which are compared against the efficacious lower bound values.

Elmi and Topaloglu [5] surveyed a new class of flexible flow shop scheduling problem consisting of a realistic technical characteristic: The simultaneously scheduling of the transporters and the jobs. To convey the jobs, multiple single-gripper robots are employed. A MILP model and simulated annealing are developed to find the best robot routes and the optimal job sequence at the same time.

Elmi and Topaloglu [6] studied the cyclic flow shop robotic cell scheduling problem with multiple robots conveying the jobs. No buffer between the machines is assumed and each part is carried to the next machine in a provided time lag given that the transfer time among machines is constant. To address the problem, a new MILP model is developed and the computational experiments support the robustness of the model.

An efficient multi bridge ant colony optimization was developed by Elmi and Topaloglu [7] to tackle the robotic flow shop with multiple robots. They suppose that there is no intermediate buffer among the processing machines, and the required time for the robots to transfer the jobs is considered constant.

Zabihzadeh and Rezaeian [24] examined flexible flow shop scheduling problem with robotic transportation and release time, where no buffer between stages is assumed and the transportation time is uniformly distributed. To address the problem, Genetic Algorithm(GA) and Ant Colony Optimization (ACO) are implemented. The performance of GA and ACO are similar on the small and medium-sized examples, however, GA outmatches ACO on the larger examples.

We use Table 1 to briefly state the remarkable characteristics of the most relevant studies above-mentioned, where IJT denotes the independent job process and transportation operations (I.e., both the job process and the transportation are not fulfilled by the same facility), ST stands for the setup time, SO characterizes the stage omission capability, and FMT stands for the transporters which are both finite and multiple in number.

Table1. Summary of the most relevant literature

Authors	Characteristics	LJT	ST	FMT	SO	Transporter Eligibility	Machine Eligibility	Heuristics	MetaHeuristics
Naderi et al (2009)	hybrid flowshop, improved SA	–	✓	–	–	–	–	–	✓
Elmi and Topaoglu (2014)	hybrid flowshop, SA	–	–	✓	–	–	✓	–	✓
Batur et al (2016)	hybrid flowshop, SA	✓	–	–	–	–	–	–	✓
Zabihzadeh and Rezaeian (2016)	hybrid flow shop, ACO, GA	✓	–	✓	–	–	✓	–	✓
Lei et al (2020)	flexible flow shop, the modified iterated greedy algorithm,	–	✓	✓	–	–	–	✓	–
Yuan et al (2020)	flexible flow shop, discrete differential evolution algorithm	–	✓	✓	–	–	–	✓	–
Yuan et al (2020)	flexible flowshop, a co-evolutionary GA, AGVs are restricted to the certain stages	–	–	✓	–	–	–	–	✓
Gheisariha et al (2021)	flexible flowshop, simulation optimization model	–	✓	–	–	–	–	–	✓
This study	hybrid flow shop, multiple transportation, RLSH, GA	✓	✓	✓	✓	✓	✓	✓	✓

As table 1 demonstrates, despite all the above-mentioned studies have both scientifically and practically made great achievements, they treated the simultaneous scheduling of jobs and transportation with some limiting assumptions, which are briefly described in the following:

-
- A number of studies aforementioned reviewed the infinite transport resources, a special case which rarely happens in reality. This assumption is only limited to industries using conveyor belt systems not all the industries.
 - Only Batur et al. [2] took the stage omission possibility into account, however, the considered HFSP problem is limited to two operational stages, hence, it is not viable for operationally complicated industries with numerous stages. Stage omission possibility provides considerable adaptability to the industries employing HFSP since it is possible to add a new job with different production plan.
 - In the studies fulfilled by Elmi and Topaoglu [5], the transportation and job process both are handled by robot, a characteristic which can considerably narrow down the application of the model. The robots considered in their studies are responsible for both transportation and process. So, after a robot has transferred a job to a stage, it should execute the job, and cannot start the next transport while executing the job. In other words, the transportation and job process are completely dependent and are unable to behave as two stand-alone components.
 - Zabihzadeh and Rezaeian [24] considered FFSP, where transportation is carried out by robots. In their study, setup time is not assumed. The MILP model presented does not sequence the travels of each robot realistically since travel time between the previous position of the robot to the current position of the job assigned to the robot is not considered, and only the travel time between the current position of the job and its next position is assumed.

-
- Some of studies assumed fixed transportation time between all the nodes, which technically is not sensible and practical.
 - In the works of Batur et al. [2], Lee and Strusevich [15], and Maggu and Das [18] the number of machines is restricted to two, which is far from practical considering the high-capacity production lines with the numerous machines most companies need.
 - None of the researches investigating HFSP with the limited transportation considered the transporter eligibility constraint, which from the technical point of view, is so remarkable since not all the jobs are transportable by all the carries given some restricting characteristics like job size, or job vulnerability.

To best of our knowledge, this study is the first research considering transporter eligibility, stage omission, and machine eligibility in a HFSP with finite multiple transporters. Furthermore, a new heuristic along with a novel MILP model are introduced.

3 Problem statement and solution approaches

In the HFSP , there are I jobs to be executed on J_i stages, each of which consists of M_j parallel unrelated machines.

To execute a job, the machine needs to be prepared for, therefore, the setup and process times both are taken into account.

As mentioned in section 1, the input and output buffers are unlimited, which implies that no limitation exists on the queue capacity for incoming and outgoing jobs. If the job coming on one of the eligible machines is the first job assigned, no setup time is required, since it is presumed that the machine is quite prepared for its first process in advance. Otherwise, the setup time is incurred after the process of the previous job is terminated. The process sequence of jobs on a machine is independent from the times they arrive at.

When the process of the job is complete, it should wait until one of the T_i transporters is ready to transport it.

After the job has been carried to the machine, the carrier either waits there to convey the job to the next stage after the process has been terminated, or it commences another travel and leaves conveying the in-process job to another carrier, or it returns later to convey it itself. The model opts out one of these strategies based on the improvement it results in the makespan.

Before going through the solution approaches, some assumptions are taken into account in the following:

1. All the jobs and transporters start their routes from stage zero.
2. All the jobs, machines, and transporters are available at time zero.

-
3. Each job, for each of its process can only be transported by one of the eligible carriers.
 4. Stage omission for jobs is allowed.
 5. At each stage, the job can only be executed by one of the eligible machines.
 6. Input and output buffers are assumed unlimited. I.e., there is no limitation on the number of the jobs waiting at the machine to be executed, or to be transferred from the machine to the next stage.
 7. Each transporter is only able to convey one job each time.

In the next section, we develop an efficacious mixed integer linear programming model to tackle the problem.

3.1 Mixed Integer Linear Programming model (MILP)

A novel mixed integer linear programming model (MILP) for hybrid flow shop scheduling problem with transportation along with the associated notations are in the following:

3.1.1 Indices

I : The number of jobs, where $i \in \{1, \dots, I\}$.

J : The number of stages (including stage zero), where $j \in \{0, 1, \dots, J\}$.

M : The number of machines (including input), where $m \in \{0, 1, \dots, M\}$.

T : The number of transporters, where $t \in \{1, 2, \dots, T\}$.

3.1.2 Sets

$M_{j,i}$: The machines of stage j eligible to execute job i .

T_i : The transporters technically able to transfer job i .

J_i : The stages, the job i should be processed on.

J_m : The stage, the machine m is located on.

$O_{i,j}$: A set determining the sequence of stages for job i , wherein the stage visited first by the job takes rank 1, the second stage visited takes rank 2 and so on.

3.1.3 Parameters

$ST_{i,j}$: The setup time of job i on stage j .

$PT_{i,j}$: The process time of job i on stage j .

$T_{m,m'}$: machine - to - machine travel time matrix.

BM : A big number (The traditional bigM).

3.1.4 Variables

$F_{i,j}$: The process finish time of job i on stage j .

$A_{i,j}$: The arrival time of the job i to stage j .

$Z_{i,j,i',j'}$: Travel time between position of job i on stage j and the that of job i' on stage j' .

$W_{i,m}$: 1, if the job i is processed by the machine m , and 0 otherwise.

$R_{i,j,t}$: 1, if transporter t transfers the job i to stage j , and 0 otherwise.

$X_{i,i',m}$: 1, if the job i is processed before the job i' on machine m , and 0 otherwise.

$S_{i,i',j,j'}$: 1, if the job i (Currently on stage j) is transferred before the job i' (Currently on stage j'), and 0 otherwise.

C_{Max} : Makespan, the maximum completion time of all the jobs.

3.1.5 The MILP model

$$\text{Minimize } C_{Max} \tag{1}$$

Subject to:

$$C_{Max} \geq F_{i,j} \quad i = 1, \dots, I, \quad j \in J_i \tag{2}$$

$$\sum_{m \in M_{j,i}} W_{i,m} = 1 \quad i = 1, \dots, I, \quad j \in J_i \tag{3}$$

$$\sum_{t \in T_i} R_{i,j,t} = 1 \quad i = 1, \dots, I, \quad j \in J_i \tag{4}$$

$$BM(2 - W_{i,m} - W_{i',m'}) + Z_{i,i',j,j'} \geq T_{m,m'} \quad i, i' = 1, \dots, I, \\ j = J_m \cap J_i, \quad j' = J_{m'} \cap J_{i'}, \quad m \in M_{j,i}, m' \in M_{j',i'} \tag{5}$$

$$A_{i,j} \geq F_{i,j'} + Z_{i,i,j,j'} \quad i = 1, \dots, I, \quad j, j' \in J_i, \quad O_{i,j} = O_{i,j'} + 1 \tag{6}$$

$$BM(3 - S_{i',i,j',j} - R_{i,j,t} - R_{i',j',t}) + A_{i,j} \geq A_{i',j'} + Z_{i',i,j',j} + \\ Z_{i,i,j,j+1} \quad i, i' = 1, \dots, I, \quad j \in J_i, \\ j' \in J_{i'}, \quad t \in T_i \cap T_{i'} \tag{7}$$

$$S_{i,i',j,j'} + S_{i',i,j',j} \leq 1 \quad i, i' = 1, \dots, I, \quad j \in J_i, \quad j' \in J_{i'} \tag{8}$$

$$F_{i,j} \geq A_{i,j} + PT_{i,j} \quad i = 1, \dots, I, \quad j \in J_i \tag{9}$$

$$BM(3 - X_{i,i',m} - W_{i,m} - W_{i',m}) + F_{i,j} \geq F_{i',j} + ST_{i,j} + PT_{i,j} \quad (10)$$

$$i, i' = 1, \dots, I, i \neq i', \quad m \in M_{i,j} \cap M_{i',j}, \quad j \in J_i \cap J_{i'}$$

$$BM(2 + X_{i,i',m} - W_{i,m} - W_{i',m}) + F_{i',j} \geq F_{i,j} + ST_{i',j} + PT_{i',j} \quad (11)$$

$$i, i' = 1, \dots, I, i \neq i', \quad m \in M_{i,j} \cap M_{i',j}, \quad j \in J_i \cap J_{i'}$$

$$X_{i,i',m} + X_{i',i,m} \geq W_{i,m} + W_{i',m} - 1 \quad i = 1, \dots, I, \quad i \neq i', \quad (12)$$

$$0 < j \leq J_i, \quad 0 < j' \leq J_{i'}$$

$$X_{i',i,m} + X_{i,i',m} \leq 1 \quad i = 1, \dots, I, \quad i \neq i', \quad 0 < j \leq J_i, \quad 0 < j' \leq J_{i'} \quad (13)$$

$$C_{Max} \geq 0 \quad (14)$$

$$Z_{i,i',j,j'} \geq 0 \quad i, i' = 1, \dots, I, \quad j \in J_i, \quad j' \in J_{i'} \quad (15)$$

$$A_{i,j} \geq 0 \quad i = 1, \dots, I, \quad j \in J_i \quad (16)$$

$$F_{i,j} \geq 0 \quad i = 1, \dots, I, \quad j \in J_i \quad (17)$$

$$W_{i,m} \in \{0, 1\} \quad i = 1, \dots, I, \quad j \in J_i, \quad m \in M_{i,j} \quad (18)$$

$$X_{i,i',m} \in \{0, 1\} \quad i, i' = 1, \dots, I, \quad j \in J_i \cap J_{i'}, \quad m \in M_{i,j} \cap M_{i',j} \quad (19)$$

$$S_{i,i',j,j'} \in \{0, 1\} \quad i, i' = 1, \dots, I, \quad j \in J_i, \quad j' \in J_{i'} \quad (20)$$

$$R_{i,j,t} \in \{0, 1\} \quad i = 1, \dots, I, \quad j \in J_i, \quad t \in T_i \quad (21)$$

Constraint (2) points out that makespan is at least equal to the time at which the last job is processed. Constraint (3) bounds each job to be processed by one of the eligible machines of the stage. Constraint (4) ensures that each job is conveyed by

one of the eligible transporters. Constraint (5) specifies the travel time between the current positions of two jobs. Constraint (6) states that the delivery time of job i should not be smaller than the sum of the process finish time of the job on the previous stage and the travel time between the previous position and the current one. Constraint (7) sequences all the transports performed by a carrier. After the job has been conveyed to the machine, the transporter either waits there to transfer the job to the next stage after the process has been terminated, or it begins another trip and leaves shipping the in-process job to another transporter, or later returns to transfer it itself. Constraint (8) guarantees that at most one of the two cases occurs, either job i is conveyed before job i' , or vice versa. Constraint (9) limits the process finish time of a job to be at least equal to the arrival time of the job in addition to the process time (We do not mention setup time here since this constraint covers general case, where the first job coming on a machine does not require setup operation, so, only the process time is considered here). Constraints (10) and (11) sequence the job processes on a machine. Constraints (12) and (13) indicates that if jobs i and i' are allocated to the same machine either job i is processed before job i' or vice versa. Constraints (14) to (21) state the range of values the variables can take.

3.1.6 The Illustrative Instance

In order to better communicate the concept of the MILP model, employing Figure 2, a simple illustrative instance is suggested, for which Tables 2 and 3 illustrate the values of all the parameters required.

According to Figure 2, three jobs need to be processed, where stage 1 is input section, stage 2 involves machines 1 and 2, and stage 3 consists of machine 3.

As the production plan, job 1 is only processed on stage 2 and other two jobs are

processed on stage 2 and then on stage 3. Hence, 5 production operations are to be scheduled. To transport the jobs, two carriers namely T1 and T2 handle all the transportation operations. The transporter 2 is the only carrier eligible to convey the job 1, while the jobs 2 and 3 can be transferred by both transporters.

The curved dark arrows each with a number above indicates the movement sequence of the carrier T1 and the dashed one shows that of T2 and the numbers refer to the order of the sequence. The large black dots with numbers below which placed on X coordinate demonstrate the time, at which the process of the respective job is complete, and the largest number among them is the makespan.

Table 2. The example's input

Stage	Job 1		Job 2		Job 3	
	Setup Time	Process Time	Setup Time	Process Time	Setup Time	Process Time
1	0	0	0	0	0	0
2	12	8	3	7	14	4
3	0	0	13	11	20	6

Table 3. Travel time matrix

Machine/Machine	Input	1	2	3
Input	0	5	5	7
1	5	0	1	4
2	5	1	0	4
3	7	4	4	0

Figure 2. The illustrative example

According to Figure 2, at the stage 2, jobs 3 and 2 are transported by transporter 1 and processed by machine 1. job 1 is conveyed by transporter 2 and executed by machine 2. Since stage 2 is the last operational stage of job 1, jobs 2 and 3 are the only jobs processed by machine 3 of stage 3. After all the operations have been fulfilled, 49 is resulted as the makespan.

The value of makespan is computed provided that the transportation resource is finite. As illustrated, the unavailability and the eligibility of transporter impacts the makespan. These technicalities are the main contribution of this model, which have been neglected in most past researches.

As far as we know, the only researches considering the finite multiple transporters are those fulfilled by Elmi and Topaoglu [5], and Zabihzadeh and Rezaeian [24]. In the study of Elmi and Topaoglu [5], the transportation and job process are completely

dependent and are unable to behave independently. So, the dependence between the transportation and the job process hugely limits the application of the model. Whereas in our study, the transportation and the job process are two stand-alone factors behaving independently.

The research carried out by Zabihzadeh Rezaeian [24] assumed multiple robots for transportation. To sequence the travels of a robot, the travel time between the current position of the robot and the current position of the job assigned to the robot is not considered. Hence, they only took into account the travel time between the current position of the job and its next position.

The flexibility granted by the machine eligibility, the stage omission, and the transporter eligibility, even more widens the range of the problems the model can handle. Since MILP is not able to solve large-sized problems, the genetic algorithm, and the novel recursive local search heuristic are presented in the next section.

3.2 Genetic Algorithm (GA)

Holland [11] introduced Genetic algorithm which is an adaptive method inspired by the genetic process of the biological organisms and are vastly utilized in many combinatorial optimization problems. The first and crucial step to carry out the Genetic Algorithm (GA) is to select the chromosome representation which is the way to encode a complete solution for implementing the GA operators.

The initial population of our GA is randomly generated using the uniform distribution and the complete solutions are constructed by the decoding method stated in Algorithm 1.

As the Algorithm 1 indicates, *List* refers to a list composed by a number of randomly

generated chromosomes, each of which is a complete solution to the problem. Every individual gene of a chromosome in *List* incorporates a number of selections. The selection consists of a job, a stage, an eligible transporter, and an eligible machine. For instance, if there are 3 jobs, each of which passes all the three stages, the length of the chromosome is $3 \times 3 = 9$, where each gene out of 9 genes is a selection of a job, a stage, a transporter, and a machine.

To construct the next generation, first, a number of best solutions(i.e., chromosomes) as selected by the cost evaluation process, is directly sent to the next generation. Thereafter, the remaining chromosomes for the next generation are produced via genetic operations: uniform crossover and mutation.

The uniform crossover operator is applied to the chromosomes to augment the exploration in the solutions. As Figure 3 shows, in this process, a random binary bit λ is generated for each gene and the value of the gene in the first offspring is defined by $X_{Child1} = \lambda X_{Parent2} + (1 - \lambda)X_{Parent1}$, where each gene X_{Child1} of the child 1 is either the gene value of $X_{Parent1}$ or $X_{Parent2}$.

In this process, swapping the genes of two parents may result in infeasible children, to prevent which two strategies to avoid the infeasibility are utilized.

In the first strategy, the algorithm opts out a gene from parent 1 and looks for a gene with the technically similar characteristics (i.e., same job and stage indices) in parent 2, and then exchange them.

In the latter strategy, regardless of the properties of the chosen gene in parent 2, the algorithm exchanges the genes, then, if the job and stage of both were similar, there would be no technical issue, but if they were not identical(e.g., job 1, stage 1 in gene 1, and job 2, stage 2 in gene 2), the swapping would lead to the infeasible children. Consequently, the result would be two genes in child 2 with job 1 and stage 1 and two

genes in child 1 each with job 2 and stage 2. To address this problem, after swapping, the algorithm commences scanning from the very left side of child 2, and once the second gene with job 1 and stage 1 has been detected, the algorithm replaces the the job 1 and the stage 1 by the job 2 and the stage 2.

Now, the job and the operation indices make sense, and to correct the machine and transporter indices, the algorithm randomly selects one of the eligible machine from stage 2, and one of the qualified transporters. Similarly, the same procedure is used for child 1. After all the steps above-mentioned have been implemented, sorting the offsprings (i.e., children 1 and 2) according to the stage order of the jobs, two new feasible children are resulted. Figure 3 demonstrates this mechanism.

Figure 3. The uniform cross over

To mutate a parent, a number of genes from each parent is selected and their machine and transporter indices are randomly regenerated considering the machine and transporter eligibility. The number of selected genes is randomly determined, which varies from 1 to the number of genes in the parent.

Algorithm 1 illustrates the implementation of the GA algorithm.

Algorithm 1 GA

Input: Jobs_List, Stages_List, Machines_List, Setup_Times, Process_Times, TravelTime_Matrix, Iterations_Number, Population_Size, Crossover_Size, Mutation_Size, Job-Vehicle_Incidence_Matrix, Job-Stage_Incidence_Matrix, Job-Machine_Incidence_Matrix.

Output: The best Makespan found.

```
1: List={} {A list to save chromosomes}
2: BestSol=A_big_Number {The best makespan found}
3: nPop=Population_Size{The size of main population}
4: Pc=Crossover_Size{The percent of population to apply crossover}
5: Pm=Mutation_Size{The percent of population to apply mutation}
6: Nc=Pc×nPop {The size of population to apply crossover}
7: Nm=Nc×nPop {The size of population to apply mutation}
8: MaxIt=Number_of_Iterations { Times the algorithm should run}
9: for i in Range(nPop) do
10:   X=Generate a Random Solution(chromosome)
11:   Add X to List
12: end for
13: for It in Range(MaxIt) do
14:   PopC= {} {To store population of crossovers and mutations}
15:   for j in Range(Round(Nc/2)) do {By each Crossover, two children are generated}
16:     Parent1=Select a random chromosome from List
17:     Parent2=Select a random chromosome from List
18:     Child1,Child2=UniformCrossOver (Parent1,Parent2)
19:     Add Child1 and Child2 to PopC
20:   end for
21:   for k in Range(Round(Nm)) do{Nm mutations are carried out}
22:     Parent=Select a random chromosome from List
23:     Child=Mutation(Parent)
24:     Add Child to PopC
25:   end for
26:   Add PopC to List
27:   Cost={} { A list to store the cost of chromosome}
28:   for c in Length(List) do
29:     Cost[c]=Compute the cost of List[c]
30:   end for
31:   Sort(List) { Sort list based on cost}
32:   Temp={}
33:   for t in Range(nPop) do
34:     Add List[t] to Temp { Select the best solutions to the size nPop for the next generation}
35:   end for
36:   List=Temp { list for next generation}
37:   if BestSol>List[0] then
38:     BestSol=List[0]
39:   end if
40: end for
```

3.3 The Recursive Local Search Heuristic (RLSH)

RLSH applies a recursion mechanism to explore through the solution space, and unlike meta-heuristics, where solutions are created randomly, the solution generation is cost-oriented. Thus, given that cost is a determining factor in the solution generation process, each solution is rather a good solution.

As Algorithm 2 illustrates, *Combination* refers to an incomplete solution to a problem structurally identical to a chromosome in GA, except, it is not a solution yet. In this algorithm, all the updates will be copied on *Combination* as it is supposed to be a complete solution to the problem.

As the lines 8 to 17 of Algorithm 2 demonstrate, in each iteration, the Algorithm 2 chooses one job, its operational stage, an eligible transporter, and one of the eligible machines at operational stage. Then, stores this selection as a "Case". After this case has been considered, the algorithm adds its updates on a copy of input *Combination*.

Each time the algorithm generates a case, thereafter, adds the case to a copy of the input combination. Running the lines 8 to 17 of Algorithm 2, the input combination indicated in line 1 of Algorithm 1 will be copied $\sum_{i \in Jobs} \sum_{j \in J_i} M_{i,j} \times T_i$ times, where $M_{i,j}$ stands for the machines eligible to execute j^{th} operation of the job i , and T_i denotes the transporters eligible to transfer job i .

Each of the newly generated combinations have a new case added to the current cases. Thereafter, they are added to *List*, then, applying the sort operation to *List*, a number of combinations as specified in lines 19 are selected for next recursion, and the rest are discarded. To this end, *InitialPer* which is a decimal is multiplied by the length of the *List* and the result is rounded to nearest integer. This process yields a

shorter list consisting of the selected combinations for the next recursion.

As line 20 implies, following *list* shortening operation, if each *Combination* included in *List* is a complete solution, then it is added to *GlobalList* as one of the candidates for final evaluation, otherwise, it will go through the lines 23 to 27 for the next recursion. In the next recursion, the recurred *Combination* will be treated as the input *Combination* with all the updates it brought along from the previous recursion.

In lines 23 to 25, before the incomplete *Combination* goes through the next recursion, its *InitialPer* is decreased by *DecreasePer* in order to narrow down the next generated branches from the incomplete *Combination*. If this operation ended up in the negative value, the *InitialPer* will be equal to the shortest acceptable length *ShortAccLen*.

The above-mentioned process will be recursively repeated until the number of the complete *Combination* added to *GlobalList* is equal to the allowed number of solutions for evaluation. Algorithm 2 clearly demonstrate the aforementioned process.

Algorithm 2 RLSH

Input: Setup_Times, Process_Times, TravelTime_Matrix, Decimal_1, Decimal_2, Decimal_3, Job-Transporter_Incidence_Matrix, Job-Stage_Incidence_Matrix, Solution

Output: The best Makespan found.

- 1: **Combination**=Solution {An incomplete solution}
- 2: **List**={} {An empty list each element of which is supposed to be a **Combination**}
- 3: **Makespan**=0
- 4: **InitialPer**=Decimal_1 {Proportion of **Combinations** in **List** to apply the recursion on each}
- 5: **DecreasePer**=Decimal_2 { A negative decimal value decreasing the percent of **InitialPer** in each recursion(to narrow down the number of evaluated cases, as the number of recursions increases)}
- 6: **ShortAccLen**=Decimal_3 { A positive decimal to keep **InitialPer** positive if it gets negative}
- 7: **Cost_List**={} { A list to store the cost(makespan) of each solution (**Combination**)}
- 8: **for** i in jobs **do**
- 9: **for** j in the machines of $M_{i,OpStage}$ **do**{ The machines of the operational stage eligible to execute job i }
- 10: **for** t in the Eligible_Transporters **do** {given job-Vehicle_Incidence_Matrix}
- 11: **TempCombination**=Copy(Combination)
- 12: **Cost_Combination_Updated**=Algorithm3.Update(i, j, t, Setup_Times, Process_Times, TravelTime_Matrix, **TempCombination**)
- 13: Add(**Combination_Updated**) to **List**
- 14: Add(**Cost**) to **Cost_List**
- 15: **end for**
- 16: **end for**
- 17: **end for**
- 18: Sort(**list**) {Sort **List** based on the cost values}
- 19: **for** k in Length(Round(**List*****InitialPer**)) **do**{The For loop applies to the first **List*****InitialPer** elements of the list }
- 20: **if** Operations of all the jobs of
- 21: List[k] are carried out **then**
- 22: Add **List**[k] to a **GlobalList** {Global List is declared outside of RLSH class to save the generated solutions}
- 23: **else**
- 24: Decimal_1=**InitialPer-DecreasePer**
- 25: **if** Decimal_1 is **NonPositive** **then**
- 26: Decimal_1=**ShortAccLen**
- 27: **end if**
- 28: Recursively Run RLSH(Setup_times, Process_times, TravelTime_Matrix, Decimal_1, Decimal_2, Decimal_3, Job-Transporter_Incidence_Matrix, Job-Stage_Incidence_Matrix, **List**[k])
- 29: **end if**
- 30: **end for**
- 31: Sort(**GlobalList**) {The cost of GlobalList[0] is the best Makespan found}

Despite the notations defined as input are obvious, some of which may not convey the meaning clearly. Hence, we provide complementary information for some notations.

Job – Transporter_Incidence_Matrix refers to the transporters qualified to transfer the job. *Job – Stage_Incidence_Matrix* denotes the stages considered for the job.

In the line 12 of the proposed RLSH algorithm, Algorithm 3 is called which updates the input *Combination*. After the updates have been done on *Combination*, the new case is added to *Combination* and the case with the maximum completion time

among the finished ones is the current cost of *Combination*.

Algorithm 3 Update

Input: Job, Machine, Transporter, Setup_Times, Process_Times, TravelTime_Matrix, Combination
Output: Max_Cost, Updated_Combination

- 1: $\mathbf{X} = \{\text{The time the transporter arrives from its previous position to the job's current position.}\}$
- 2: $\mathbf{Y} = \{\text{The time the transporter arrives from its previous position (the job's current position) to the machine to deliver the job.}\}$
- 3: $\mathbf{Cost} = \{\text{Cost of the new case.}\}$
- 4: $\mathbf{MaxCost} = \{\text{Cost of the case with maximum cost in Combination.}\}$
- 5: **Begin**
- 6: Compute $\mathbf{X} = \text{Vehicle_AvailabilityTime} + \text{TravelTime}[\text{Vehicle_CurrentPos}][\text{Job_CurrentPos}]$
- 7: $\text{Vehicle_CurrentPos} = \text{Job_CurrentPos}$
- 8: Compute $\mathbf{Y} = \mathbf{X} + \text{TravelTime}[\text{Vehicle_CurrentPos}][\text{Machine}]$
- 9: $\text{Vehicle_CurrentPos} = \text{Machine}$
- 10: Compute $\mathbf{Cost} = \mathbf{Max} \{ \mathbf{Y}, (\text{Machine_AvailabilityTime} + \text{Setup_Time}(\text{if required})) \} + \text{Process_Time}$
- 11: {In the case the sum of machine availability time and the setup time(if required) is smaller or equal to $\mathbf{Y}$, setup time is not considered in line 10. Else, as soon as the machine is available, the setup operation starts.}
- 12: $\text{Vehicle_AvailabilityTime} = \mathbf{Y}$ {The time the vehicle is available again.}
- 13: $\text{Machine_AvailabilityTime} = \mathbf{Cost}$ {The time the machine is available again.}
- 14: $\text{Job_AvailabilityTime} = \mathbf{Cost}$ {The time the job is ready to be transferred.}
- 15: $\text{Job_OpStage} = \text{next Job_OpStage}$ {The next stage is updated given Job-Stage_Incidence_Matrix.}
- 16: Construct the **Case** {The new case including the input job, machine's stage, machine, and transporter is built, and all updates are also copied.}
- 17: $\text{Updated_Combination} = \text{Add Case to Combination}$
- 18: Find $\mathbf{Max_Cost}$ {The cost of Update_Combination}
- 19: **End**
- 20: **Return** Max_Cost, Updated_Combination

After Algorithms 2 and 3 have been run, and all the produced combinations are complete, a number of solutions to the problem are obtained, the minimum of which is the best makespan found. To better convey the key concepts of the RLSH algorithm, an example in Figure 6 is illustrated, where two jobs, one transporter, and three machines are considered. Machines 1 and 2 are at stage 1 and Machine 3 is located at stage 2. Both jobs have the same operational route: visiting one of the machines 1 and 2 on the first operation, then moving to Machine 3 for the next operation.

Techninal Input= InitialPer=0.5, DecreasePer=0.1666, ShortAccLen=1, Products=2, Machines= 3 , Transporter=1

Best Makespan= 95 with Cases 1, 5, 4, and 6

Figure 4. The RLSH example

As reflected in Figure 4, in the first level, 4 cases can at most be considered. Each case is copied on a incomplete "Combination", then added to "List". Hence, in the first level, there are four combinations in the list, each of which is a branch which can lead to a solution to the problem.

After the combinations have been stored in the list, they are sorted, and a number of them (i.e., 2 combinations) with lower costs are selected for the next recursion. This process takes place in line 19 of Algorithm 2.

Each of these two combinations has already one case added, and in the next recursion, the algorithm finds the second best case for each.

In level 2 (next recursion), there are three possible cases for each combination, and after the sort operation, a number of them are chosen for the next recursion.

Now, one combination on the left branch (list) and one on the right branch(list) are continued, and each has two possible cases as demonstrated in level 3 (third recursion). After they are sorted, one of each (According to line 19 of Algorithm 2) is selected for the next recursion.

Presently, each continuing combination includes 3 cases, and in level 4 (fourth recursion), the last case is added and the number of cases for each combination is equal to the number of total operations ($\sum_{i \in Jobs} Operations_i$) which indicates that all the required operations have been processed and the algorithm is terminated.

Consequently, comparing the two resulted combinations, it is deduced that the left branch produced a better makespan which is the best solution found.

4 Computational Experiments

In this section, the results of computational experiments assessing the performances of the proposed approaches are presented.

Similar to Elmi and Topaloglu’s research [5], the sizes of problems were established by changing the levels of four parameters, which respectively are the jobs, stages, machines, and transporters. The developed MILP was solved via Gurobi optimizer and along with GA and RLSH algorithms were implemented in Python. All the solution approaches were run on a laptop with Intel Core i9 2.3 GHz CPU with 16 GB memory.

4.1 Parameters Setting

In order to calibrate the developed GA and RLSH, the Taguchi method is employed to statistically design the empirical investigation. Essentially, the traditional procedures to design experimental investigation are too complex and difficult to fulfill since a large number of experiments have to be carried out once the number of parameters augments. To address the issue, the Taguchi method efficiently examines the entire parameter space using only a small number of experiments.

For this purpose, we define the assessment parameter $Sol_{Length} = (length(Products) \times length(Stages) \times length(Machines) \times length(Vehicles))$. To best exploit the algorithms, for each individual range of Sol_{Length} , the parameters are calibrated and indicated in Table 4. Alongside the parameters considered for RLSH, we set limitation on the maximum number of the generated solutions (MGS) in order to terminate the algorithm in a reasonable time.

Table 4. Parameter calibration

Sol_{Length}	RLSH				GA			
	InitialPer	MGS	ShortAccLen	DecreasePer	Population	Max	Nc	Nm
	Iteration							
1-50	1	-	1	0	200	200	0.9	0.7
51-200	1	-	1	0	300	300	0.9	0.3
201-500	1	-	0.3	0.3	400	600	0.7	0.25
501-3000	1	10000	0.02	0.5	400	500	0.65	0.25
3001-10000	1	1500	0.002	0.9	350	450	0.65	0.25
$10001 \leq$	0.3	300	0.00001	0.2999	300	300	0.6	0.15

4.2 Numerical Instances

Employing parameter setting presented in Table 4, we dealt with 36 different-sized problems, which are reported in Table 5, where Sol_{Length} refers to the scale of the problem, and Time(m) stands for the computation time in minutes. The column of jobs ranges from 9 to 500, where the total number of operations (NOP) is stated. The number of jobs varies from 3 to 50, which by the carriers ranging from 1 to 8 are distributed to the stages varying from 3 to 10. The number of machines placed on the stages also fluctuates between 1 and 8.

The uniform distribution is used to randomly generate the process times, setup times, and distance matrix from $U[0, 25]$.

Table 5. Computational results

No	NOP	Jobs	Stages	Machines	Transporters	Sol(Len)	Gurobi(Optimal)		GA		RLSH	
							Obj	Time(m)	Obj	Time(m)	Obj	Time(m)
1	9	3	3	3	1	27	139	0.01	139	0.01	139	0.01
2	12	3	4	4	1	48	84	0.01	84	0.01	84	0.01
3	12	4	3	3	2	72	90	0.01	90	0.01	90	0.01
4	16	4	4	5	1	80	191	0.05	191	0.05	191	0.05
5	15	5	3	3	3	135	100	0.1	100	0.2	100	0.1
6	21	7	3	3	3	189	88	0.2	88	0.2	88	0.3
7	24	6	4	4	2	192	270	2	270	1	270	0.5
8	42	4	6	7	2	336	88	1	88	1	88	0.05
9	33	11	3	3	4	396	135	4	135	1	135	1
10	80	20	4	5	1	400	900	20	900	3	925	1
11	36	6	6	6	2	432	153	2	153	1	153	0.5
12	80	16	5	6	1	480	1190	40	1250	3	1254	1
13	44	11	4	4	3	480	182	17	201	1	192	1
14	64	16	4	4	2	512	-	-	307	3	301	2
15	75	15	5	5	2	750	-	-	470	2	470	2
16	75	15	5	7	2	1050	-	-	439	1	450	1
17	65	13	5	5	4	1300	-	-	286	2	286	2
18	68	17	4	7	3	1428	-	-	280	1	270	2
19	78	13	6	7	3	1638	-	-	287	1	276	1
20	80	16	5	7	3	1344	-	-	275	2	265	1
21	80	16	5	8	3	1920	-	-	265	1	235	0.3
22	85	17	5	8	3	2040	-	-	335	2	326	0.5
23	102	17	6	8	3	2448	-	-	461	3	438	0.5
24	108	18	6	8	3	2592	-	-	446	2	418	0.5
25	120	20	6	10	3	3000	-	-	489	1	443	1
26	132	22	6	10	3	5808	-	-	529	2	514	0.7
27	196	28	7	11	4	9408	-	-	597	1	595	0.5
28	210	30	7	11	4	10800	-	-	494	1	494	0.5
29	224	32	7	12	5	14700	-	-	428	1	414	1
30	280	35	8	14	5	19600	-	-	788	2	688	2
31	296	37	8	15	5	22200	-	-	568	2	454	2
32	320	40	8	15	6	28800	-	-	737	2	593	2
33	387	43	9	16	6	37152	-	-	802	3	667	2
34	405	45	9	17	7	48195	-	-	850	4	695	3
35	480	48	10	18	7	60480	-	-	809	4	687	4
36	500	50	10	20	8	80000	-	-	925	5	684	5

Figure 5. Computational results

As both Table 5 and Figure 5 corroborate, for Gurobi is not computationally capable to tackle the large-sized problems, from the thirteenth instance on, it is removed from comparison. Table 5 and Figure 5 also indicate that the larger the size of the numerical instances grows, the better RLSH performs than GA. To support this, similar to Elmi and Topaloglu [5], we utilize Wald test.

Before implementing Wald test, it is necessary to compute the gap between the means

of GA and RLSH using the formulas below.

$$MeanGAP = \left(\frac{\overline{GA}_{Obj} - \overline{RLSH}_{Obj}}{\overline{RLSH}_{Obj}} \right) \times 100 \quad (22)$$

Computing the equation (20), *MeanGAP* equals to 5.29 %. In other words, if the solutions obtained by RLSH algorithm is implemented in a real manufacturing system, a given annual production volume of this system is gained within 5.29% shorter time than GA.

Due to the stochastic nature of the suggested GA, it may lead to a distinctive GAP for dissimilar problem instances. Thus, GAP can be treated as a random variable and using GAP formulas, the data reported in Table 5 are considered as a sample for this variable. Let μ indicates the value of GAP to be tested for the following hypothesis:

$$\begin{aligned} H_0 : \mu &\geq 5.29 \\ H_1 : \mu &< 5.29 \end{aligned} \quad (23)$$

Let us presume the probability of a type II error as $\beta = 0.1$, and $(1 - \alpha) = 0.95$ confidence interval for a sample of size 36. The value of t-statistic is calculated via the following equation:

$$t_{GA} = \left(\frac{\sqrt{n}(IndGAP - 5.29)}{S_{GA}} \right) \quad (24)$$

Where *IndGAP* and *S* characterize the sample mean and standard deviation respectively. These parameters are computed as *IndGAP*=5.0252, *S_{GA}* = 7.447 and *t_{GA}* = -0.214 for GA. Since *t_{GA}* is larger than $t_{0.05,36} = -1.6905$, there is no solid

evidence against the H_0 hypothesis. Hence, the proposed RLSH reduces the average makespan of GA by 5.29 % with 0.95 confidence interval.

Figure 6. Box plot of the computational times

Implementing Wald test, it has been ratified that RLSH augments the productivity. Employing ANOVA one-way test, we assess if there are significant differences between the computation times of GA and RLSH as demonstrated in Table 5 and Figure 5. The hypothesis is as below given than the significance level is 0.05:

$$\begin{aligned}
 H_0 : p &< 0.05 \\
 H_1 : p &\geq 0.05
 \end{aligned}
 \tag{25}$$

Table 6. The ANOVA test summary

	df	sum_sq	mean_sq	F	P value
C(Solution Approaches)	1	4.339	4.229	3.046	0.08539
Residuals	70	97.173	1.388	NaN	NaN

As illustrated by Table 6 and Figure 6 , $pvalue = 0.08539$ is larger than 0.05, supporting that no significant difference exists, and consequently, H_0 is rejected. Based on the test results, we infer that there is no significant difference between computation times of the solution approaches.

5 Conclusion

In this study, we scrutinize the hybrid flow shop scheduling problem (HFSP) with transportation, where there are multiple jobs, transporters, stages, and machines. Technically, the job may skip some stages, and may not be transportable by all the carriers, and executable by all the machines. Initially, a novel mixed-integer linear programming (MILP) model is proposed to minimize makespan. Since the MILP model is unable to generate favorable solutions in a reasonable computing time as the size of the problems augments, two alternative solution approaches namely GA and a new heuristic(i.e., RLSH) are developed and compared against each other on a set of 36 numerical instances generated from uniform distribution.

As the Wald test supports, RLSH outperforms GA in performance, and using ANOVA test with $\alpha = 5\%$, it is concluded that there is no significance difference between the computation times of the solution approaches.

Finally, despite the RLSH is computationally fast and reliable, it is still unproved if RLSH is able to compete with other meta-heuristics such as PSO, and SA. Therefore, it is strongly recommended to compare RLSH against PSO and SA on a larger set of instances. Furthermore, the uncertain nature of the parameters can be studied and tackled using uncertainty techniques such as fuzzy programming, robust optimization and stochastic optimal control.

6 Declaration

6.1 Funding

Not applicable.

6.2 Conflicts of interest

There is no Conflicts of interest.

6.3 Availability of data

Data are randomly generated.

6.4 Code availability

It is available in the format of Python coding language.

6.5 Ethics approval

All procedures performed in the study are in accordance with the ethical standards of the institutional and/or national research committee.

6.6 Consent to participate

All the authors (i.e. Arash Amirteimoori, Reza Kia, and Reza Tavakkoli-Moghaddam) formally confirm the participation consent in this research.

6.7 Consent for publication

All the authors (i.e. Arash Amirteimoori, Reza Kia, and Reza Tavakkoli-Moghaddam) formally confirm the unanimity on publishing this research.

References

- [1] Jan Bank and Frank Werner. Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modelling*, 33(4-5):363–383, 2001.
- [2] G Didem Batur, Serpil Erol, and Oya Ekin Karasan. Robot move sequence determining and multiple part-type scheduling in hybrid flexible flow shop robotic cells. *Computers & Industrial Engineering*, 100:72–87, 2016.
- [3] Peter Brucker, Sigrid Knust, TC Edwin Cheng, and Natalia V Shakhlevich. Complexity results for flow-shop and open-shop scheduling problems with transportation delays. *Annals of Operations Research*, 129(1-4):81–106, 2004.
- [4] Shuang Cai, Ke Yang, and Ke Liu. Multi-objective optimization of the distributed permutation flow shop scheduling problem with transportation and eligibility constraints. *Journal of the Operations Research Society of China*, 6(3):391–416, 2018.
- [5] Atabak Elmi and Seyda Topaloglu. A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. *Computers & Operations Research*, 40(10):2543–2555, 2013.
- [6] Atabak Elmi and Seyda Topaloglu. Multi-degree cyclic flow shop robotic cell scheduling problem: Ant colony optimization. *Computers & Operations Research*, 73:67–83, 2016.

-
- [7] Atabak Elmi and Seyda Topaloglu. Multi-degree cyclic flow shop robotic cell scheduling problem with multiple robots. *International Journal of Computer Integrated Manufacturing*, 30(8):805–821, 2017.
- [8] Parviz Fattahi, Seyed Mohammad Hassan Hosseini, Fariborz Jolai, and Reza Tavakkoli-Moghaddam. A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations. *Applied Mathematical Modelling*, 38(1):119–134, 2014.
- [9] Elmira Gheisariha, Madjid Tavana, Fariborz Jolai, and Meysam Rabiee. A simulation–optimization model for solving flexible flow shop scheduling problems with rework and transportation. *Mathematics and Computers in Simulation*, 180:152–177, 2021.
- [10] Jatinder ND Gupta. Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 39(4):359–364, 1988.
- [11] John H Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.
- [12] Johannes Adzer Hoogeveen, Jan Karel Lenstra, and Bart Veltman. Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard. *European Journal of Operational Research*, 89(1):172–175, 1996.
- [13] Omid Kheirandish, Reza Tavakkoli-Moghaddam, and Mehdi Karimi-Nasab. An artificial bee colony algorithm for a two-stage hybrid flowshop scheduling problem with multilevel product structures and requirement operations. *International Journal of Computer Integrated Manufacturing*, 28(5):437–450, 2015.

-
- [14] Hiroshi Kise, Tadayoshi Shioyama, and Toshihide Ibaraki. Automated two-machine flowshop scheduling: A solvable case. *IIE Transactions*, 23(1):10–16, 1991.
- [15] Chung-Yee Lee and Vitaly A Strusevich. Two-machine shop scheduling with an uncapacitated interstage transporter. *IIE Transactions*, 37(8):725–736, 2005.
- [16] Chuanjin Lei, Ning Zhao, Song Ye, and Xiuli Wu. Memetic algorithm for solving flexible flow-shop scheduling problems with dynamic transport waiting times. *Computers & Industrial Engineering*, 139:105984, 2020.
- [17] Parkash Lal Maggu, Ghansham Das, and Ravinder Kumar. On equivalent-job for job-block in $2 \times n$ sequencing problem with transportation-times. *Journal of the Operations Research Society of Japan*, 24(2):136–146, 1981.
- [18] PL Maggu and G Das. On $2 \times n$ sequencing problem with transportation times of jobs. *Pure and Applied Mathematika Sciences*, 12(1):1–6, 1980.
- [19] B Naderi, M Zandieh, A Khaleghi Ghoshe Balagh, and V Roshanaei. An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. *Expert Systems with Applications*, 36(6):9625–9633, 2009.
- [20] John Walter Stevens and DD Gemmill. Scheduling a two-machine flowshop with travel times to minimize maximum lateness. *International Journal of Production Research*, 35(1):1–15, 1997.
- [21] Xu Xin, Qiangqiang Jiang, Sihang Li, Shuaiyu Gong, and Kang Chen. Energy-efficient scheduling for a permutation flow shop with variable transportation

-
- time using an improved discrete whale swarm optimization. *Journal of Cleaner Production*, 293:121–126, 2021.
- [22] Shuaipeng Yuan, Tieke Li, and Bailin Wang. A co-evolutionary genetic algorithm for the two-machine flow shop group scheduling problem with job-related blocking and transportation times. *Expert Systems with Applications*, 152:113360, 2020.
- [23] Shuaipeng Yuan, Tieke Li, and Bailin Wang. A discrete differential evolution algorithm for flow shop group scheduling problem with sequence-dependent setup and transportation times. *Journal of Intelligent Manufacturing*, pages 1–13, 2020.
- [24] Seyedeh Sarah Zabihzadeh and Javad Rezaeian. Two meta-heuristic algorithms for flexible flow shop scheduling problem with robotic transportation and release time. *Applied Soft Computing*, 40:319–330, 2016.