Job-Shop Scheduling with Cooperative Transportation Resources

Jean-Philippe Gayon, Philippe Lacomme and Amine Oussama

Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France
j-philippe.gayon@uca.fr, philippe.lacomme@isima.fr and amine.oussama@doctorant.uca.fr

Received xxxx 2025

Abstract. We study a Job Shop Scheduling Problem with Transportation resources (JSPT), which involves the simultaneous scheduling of machines and robots. In constrast to the existing literature, we consider scenarios where a transportation operation may require multiple robots simultaneously, requiring synchronization of resources over time. We formulate this problem using a Mixed Integer Linear Programming (MILP) approach. We show how our problem can be represented by a disjunctive graph. We explain how to describe a feasible solution using an operation sequence vector and a robot assignment vector, and we provide a fast algorithm for evaluating a solution described in this way. We then propose a GRASP×ELS meta-heuristic combined with a local search procedure. Finally, in a numerical study, we adapt instances from literature to align with our problem framework and also create new large-scale instances specifically designed for our problem. For small to medium instances, the meta-heuristic demonstrates competitive performance against exact solutions. For large instances, the metaheuristic outperforms the MILP approach.

Keywords: Job shop, Synchronization, Scheduling, Transportation, Disjunctive graph, Meta-heuristic, Mixed integer linear programming

1 Introduction

A Flexible Manufacturing System (FMS) is a largely automated manufacturing system designed to easily adapt to the type and quantity of the product being manufactured. FMS represent a cornerstone of modern production environments, offering adaptability and efficiency in handling a wide variety of products. A critical component of manufacturing systems, particularly FMS, is material handling and transportation, which has increasingly relied on technologies such as Automated Guided Vehicles (AGVs). The integration of these autonomous mobile robots into manufacturing systems has brought substantial benefits, including increased accuracy, better space management and reduced labor costs. The efficiency and overall performance of modern manufacturing systems, particularly FMS, are profoundly influenced by both production and transportation scheduling decisions. The problem of jointly scheduling production operations on machines and scheduling AGVs for material handling and transportation was extensively studied within the framework of Job-Shop scheduling Problem with Transportation resources (JSPT).

In the Job-Shop scheduling Problem (JSP), we are given several jobs and several machines. Each job consists of a sequence of production operations that must be executed in a predefined order on designated machines. Each production operation is assigned to a specific machine, on which it must be processed without preemption for a fixed duration. The objective is typically to determine a feasible schedule for all job operations that minimizes the makespan. The JSPT extends the JSP by incorporating transportation tasks: after completing a production operation on a machine, a job must be transported to the next machine by a mobile robot. This requires accounting for transportation times, which can influence the overall schedule and makespan. The JSPT is considered one of the most challenging problems as it combines three interconnected subproblems: job scheduling, vehicle assignment, and vehicle routing. The objective is to determine (1) a schedule of production operations on machines, (2) an assignment of robots to transportation operations, and (3) a schedule of transportation operations on robots, all while minimizing the makespan.

Multi-robot systems (MRS) are increasingly employed across diverse domains, including agriculture, mining, disaster relief, space exploration, logistics, and warehouse automation. One of their key applications is cooperative object transport (COT), where multiple robots collaborate to move objects that are too heavy or bulky for a single unit—an approach with strong relevance to real-world scenarios (An et al., 2023). While small loads can be managed by a single robot, larger or heavier loads require multiple robots working in synchronization to ensure safe and stable transportation and prevent damage (see e.g. Mecabotix company (2023)). This challenge is particularly prominent in industrial settings such as factories and warehouses, where robots must operate and coordinate within structured environments (Albert et al., 2025; Chaikovskaia et al., 2025). More recently, research efforts have extended COT to unstructured or dynamic contexts, including space missions, rough terrains, and underwater operations (An et al., 2023). In this paper, we refer to these robots as cooperative robots, meaning robots with the technical capability to regroup and form a more capable unified system when needed. For a review on modular reconfigurable robotics system see Seo et al. (2019).

The main goal of this paper is to investigate a new FMS scheduling problem where transportation is performed by several cooperative robots. We study an extension of the JSPT where multiple robots are required to perform a transportation operation. This aspect adds new layers of complexity to the original problem, as it requires assigning a group of robots to a transportation task rather than a single robot. Additionally, proper synchronization of robots over time is essential to ensure effective cooperation on the same transportation task. As such, the JSPT is a special case of our problem on which exactly one robot is assigned to each transportation task. We call this new variant the Job-Shop Scheduling Problem with Cooperative Transportation Resources (JSPT-C). To the best of our knowledge, this problem has not been explored before.

Our contributions can be summarized as follows. We first introduce the JSPT-C problem and propose a MILP formulation. Then we extend the disjunctive graph model to integrate transportation operations performed by several cooperative robots, allowing it to represent both instances and solutions to the problem. Next, we develop an indirect solution representation using Bierwirth's vector combined with a robot assignment vector. By leveraging properties derived from the longest path, we define suitable neighborhood structures to facilitate a local search methodology, which is incorporated into a GRASPxELS metaheuristic framework. Finally, we validate our model and methods through numerical experiments on both classical JSPT instances from the literature and newly generated JSPT-C instances.

2 Literature review

The job-shop scheduling problem (JSP) is one of the most well-known and extensively studied problems in production task scheduling. The JSP has been proven to be NP-hard, even in simplified cases (Garey et al., 1976). Due to its numerous real-world applications, various solution approaches have been developed over time (see Pinedo (2012) for a recent survey). The disjunctive graph by Roy and Sussman (1964) remains widely used to model this problem. Meanwhile, Bierwirth's (1995) proposal for the job shop remains within the global trend of indirect representation schemes and shows that it is possible to express the job sequencing on machines as a vector by repetitions, which also defines a topological order of nodes in the disjunctive graph. Neighborhood-based metaheuristics have been shown to be effective for solving the JSP. The block approach introduced by Grabowski et al. (1985) has been widely used to derive problem-specific properties and define suitable neighborhoods (e.g., Van Laarhoven et al. (1992); Brucker et al. (1994)). Within this framework, numerous transition operators have been developed for the JSP to generate neighbors for a given solution (see Kuhpfahl and Bierwirth (2016) for a review).

In recent years, increasing attention has been devoted to shop scheduling problems that include transportation aspects (see Berghman et al. (2023) for a survey). These problems consider the transfer of jobs between machines using mobile robots, introducing transportation times into the scheduling process. The transportation aspect can either be integrated into the overall production process by incorporating it into job setup times or treated as separate operations executed on a special machine (robot), effectively doubling the number of job operations. One of the most extensively studied variants is the JSPT, for which the first mathematical formulation was introduced by Raman (1986) under the assumption that vehicles return to the depot station after each transportation operation. Ulusoy and Bilge (1992) later proposed a formulation that removed this constraint. Bilge and Ulusoy (1995) further investigated this problem within the context of a flexible manufacturing system with a job-shop structure. They proposed a heuristic for the joint scheduling of machines and vehicles and introduced the first set of benchmark instances. Subsequent research studied this problem as a JSPT. Knust (2000) studied the JSPT with a single robot and extended the disjunctive graph model to incorporate a single robot that performs all transportation operations. Later, Hurink and Knust (2002, 2005) further explored the JSPT with a single robot, deriving problem-specific properties to define efficient neighborhoods for local search methods. They also proposed a tabu search algorithm and introduced a new set of benchmark instances. Lacomme et al. (2007) extended the disjunctive graph model for the JSPT to accommodate multiple robots. The objective was to determine an assignment of robots to transportation tasks, a schedule for production operations on machines, and a schedule of transportation operations on robots while minimizing the makespan. The block approach, initially introduced for the JSP, has been leveraged for the JSPT with a single robot by Hurink and Knust (2002, 2005) and for the JSPT with multiple robots by Lacomme et al. (2013) to define suitable neighborhoods for these problem variants. Deroussi et al. (2008) proposed a metaheuristic for the JSPT with multiple robots, integrating transportation as part of the overall production process. More recently, several studies have addressed the integrated scheduling of machines and transportation resources using both exact methods and hybrid metaheuristics. Ham (2021) proposed a constraint programming approach for the JSPT, achieving superior performance compared to previous benchmark methods. Fontes and Homayouni (2019) formulated a mixed-integer linear programming model for the joint scheduling of machines and AGVs, in which two interconnected chains of decisions (one for machine operations and another for AGVs) are linked through completion time constraints, yielding promising results with a commercial solver. Building on this line of work, Fontes et al. (2023) introduced a hybrid particle swarm optimization and simulated annealing (PSO-SA) algorithm, augmented by a lower-bounding procedure, which demonstrated strong performance and robustness across a wide range of benchmark instances. In parallel, Yao et al. (2024) proposed problem-specific neighborhood structures tailored to the JSPT, showing that leveraging the structural characteristics of the integrated problem significantly enhances both search efficiency and solution quality. Similarly, Berterottière et al. (2024) tackled the Flexible JSPT by introducing a tabu search metaheuristic equipped with a constant-time move evaluation strategy, enabling the efficient exploration of large neighborhoods. For a comprehensive review of the JSPT, see Nouri et al. (2016).

Synchronizing multiple resources over time to complete a task is a common challenge in related domains, such as vehicle routing problems (see Drexl (2012). for a survey), where synchronization may involve vehicles, drivers, or medical staff. With the advent of new-generation cooperative robots and their potential in improving manufacturing efficiency, researchers have begun addressing challenges related to their integration into manufacturing and logistics environments. For example, Chaikovskaia et al. (2022) proposed a MILP model to optimize the sizing of a fleet of cooperative robots in logistics, while Nguyen et al. (2023) introduced the first formulation of a pickup and delivery problem with cooperative robots.

The problem of jointly scheduling production and transportation operations in manufacturing systems, where transportation tasks are carried out by cooperative robots, remains unexplored. This challenge extends beyond the traditional JSPT, as it requires the simultaneous coordination of multiple resources (i.e., cooperative robots) to perform a transportation task. Each transportation task must be assigned to a suitable number of robots, and their synchronization must be ensured to guarantee the task execution. Our preliminary investigation (Gayon et al., 2024) introduced this issue but considered a simplified scenario, where robots performing multi-robot transportation operations (i.e., tasks requiring the cooperation of multiple robots) were pre-assigned and fixed in advance. As a result, the robot assignment sub-problem was only addressed for transportation operations requiring a single robot. This paper extends that work by generalizing the model to include robot assignment for all transportation operations, including those requiring multiple robots. This generalization significantly increases the problem's complexity, as it involves selecting a subset of robots for each task rather than assigning a single robot, adding a strong combinatorial challenge to the problem.

3 Problem definition

In the JSPT-C, we are given a set of N jobs $\mathcal{J} = \{J_1, .. J_N\}$ that have to be processed on a set of M machines $\mathcal{M} = \{M_1, .. M_M\}$, and each job consists of N_i consecutive production operations (route on the machines). A machine can handle only one job at a time, and each job is processed on a specific machine at most once. Additionally, jobs are transported between machines using a set of NR cooperative robots $\mathcal{R} = \{R_1, .., R_{NR}\}$. Each production operation is preceded by a transportation operation, including the first production operation of the job, as the job must be loaded from an initial depot. However, no transportation operation is considered after the final production operation of the job (no return to the depot). Hence, each job consists of N_i production operations and N_i transportation operations, and the total number of operations per job is $NO_i = 2 \times N_i$. The ordered sequence of all operations of J_i (production and transportation) is denoted $\mathcal{O}_i = (O_{i1}, \dots, O_{i,NO_i})$. An operation O_{ij} corresponds to a production operation when the index j is even and corresponds to a transportation operation when j is odd. Fig. 1 illustrates some notations for a problem with 2 jobs, 2 machines and 3 robots.

A production operation O_{ij} has a dedicated machine denoted M_{ij} and P_{ij} is the uninterrupted processing time of job J_i on M_{ij}. A transportation operation O_{ij} transfers job J_i from machine $M_{i,j-1}$ to machine $M_{i,j+1}$ (with $M_{i,0} = D$). A transportation operation O_{ij} requires simultaneously R_{ij} robots that must be selected among a subset of robots $\mathbb{C}_{ij} \subset \mathcal{R}$. Therefore, instead of assigning a single robot, we need to assign R_{ij} robots to each transportation operation O_{ij}. The subset of robots assigned to each a transportation operation O_{ij} is denoted $\mathcal{R}_{ij}$. The subset $\mathbb{C}_{ij}$ represents the set of robots with the technical capabilities to handle job J_i during transportation operation O_{ij}. If a robot $R_r \in \mathbb{C}_{ij}$, we say that R_r is *compatible* with O_{ij} . We assume that $R_{ij} \leq |\mathbb{C}_{ij}|$ to ensure that O_{ij} can be executed. There are $\binom{|\mathbb{C}_{ij}|}{R_{ij}}$ possible ways to assign robots to O_{ij}, which significantly increases the size of the search space in our problem. To simplify notations, we assume that the duration of a transportation operation O_{ij} does not depend on the number of robots assigned nor on the robots chosen. However, the modeling framework remains the same if we assume that the transportation times depend on the number of robots. Travel times between machines depend solely on distances and remain the same for all robots, with robot cooperation being merely a requirement for executing the transportation operation and exerting no influence on the transportation time. We denote by T_{sd} the loaded travel time from machine s (source) to machine d (destination). It follows that the duration of transportation operation O_{ij} is $T_{M_{i,j-1},M_{i,j+1}}$. Empty travel times between machines must also be considered. We denote by V_{sd} the unloaded travel time from machine s to machine d, and we assume that $V_{sd} \leq T_{sd}$.

The objective is to determine (1) an assignment of the required number of robots to transportation operations, (2) a schedule of machine-operations and (3) a schedule of transport-operations with robots assignment, while minimizing the makespan, i.e. the end of the last operation on the last machine.

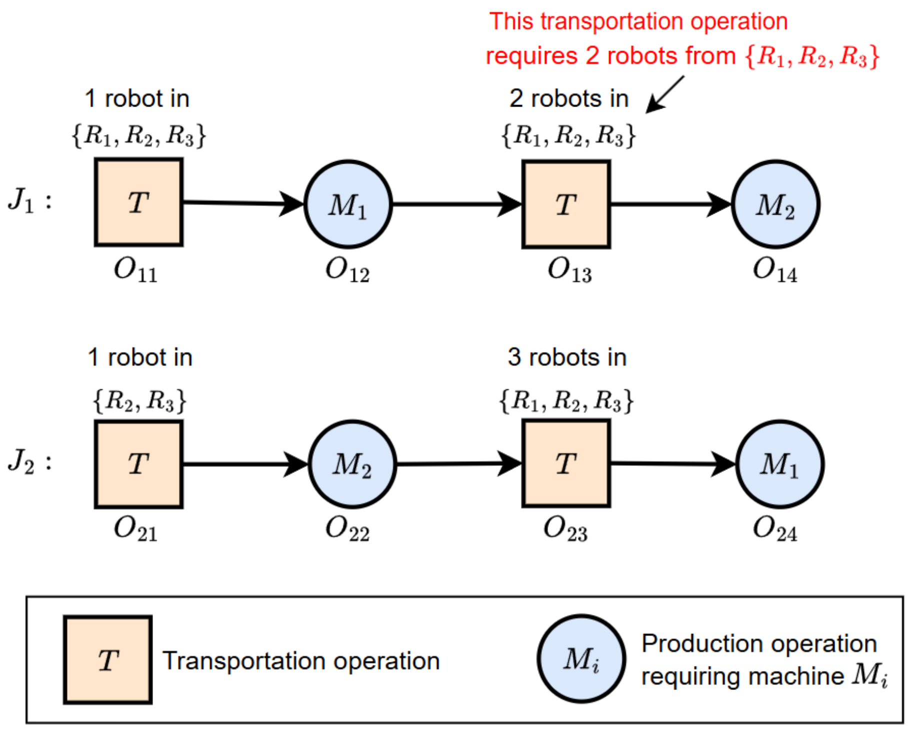

This transportation operation requires 2 robots from $\{R_1, R_2, R_3\}$
1 robot in $\{R_1, R_2, R_3\}$
2 robots in $\{R_1, R_2, R_3\}$
J_1 :
T
M_1
T
M_2
O_{11}
O_{12}
O_{13}
O_{14}
1 robot in $\{R_2, R_3\}$
3 robots in $\{R_1, R_2, R_3\}$
J_2 :
T
M_2
T
M_1
O_{21}
O_{22}
O_{23}
O_{24}
T Transportation operation
M_i Production operation requiring machine M_i

Fig. 1. An illustration of jobs and operations for a problem with 2 jobs, 2 machines and 3 robots

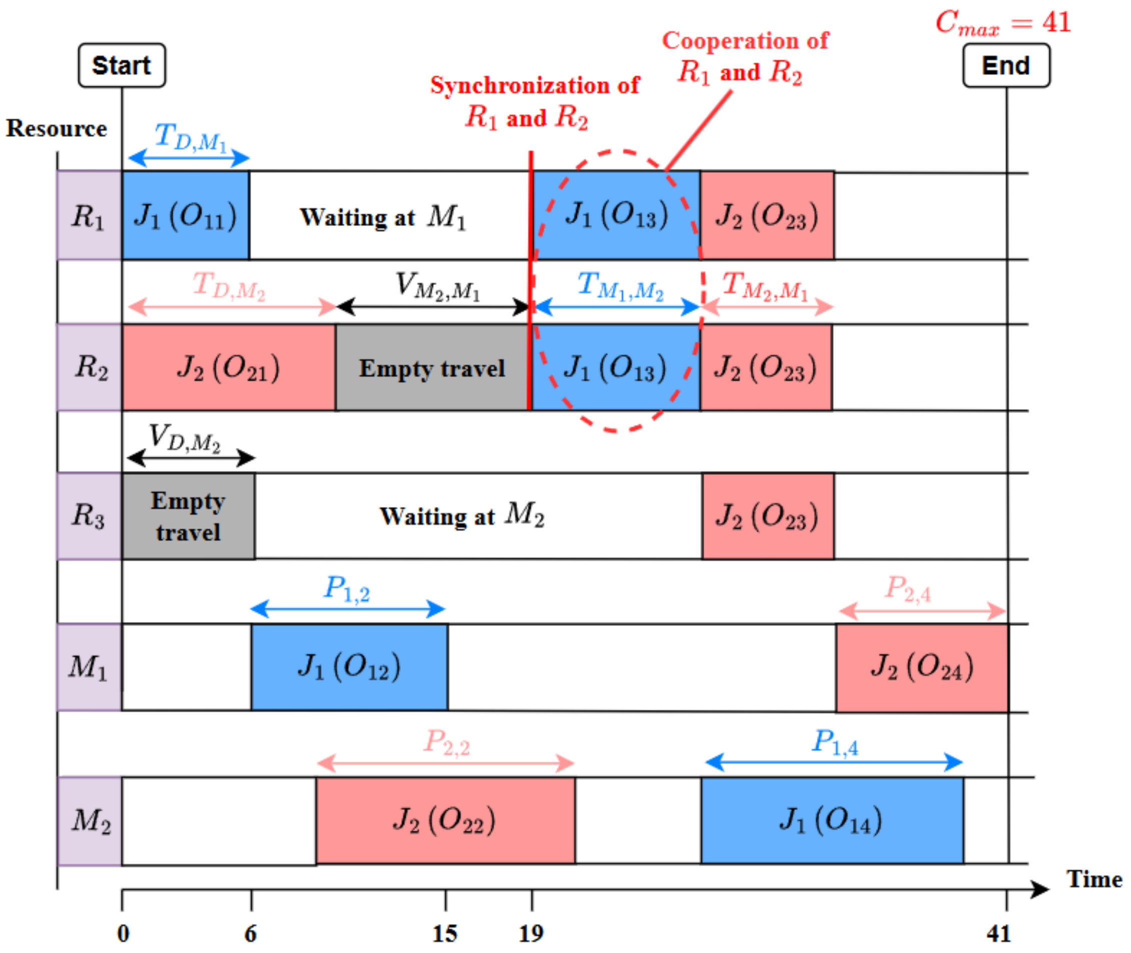


Fig. 2. Gantt chart representing a feasible schedule

There are two main differences between the classical JSPT and the proposed JSPT-C. First, in JSPT-C framework, a transportation operation may require multiple robots simultaneously to be executed. We need to select, for each transportation task, the required number of robots from a predefined set of compatible robots and to ensure their coordinated assignment to the same operation. For example, in Fig. 1, transportation operation O_{13} requires the cooperation of two robots, which must be selected from the set of compatible robots $\{R_1, R_2, R_3\}$.

The second major difference lies in the synchronization requirement associated with robot cooperation. Specifically, when a transportation operation involves multiple robots, it can not start until two conditions are met: (1) the preceding production operation of the corresponding job has been completed, and (2) all assigned robots are simultaneously available at the source machine. To ensure synchronization, robots are routed to the source machine as early as possible, where they wait until both prerequisites are satisfied. Once these conditions are met, the robots must initiate the transportation operation simultaneously. Moreover, they must remain engaged in the operation for the same duration, thereby ensuring a fully coordinated execution of the transport task. Fig. 2 presents a Gantt-chart representation of a feasible schedule for the example introduced in Fig. 1. In this example, transportation operation O_{13} moves job J_1 from machine M_1 to machine M_2 and requires the simultaneous cooperation of two robots chosen from the compatible set $\{R_1, R_2, R_3\}$. In the schedule shown in Fig. 2, robots R_1 and R_2 are assigned to perform this task. The synchronization protocol is illustrated as follows. Upon completing the preceding transportation task O_{11}, robot R_1 reaches the source machine M_1 at $t = 6$ and waits. The first prerequisite—completion of O_{12} the preceding production operation of J_1 which is on M_1—is satisfied at $t = 15$. The second prerequisite—the arrival of the robot R_2 at M_1—is satisfied at $t = 19$. Only when these two conditions hold simultaneously do robots R_1 and R_2 initiate transportation operation O_{13} in perfect synchrony at $t = 19$ and remain jointly engaged for the full duration of the task.

4 Linear formulation of the problem

4.1 Data

N : Number of jobs to schedule.
M : Number of machines.
NR : Number of robots.
NO_i : Total number of operations to schedule for one job ($NO_i = 2 \times N_i$).
L_{ij} : Binary parameter that equals 0 if operation O_{ij} is a machine operation, and equals 1 if O_{ij} is a transportation operation. This parameter is defined for $i = 1..N$ and $j = 1..NO_i$.
M_{ij} : Machine required by production operation O_{ij}. This parameter is defined for $i = 1..N$ and $j = 1..NO_i$ if $L_{ij} = 0$.
T_{sd} : Loaded travel time from machine s to machine d.
V_{sd} : Unloaded (empty) travel time from machine s to machine d.
D_{ij} : Processing time of operation $j = 1..NO_i$ of job $i = 1..N$. It is the processing time on the machine if O_{ij} is a production operation, and it is transportation time from $M_{i,j-1}$ to $M_{i,j+1}$ if O_{ij} is a transportation operation.
R_{ij} : Number of robots required to perform transportation operation O_{ij}. This parameter is defined for $i = 1..N, j = 1..NO_i$ if $L_{ij} = 1$.
C_{ij}^r : Binary parameter that equals 1 if robot R_r is compatible with transportation operation O_{ij}. This parameter is defined for $i = 1..N, j = 1..NO_i,\ r = 1..NR$ if $L_{ij} = 1$.
$C_{ij} = \left(C_{ij}^r\right)_{r=1..NR}$: Binary vector of parameters C_{ij}^r. This parameter is defined for $i = 1..N, j = 1..NO_i$ if $L_{ij} = 1$.
H : A large positive number.

4.2 Decision Variables

st_{ij} : Starting time of operation $j = 1..NO_i$ of job $i = 1..N$.
$C_{\max}$: Makespan (Completion time of all jobs).
b_{ijkp} : Binary variable that equals 1 if operation $\mathrm{O_{ij}}$ is scheduled before operation O_{kp}, and equals 0 otherwise. This variable is defined for $i = 1..N-1,\ j = 1..NO_i, k = 1..N, p = 1..NO_k$ if $i < k$.
a_{ij}^r : Binary variable that equals 1 if transportation operation O_{ij} is assigned to robot R_r, and equals 0 otherwise. This variable is defined for $i = 1..N$ and $j = 1..NO_i$ if $L_{ij} = 1$.
w_{ijkp}^r : Binary variable that equals 1 if both transportation operations O_{ij} and O_{kp} are assigned to robot R_r, and equals 0 otherwise. This variable is defined for $i = 1..N, j = 1..NO_i,\ k = 1..N, p = 1..NO_k$ and $r = 1..NR$ if $i < k$ and $L_{ij} = L_{kp} = 1$.
z_{ijkp}: Binary variable that equals 1 if both transportation operations O_{ij} and O_{kp} are assigned to at least one common robot, and equals 0 otherwise. This variable is defined for $i = 1..N, j = 1..NO_i, k = 1..N$ and $p = 1..NO_k$ if $L_{ij} = L_{kp} = 1$.

4.3 Objective function

Minimize C_{max}
The objective is to minimize the makespan C_{max}, i.e. the time at which all jobs have been processed.

4.4 Constraints

Makespan
For each job $i = 1..N$:

$$st_{i,NO_i} + D_{i,NO_i} \leq C_{max} \tag{1}$$

The makespan C_{max} must be larger than the completion time of the last operation of each job.

Precedence constraints

For each operation O_{ij} ($i = 1..N$ and $j = 1..NO_i$):

$$st_{ij} + D_{ij} \leq st_{i,j+1} \tag{2}$$

The starting time of operation $O_{i,j+1}$ is greater than the finishing time of operation O_{ij}.

Number of robots required for a transportation operation

For each transportation operation O_{ij} ($i = 1..N, j = 1..NO_i$ and $L_{ij} = 1$):

$$\sum_{r=1}^{NR} a_{ij}^r = R_{ij} \tag{3}$$

We must ensure that a transportation operation O_{ij} is assigned to exactly R_{ij} robots.

Robot compatibility constraints for a transportation operation

For each transportation operation O_{ij} ($i = 1..N, j = 1..NO_i$ and $L_{ij} = 1$) and for each robot $r = 1..NR$ incompatible with O_{ij} ($C_{ij}^r = 0$):

$$a_{ij}^r = 0 \tag{4}$$

For each transportation operation O_{ij}, we must ensure that it is not assigned to robots that are not compatible with it.

Disjunction between production operations processed on the same machine

For each pair of distinct production operations O_{ij} and O_{kp} processed on the same machine ($i = 1..N-1, j = 1..NO_i, k = 1..N, p = 1..NO_k$ such that $i < k$, $L_{ij} = L_{kp} = 0$ and $M_{ij} = M_{kp}$):

$$st_{ij} + D_{ij} \leq st_{kp} + H.\left(1 - b_{ijkp}\right) \tag{5}$$

$$st_{kp} + D_{kp} \leq st_{ij} + H.b_{ijkp} \tag{6}$$

Two production operations O_{ij} and O_{kp} $\left(L_{ij} = L_{kp} = 0\right)$ that require the same machine ($M_{ij} = M_{kp}$) cannot be executed simultaneously. A disjunction between the two operations implies that either O_{ij} is scheduled first, followed by O_{kp}, and in this case we have $st_{ij} + D_{ij} \leq st_{kp}$, or O_{kp} is scheduled first, followed by O_{ij}, and we have $st_{kp} + D_{kp} \leq st_{ij}$.

Disjunction between transportation operations that require at least one robot in common

For each pair of distinct transportation operations O_{ij} and O_{kp} ($i = 1..N-1,\ j = 1..NO_i,\ k = 1..N, p = 1..NO_k$ such that $i < k$ and $L_{ij} = L_{kp} = 1$) and for each robot $r = 1..NR$:

$$a_{ij}^r + a_{kp}^r \leq 1 + w_{ijkp}^r \tag{7}$$

$$w_{ijkp}^r \leq a_{ij}^r \tag{8}$$

$$w_{ijkp}^r \leq a_{kp}^r \tag{9}$$

$$\sum_{r=1}^{NR} w_{ijkp}^r \leq H.z_{ijkp} \tag{10}$$

$$z_{ijkp} \leq \sum_{r=1}^{NR} w_{ijkp}^r \tag{11}$$

Constraints (7), (8) and (9) ensure that two transportation operations O_{ij} and O_{kp} ($L_{ij} = L_{kp} = 1$) are assigned to the same robot R_r ($w_{ijkp}^r = 1$) if and only if $a_{ij}^r = a_{kp}^r = 1$. Constraints (10) and (11) ensure that O_{ij} and O_{kp} share at least one robot ($z_{ijkp} = 1$) if and only if there exists a robot R_r such that $w_{ijkp}^r = 1$ (i.e. $\sum_{r=1}^{NR} w_{ijkp}^r > 0$).

For each pair of distinct transportation operations O_{ij} and O_{kp} ($i = 1..N-1,\ j = 1..NO_i,\ k = 1..N, p = 1..NO_k$ such that $i < k$ and $L_{ij} = L_{kp} = 1$)

$$st_{ij} + T_{M_{i,j-1},M_{i,j+1}} + V_{M_{i,j+1},M_{k,p-1}} \leq st_{kp} + H.\left(1 - b_{ijkp}\right) + H.\left(1 - z_{ijkp}\right) \tag{12}$$

$$st_{kp} + T_{M_{k,p-1},M_{k,p+1}} + V_{M_{k,p+1},M_{i,j-1}} \leq st_{ij} + H.b_{ijkp} + H.\left(1 - z_{ijkp}\right) \tag{13}$$

Constraints (12) and (13) mean that if two transportation operations O_{ij} and O_{kp} share at least one robot ($w_{ijkp} = 1$), then they cannot be executed simultaneously.

5 Solution representation and evaluation

We assume that we are given a feasible assignment of transportation operations to robots. We show how to extend the disjunctive graph for the JSPT (Hurink and Knust, 2005; Lacomme et al., 2013) to include transportation operations that require several robots simultaneously. Then, we show how to indirectly represent a feasible solution with an operations sequence vector and a robot assignment vector. Finally, we present a fast algorithm to evaluate a solution.

5.1 Disjunctive graph modeling

This problem can be represented using a disjunctive graph, where the vertices correspond to production and transportation operations (see Fig. 3 for an example). Two dummy vertices, 0 and *, are introduced to represent the start and end of the schedule, respectively. Each operation O_{ij} is connected to its subsequent operation $O_{i,j+1}$ by a conjunctive (solid) arc, reflecting precedence constraints and is weighted by duration D_{ij}. For each job J_i, a dummy conjunctive arc connects vertex 0 to its first operation $O_{i,1}$, with a weight of 0, and another connects its last operation O_{i,NO_i} to vertex *, weighted by D_{i,NO_i}. If two operations O_{ij} and O_{kp} require at least one resource in common (machine or robot), they cannot be executed simultaneously. Therefore, they are connected by two disjunctive (dashed) arcs in opposite directions. These arcs represent the choice of which of the two operations is scheduled first on the shared resource. When O_{ij} and O_{kp} are production operations that require the same machine, the weight of the disjunctive arc from O_{ij} to O_{kp} is D_{ij}. When O_{ij} and O_{kp} are transportation operations sharing at least one robot, the weight of the disjunctive arc from O_{ij} to O_{kp} is $D_{ij} + V_{M_{i,j+1},M_{k,p-1}}$, representing the time required for the shared robots to complete O_{ij} and travel to the source machine for O_{kp}. Operations sharing the same resource (machine or robot) form cliques of double arcs in this graph. Consequently, a transportation operation involving multiple robots may belong to several cliques. Disjunctive arcs linking two transportation operations of the same job that share common robots are not represented, as they are unnecessary due to the presence of conjunctive arcs.

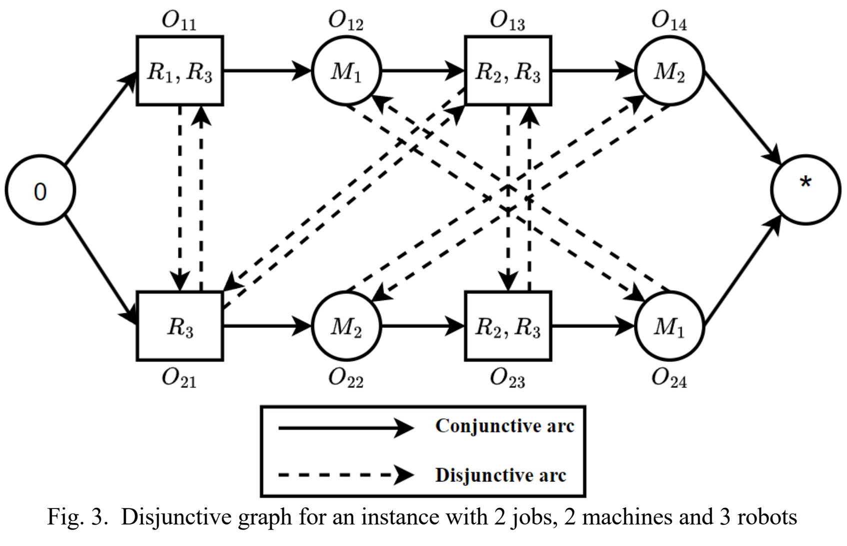


Fig. 3. Disjunctive graph for an instance with 2 jobs, 2 machines and 3 robots

We illustrate the disjunctive graph in Fig. 3, which involves 2 jobs, 2 machines, and 3 robots, with 2 production operations per job. In this figure, the selected robots to perform each transportation operation are specified. For instance, transportation operation O_{11} requires robots R_1 and R_3. To obtain a feasible solution, we must determine the sequence in which operations sharing a common resource will be performed. In the disjunctive graph, this is achieved by selecting one disjunctive arc from each pair of disjunctive arcs linking two operations that require resources in common and deleting the other. Selecting the disjunctive arc from O_{ij} to O_{kp} represents the decision of scheduling O_{ij} before O_{kp}. After applying this process to all pairs of operations requiring the same resource, the resulting graph is referred to as a fixed disjunctive graph and it represents a feasible solution if and only if it is acyclic.

We illustrate in Fig. 4 an acyclic fixed disjunctive graph representing a feasible schedule for the example in Fig. 3. For instance, O_{12} is scheduled before O_{24} on machine M_1. Similarly, O_{11} is scheduled before O_{21} on robot R_1, and O_{13} is scheduled before O_{23} on both robots R_2 and R_3.

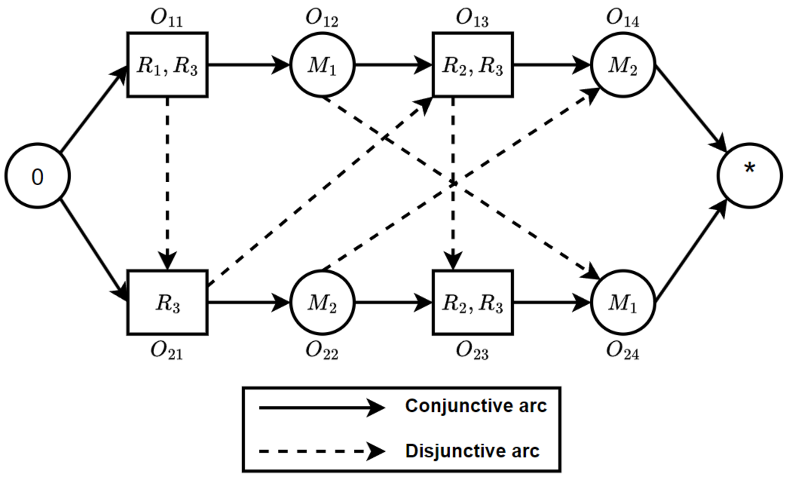


Fig. 4. Acyclic fixed disjunctive graph

After assigning weights to the arcs, the earliest starting times of operations are computed on the acyclic fixed disjunctive graph using a longest path algorithm from vertex 0 to vertex * to determine a semi-active solution. Fig. 5 illustrates one semi-active solution where the makespan $C_{max} = 60$ which corresponds to the length of the critical path of the acyclic fixed disjunctive graph. The disjunctive arc from O_{11} to O_{21}, which appears on the critical path, indicates that the schedule was delayed because O_{21}, which requires robot R_3, had to wait for the completion of O_{11} and the subsequent empty travel of R_3 from M_1 to the depot. The evaluation of the graph permits us to compute a semi-active solution that is left shifted and that can be represented using a Gantt chart (see Fig. 6 for an example). One of the challenging aspects of solving a problem modeled by a disjunctive graph is efficiently finding acyclic disjunctive graphs which are the only ones that describe feasible solutions.

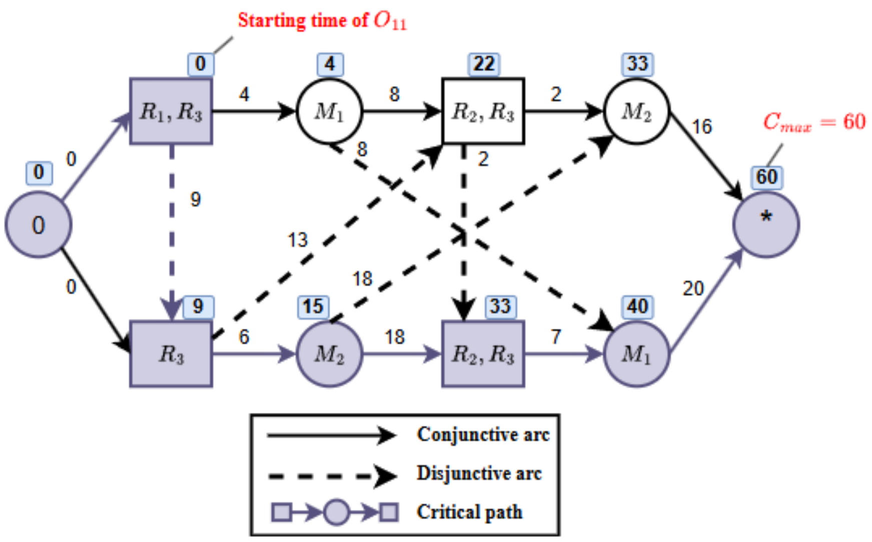


Fig. 5. Evaluating a feasible solution

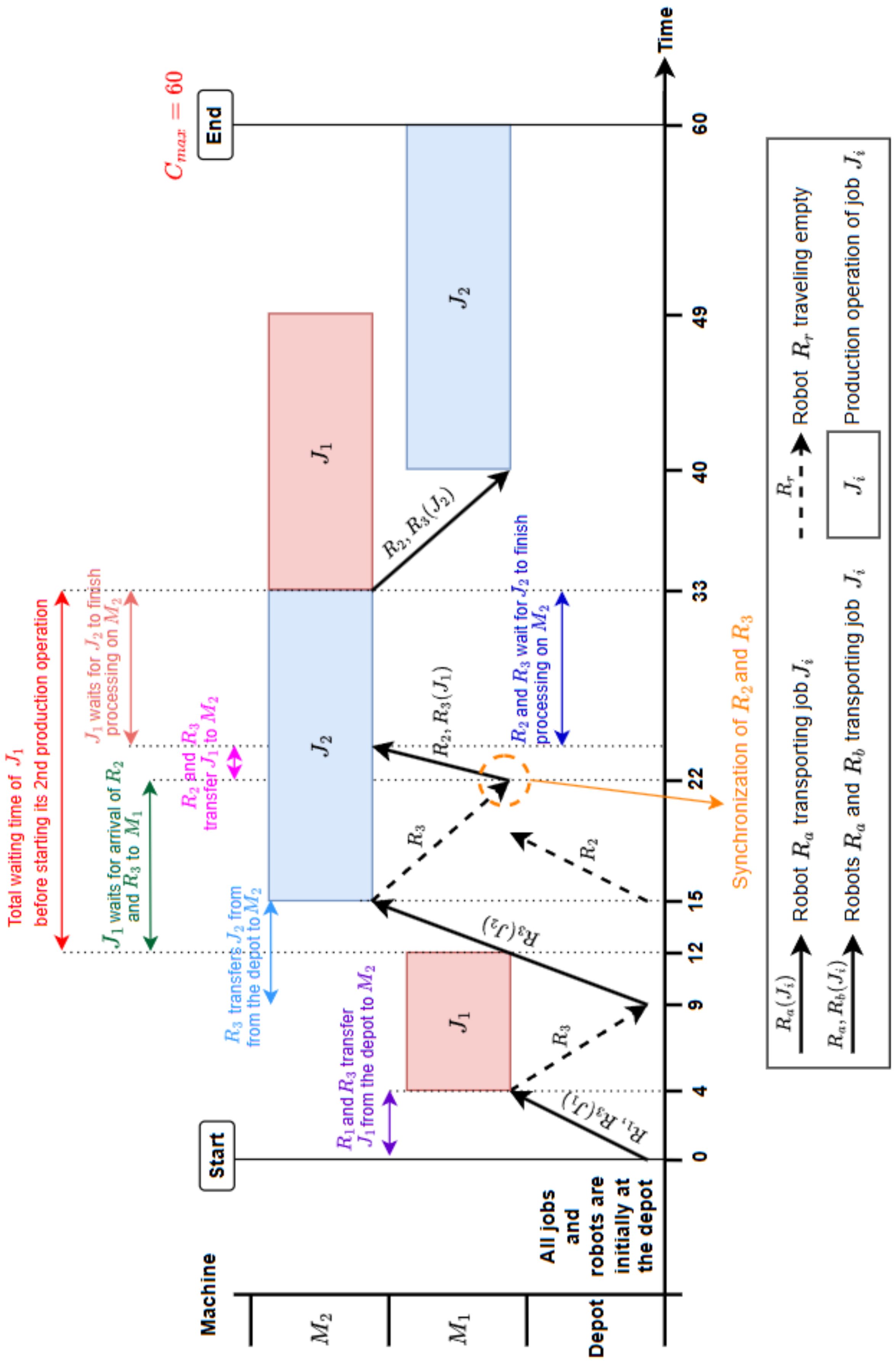


Fig. 6. Another Gantt chart representing a feasible schedule

5.2 Indirect vector representation of a solution

The concept of vector by repetitions was initially used to represent feasible solutions for the JSP (Bierwirth, 1995) and was later extended for the JSPT (Lacomme et al, 2007). In this section, we demonstrate how this concept can be adapted to represent a feasible solution for the JSPT-C. The vector by repetitions (Bierwirth, 1995) and the mapping function that links any vector to one acyclic fixed disjunctive graph permits us to investigate only the set of acyclic fixed disjunctive graphs (feasible solutions).

The Bierwirth's vector representation can be extended considering two vectors. The first one concerns the robot assignment for transportation operations and the second one concerns the sequence of all job operations:

- ***Operations assignment (OA) vector***. The vector OA contains the selected robots to perform each transportation operation, starting from the first transport operation of J_1 to the last transport operation of J_N. A robot R_r in the vector is represented by the number r. For instance, the assignment of robots used in Fig. 3 can be represented by the OA vector shown in Fig. 7. Note that, in terms of implementation, we use a vector of NR binary variables for each transportation operation to indicate whether a robot is assigned to it.

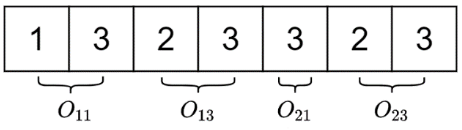


Fig. 7. An example of OA vector

- ***Operations sequence (OS) vector.*** The vector OS is a vector by repetitions (Bierwirth, 1995). It consists of a permutation with repetition (multi-permutation) of the vector $(1, \dots, N)$, where each job J_i appears exactly $NO_i = 2N_i$ times (NO_i is the total number of operations of J_i). Hence an OS vector has exactly $\sum_{i=1}^{N} NO_i$ components. The vector is read from left to right and represents the sequence in which operations are scheduled. The j-th occurrence of number i in the vector OS refers to O_{ij} (the j-th operation of job J_i). If j is even, then O_{ij} is a production operation requiring machine M_{ij}. Conversely, if j is odd then O_{ij} is a transportation operation assigned to the subset of robots specified in the OA vector.

Given a pair (OA, OS), we can build a feasible schedule by considering the three following rules when reading the OS vector:

- **Rule 1:** The sequence of occurrences of the same number represents precedence constraints (conjunctive arcs).
- **Rule 2:** If O_{ij} and O_{kp} are two production operations requiring the same machine ($M_{ij} = M_{kp} = M_\alpha$), and O_{ij} precedes O_{kp} in the OS vector without any intermediate production operation requiring M_α, then we have a disjunctive arc from O_{ij} to O_{kp}, which means O_{ij} is scheduled before O_{kp} on M_α.
- **Rule 3:** If O_{ij} and O_{kp} are two transportation operations of different jobs sharing a robot R_r ($R_r \in \mathcal{R}_{ij} \cap \mathcal{R}_{kp}$), and O_{ij} precedes O_{kp} in the OS vector without any intermediate transportation operation requiring R_r, then we have a disjunctive arc from O_{ij} to O_{kp}, which means O_{ij} is scheduled before O_{kp} on R_r

We illustrate in Fig. 8 the pair of vectors (OA, OS) that permit us to define the sequence of operations presented in Fig. 9 considering the 3 rules introduced early.

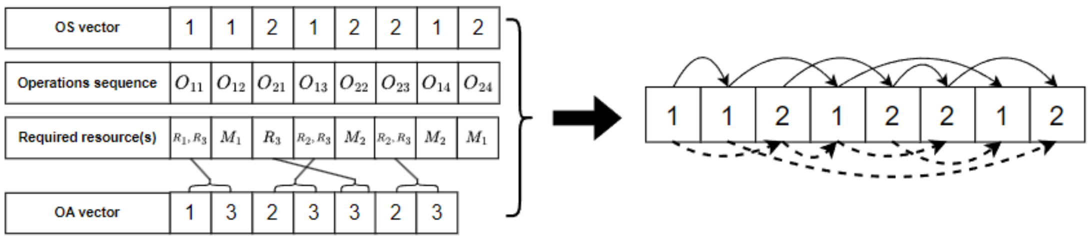


Fig. 8. (OA,OS) vectors representing an acyclic fixed disjunctive graph

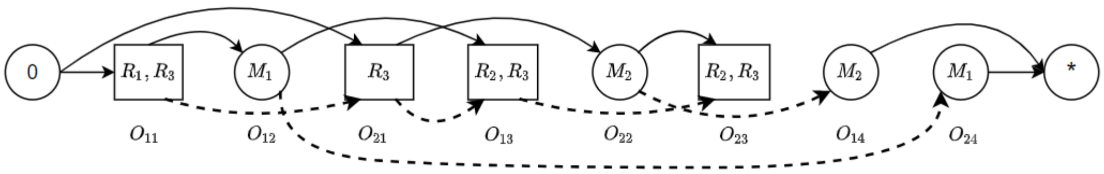


Fig. 9. Acyclic fixed disjunctive graph represented by (OA,OS) vectors

With respect to the robot assignments provided by the OA vector, and by using the 3 rules on OS vector, the (OA, OS) vectors represent indirectly a fixed disjunctive graph. Since all the arcs go from left to right, it is easy to see that the resulting graph is acyclic. Consequently, by construction, the pair (OA, OS) always describes (indirectly) a feasible solution. Furthermore, the OS vector itself is a topological order for the associated acyclic fixed disjunctive graph. This solution representation establishes a mapping between the space of all possible (OA, OS) vectors and the space of acyclic fixed disjunctive graphs (feasible solutions). Consequently, instead of operating directly on graphs, we work exclusively with vectors, with the guarantee that each pair of vectors describes a feasible solution. With this approach, the search space is confined to the set of all possible (OA, OS) vectors.

5.3 Solution evaluation

Given a pair of (OA, OS) vectors, the solution is evaluated using a longest-path algorithm that uses the topological order defined by OS and the 3 rules introduced early to identify and relax the conjunctive and disjunctive arcs. The evaluation algorithm, shown in Fig. 10, begins by initializing operation start times and their predecessors in the longest path. Each element of OS represents an operation: at iteration u, the current job i is the u-th element of OS, and the operation number j is determined by the number of times i has appeared in OS so far. The starting time of operation O_{ij} is updated to the earliest time that satisfies precedence and disjunctive constraints.

For precedence constraints, if O_{ij} is not the first operation of job i $(j > 1)$ then O_{ij} cannot start before $O_{i,j-1}$ finishes, which is ensured by relaxing the conjunctive arc from $O_{i,j-1}$ to O_{ij}.

For disjunctive constraints, we use an array lp (and lt, respectively) to track the last production operation that used each machine (and the last transport operation that used each robot, respectively). If O_{ij} is not the first production operation that uses the corresponding machine M_{ij}, then O_{ij} cannot start before the preceding operation on M_{ij}, denoted O_{kp}. This constraint is handled by relaxing the disjunctive arc from O_{kp} to O_{ij}. Similarly, if O_{ij} is a transport operation, the R_{ij} robots required by O_{ij}, as specified in OA vector, must all be available. Each required robot R_r must complete its previous transport operation O_{kp} and any subsequent unloaded travel to $M_{i,j-1}$, which is enforced by relaxing the disjunctive arc (O_{kp}, O_{ij}). After all arcs relaxed, the lp or lt arrays are updated.

Finally, the starting times of the final operations for each job i are calculated by relaxing the arcs $\left(O_{i,NO_i}, *\right)$. The Makespan, representing the solution's fitness, is the finishing time of the last job, equivalent to the length of the longest path.

Let us note that unlike classical Job-Shop Scheduling Problems with Transportation Resources, in this problem each transportation operation may require relaxing up to R_{ij} disjunctive arcs, instead of just one.

Time complexity of solution evaluation. If a topological order is provided for a Directed Acyclic Graph $G = (V, E)$ the time complexity of the longest path algorithm on G is $O(|E|)$. In our case, the number of arcs in the fixed disjunctive graph depends on the instance. We consider the worst-case scenario, where the number of arcs is maximized. For each job, the number of conjunctive arcs is the number of operations per job, so the total number of conjunctive arcs in the graph is $2 \times N \times M$. On each machine, we schedule sequentially N jobs, thus a disjunctive arc links each pair of consecutive production operations on this machine, which means we have $N - 1$ disjunctive arcs for each machine, hence a total of $(N - 1) \times M$ machine disjunctive arcs. For each transport operation, there are at most $R_{ij} \leq NR$ outgoing disjunctive arcs (one for each robot). With $N \times M$ transport operations in total, the upper bound for these arcs is $N \times M \times NR$. Finally, the time complexity is in $O(N \times M \times NR)$.

Algorithm 1: Evaluate

```
Input   : $S = (OS, OA)$ : A complete selection
Output: $C_{max}$ : Makespan
Function $EVALUATE(S)$:
    // Initialization
    $C_{max} \leftarrow 0$
    for $i \leftarrow 1$ to $N$ do
        for $j \leftarrow 1$ to $M$ // For every operation operation $O_{ij}$
         do
            $st(i,j) \leftarrow 0$      // Initialize starting time of operation $O_{ij}$
            $P(i,j) \leftarrow (0,0)$ // Initialize father of $O_{ij}$ for longest path algorithm on $G(S)$
        end for
        $np(i) \leftarrow 0$ // Initialize the number of times job $i$ is encouneterd in $OS$ vector
    end for
    $P(*) \leftarrow (0,0)$ // Initialize father of dummy node *
    for $i \leftarrow 1$ to $M$ // For every machine $M_i$
     do
        $lm(i) \leftarrow (0,0)$ // Initialize the last machine operation processed on $M_i$
    end for
    for $r \leftarrow 1$ to $NR$ // For every robot $R_r$
     do
        $lt(r) \leftarrow (0,0)$ // Initialize the last transport operation that used $R_r$
    end for
    // Longest path algorithm on $G(S)$ using topological order given by $OS$ vector
    $NTO \leftarrow \sum_{i=1}^{n_i}(2 \times n_i)$ // Total number of operations
    for $u \leftarrow 1$ to $NTO$// For every operation in $OS$
    $i \leftarrow OS(u)$           // Job of operation $u$
    $np(i) \leftarrow np(i) + 1$ // Update count of job $i$ encounters
    $j \leftarrow np(i)$            // Job operation number
    // Precedence constraints
    if $j > 1$ // If $j$ is not the first operation of job $i$
     then
        if $st(i,j-1) + D(i,j-1) > st(i,j)$ // Conjunctive arc $(O_{i,j-1}, O_{i,j})$ relaxation
         then
            $st(i,j) \leftarrow st(i,j-1) + D(i,j-1)$ // Update starting time of $O_{ij}$
            $P(i,j) \leftarrow (i,j-1)$                     // Update father of $O_{ij}$
        end if
    end if
    // Disjunctive constraints
    if $O_{ij}$ is a machine operation then
        $machine \leftarrow M_{i,j}$ // Corresponding machine
        $(k,p) \leftarrow lm(machine)$ // $O_{kp}$ is the last operation processed on the machine before $O_{ij}$
        if $(k,p) \neq (0,0)$ // If $O_{ij}$ is not the first operation processed on the machine
         then
            if $st(k,p) + D(k,p) > st(k,p)$ // Disjunctive arc $(O_{k,p}, O_{i,j})$ relaxation
             then
                $st(i,j) \leftarrow st(k,p) + D(i,j)$ // Update starting time of $O_{ij}$
                $P(i,j) \leftarrow (k,p)$                 // Update father of $O_{ij}$
            end if
        end if
        $lm(machine) \leftarrow (i,j)$ // Update the last operation processed on the machine
    else
        for $robot \in OA(i,j)$// For every robot assigned to transport operation $O_{ij}$
         do
            $(k,p) \leftarrow lt(robot)$ // $O_{kp}$ is the last operation that used the robot before $O_{ij}$
            if $(k,p) \neq (0,0)$ // If $O_{ij}$ is not the first operation that used the robot
             then
                if $st(k,p) + D(k,p) > st(k,p)$ // Disjunctive arc $(O_{k,p}, O_{i,j})$ relaxation for the robot
                 then
                    $st(i,j) \leftarrow st(k,p) + D(i,j)$ // Update starting time of $O_{ij}$
                    $P(i,j) \leftarrow (k,p)$                 // Update father of $O_{ij}$
                end if
            end if
            $lt(robot) \leftarrow (i,j)$ // Update the last operation that used the machine
        end for
    end if
     do

    end for
    $i \leftarrow 1$ to $N$ // Calculate Makespan by relaxing arcs $(O_{i,NO}, *)$
    if $st(i, 2 \times n_i) + D(i, 2 \times n_i) > C_{max}$ then
        $C_{max} \leftarrow st(i, 2 \times n_i) + D(i, 2 \times n_i)$ // Update Makespan
        $P(*) \leftarrow (i, 2 \times n_i)$ // Update father of dummy node *
    end if
    return $C_{max}$
end
```

Fig. 10. Efficient solution evaluation

6 Iterative search based metaheuristic

In all this section we assume that we are given a feasible solution S represented by an acyclic fixed disjunctive graph $G(S)$ and indirectly by a pair (OA, OS). We adapt the block approach of (Grabowski, 1986) to define neighborhoods for the solution, and we introduce an iterative search based metaheuristic that leverages the indirect representation to explore only the set of acyclic fixed disjunctive graphs, which model exclusively feasible solutions. The efficiency lies in the proper definition of an efficient local search procedure and an efficient search space exploration.

6.1 Definitions

A sequence of successive nodes in a critical path of $G(S)$ is called a:

- ***Machine-block*** if the sequence contains at least two nodes and represents a maximal number of consecutive production operations processed on the same machine; and
- ***Robot-block*** if the sequence contains at least two nodes and represents a maximal number of transportation operations such that two successive operations (O_{ij}, O_{kp}) in the sequence share at least one robot.

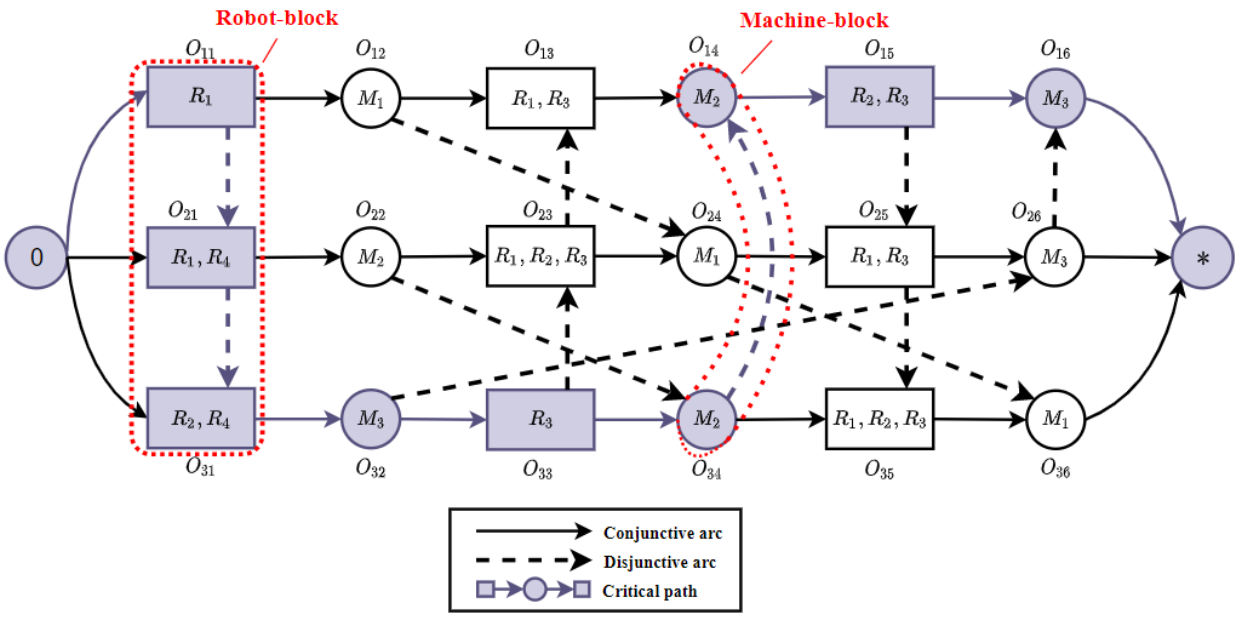


Fig. 11. A feasible solution and its critical path

We illustrate the definitions using an acyclic fixed disjunctive graph for a problem with 3 jobs, 3 machines, and 4 robots, as shown in Fig. 11. For example, the sequence of production operations O_{34}, O_{14} forms a machine-block since both operations are consecutive on the critical path and require machine M_2. Similarly, the sequence of transport operations O_{11}, O_{21}, O_{31} forms a robot-block because: 1) they are consecutive on the critical path; 2) O_{11} and O_{21} share robot R_1, while O_{21} and O_{31} share robot R_4. It is worth noting that although O_{11} and O_{31} belong to the same robot block, they do not share any robot. This demonstrates the generalization of the robot-block concept: O_{21} shares R_1 with O_{11} while it shares R_4 with O_{31} which created a block on the critical path.

6.2 Local search operators

To define neighborhoods for the job-shop scheduling problem, various transition operators representing elementary schedule modifications have been proposed (see Kuhpfahl and Bierwirth (2016) for an overview). It is also well-known that in order to improve a solution, such modifications should be limited to operations that are on the critical path. The blocks approach is a way to identify specific parts of the solution where modifications can potentially be applied to achieve a strictly better solution. For a machine-block, we can schedule the first production operation of the block after some other operation of the same block, or we can schedule the last production operation of the block before some other operation of the same block (Brucker et al., 1994). For a robot-block, we can also interchange the processing order of two consecutive transportation operations due to empty-travel times (Hurink and Knust, 2002). In our problem, we can also change the robot assignments for two consecutive transportation operations in a robot-block such that the conflicts on the shared robots are resolved, however this is not always possible if robot compatibility constraints are considered. In the iterative search procedure, we will focus only on these specific changes to define transition operators.

We use two transition operators to generate neighboring solutions:

- ***Critical Transpose (CT) operator***. Originally introduced by Van Laarhoven et al. (1992) for the classical job-shop problem, this operator, also known as interchange or swap, involves reversing a disjunctive arc between two adjacent operations within a block on the critical path. In our problem, it can be applied to both machine and robot blocks. For machine-blocks, the CT operator can reduce the makespan only when applied to the first or last disjunctive arc of the block. While it is less efficient in machine-blocks—since applying it to an internal arc cannot reduce the makespan—it has the notable advantage of always producing a feasible solution. The CT operator can be directly applied to the OS vector representing the solution: To swap two adjacent operations on some block of the critical path of S, we use the ***Call_Swap*** procedure. This procedure locates the positions of the two operations in the OS vector and swaps their corresponding values, producing a new vector that represents a neighboring solution.

- ***Critical Conflict Resolve (CCR) operator.*** We define a new transition operator that can be applied to a robot-block, which involves a sequence of robot assignment changing for two adjacent transportation operations on the block such that all robot conflicts are resolved (i.e. they no longer share any robots). This results in deleting the disjunctive arc between the two operations which would produce a new acyclic fixed disjunctive graph that does not contain the previous critical path and therefore may potentially contain a new shorter critical path. Due to compatibility constraints, this operator is not always possible and depends on the number of available compatible robots. For two adjacent transportation operations O_{ij} and O_{kp} within a robot-block, let $\mathbb{C}_{ij}^{-} = \mathbb{C}_{ij} \setminus (\mathcal{R}_{ij} \cup \mathcal{R}_{kp})$ and $\mathbb{C}_{kp}^{-} = \mathbb{C}_{kp} \setminus (\mathcal{R}_{ij} \cup \mathcal{R}_{kp})$. The set $\mathbb{C}_{ij}^{-}$ (respectively $\mathbb{C}_{kp}^{-}$) represents the robots compatible with O_{ij} (respectively O_{kp}) that are not currently assigned to either O_{ij} or O_{kp}. For example, if O_{ij} and O_{kp} share a robot R_r, a robot from $\mathbb{C}_{ij}^{-}$ can be assigned for O_{ij} instead of R_r, which solves the conflict on R_r between the two operations. Since O_{ij} and O_{kp} may share several robots simultaneously, we can easily see that the sufficient and necessary condition to be able to apply the CCR operator on the disjunctive arc between O_{ij} and O_{kp} is to have $|\mathcal{R}_{ij} \cap \mathcal{R}_{kp}| \leq |\mathbb{C}_{ij}^{-} \cup \mathbb{C}_{kp}^{-}|$. This condition means that there is a sufficient number of robots compatible with one/or both operations and not used in neither of them, to replace in either operation all the shared robots such that all conflicts between the operations are resolved.

To check if it possible to solve all the robot conflicts between two consecutive transportation operations on some robot-block, we use the $Call\_Check\_Resolve$ procedure (see Fig. 12) which iterates through all robots and by referring to the OA vector calculates NSR the number of shared robots between the two transportation operations ($NSR = |\mathcal{R}_{ij} \cap \mathcal{R}_{kp}|$), and NCR the number of available compatible robots that can replace the shared robots ($NCR = |\mathbb{C}_{ij}^{-} \cup \mathbb{C}_{kp}^{-}|$), and returns a Boolean res that indicates if the condition is satisfied ($NCR \geq NSR$). The function also returns the set P_0 of shared robots between the two operations ($P_0 = \mathcal{R}_{ij} \cap \mathcal{R}_{kp}$), the set P_1 of available robots compatible exclusively with O_{ij}, the set P_2 of of available robots compatible exclusively with O_{kp}, and the set P_3 of available robots that are compatible with both operations. Available means it is not assigned to neither operation.

```
Algorithm 2: Call_Check_Resolve
Input  : $O_{ij}$, $O_{kp}$: Two adjacent transport operations of a robot-block
Output: $res$ : Boolean indicating if the condition is satisfied
         $P_0$: Set of shared robots
         $P_1$: Set of robots compatible with $O_{ij}$ but not with $O_{kp}$ and are not assigned to $O_{ij}$
         $P_2$: Set of robots compatible with $O_{kp}$ but not with $O_{ij}$ and are not assigned to $O_{kp}$
         $P_3$: Set of robots compatible with $O_{ij}$ and $O_{kp}$ and are not assigned to any of them
Function Call_Check_Resolve($O_{ij}$, $O_{kp}$):
    // Initialization
    $res \leftarrow False$
    $NSR \leftarrow 0$ // Number of shared robots between $O_{ij}$ and $O_{kp}$
    $NCR \leftarrow 0$ // Number of compatible robots with either operations unassigned to any of them
    $P_0 \leftarrow \emptyset$
    $P_1 \leftarrow \emptyset$
    $P_2 \leftarrow \emptyset$
    $P_3 \leftarrow \emptyset$
    for $R_r \in \mathcal{R}$ do
        if $R_r$ is assigned to both $O_{ij}$ and $O_{kp}$ then
            $P_0 \leftarrow P_0 \cup \{R_r\}$ // Add $R_r$ to the set of shared robots
            $NSR \leftarrow NSR + 1$ // Increment the number of shared robots
        else
            if $R_r$ is not assigned to either $O_{ij}$ or $O_{kp}$ then
                if $R_r$ is compatible with $O_{ij}$ but not with $O_{kp}$ then
                    $P_1 \leftarrow P_1 \cup \{R_r\}$ // Add $R_r$ to the set of robots compatible exclusively with $O_{ij}$
                    $NCR \leftarrow NCR + 1$ // Increment the number of available compatible robots
                end if
                if $R_r$ is compatible with $O_{kp}$ but not with $O_{ij}$ then
                    $P_2 \leftarrow P_2 \cup \{R_r\}$ // Add $R_r$ to the set of robots compatible exclusively with $O_{kp}$
                    $NCR \leftarrow NCR + 1$ // Increment the number of available compatible robots
                end if
                if $R_r$ is compatible with both $O_{ij}$ and $O_{kp}$ then
                    $P_3 \leftarrow P_3 \cup \{R_r\}$ // Add $R_r$ to the set of available robots compatible with both operations
                    $NCR \leftarrow NCR + 1$ // Increment the number of available compatible robots
                end if
            end if
        end if
    end for
    // Check if it is possible to apply $CCR$ operator on the two operations
    if $NCR \geq NSR$ then
        $res \leftarrow True$
    end if
    return $res, P_0, P_1, P_2, P_3$
end
```

Fig. 12. *Call_Check_Resolve* pseudo-algorithm

If the condition is satisfied for a pair of two consecutive transport operations O_{ij} and O_{kp} on a robot-block, we apply the CCR operator on the two operations directly on the OA vector by updating the robot assignments of one or both operations. This is done by the ***Call_Resolve*** procedure which selects at each iteration a robot that is not assigned to neither operation to replace a shared robot in either operation based on its compatibilities. It updates then the OA vector accordingly and the process is repeated until all conflicts are resolved, which means the total number of iterations is $NSR = |P_0|$. The pseudo-algorithm is given in Fig. 13.

```
Algorithm 3: Call_Resolve
Input  : $O_{ij}, O_{kp}$
         $P_0, P_1, P_2, P_3$
Function Call_Resolve($O_{ij}, O_{kp}, P_0, P_1, P_2, P_3$):
    for $R_c \in P_0$ do
        Select randomly $R_r \in P_1 \cup P_2 \cup P_3$
        if $R_r \in P_1$ then
            Replace $R_c$ by $R_r$ in the list of robots assigned to $O_{ij}$
            $P_1 \leftarrow P_1 \setminus \{R_r\}$
        else
            if $R_r \in P_2$ then
                Replace $R_c$ by $R_r$ in the list of robots assigned to $O_{kp}$
                $P_2 \leftarrow P_2 \setminus \{R_r\}$
            else
                Replace $R_c$ by $R_r$ in the list of robots assigned to $O_{ij}$ or $O_{kp}$ (50% probability)
                $P_3 \leftarrow P_3 \setminus \{R_r\}$
            end if
        end if
    end for
end
```

Fig. 13. *Call_Resolve* pseudo-algorithm

6.3 Local Search Algorithm

A local search procedure is necessary to locally optimize a given solution. Below, we present an efficient local search procedure that simulates gradient descent in convex optimization. The process is based on critical path analysis, where the objective is to identify blocks and apply an operator that results in an improved solution. The local search starts with an initial solution and iteratively explores its neighborhood by analyzing the vertices of the critical path, beginning with the last operation. In each iteration, it examines two consecutive operations on the critical path, and if a block is detected, it searches for an operator that would yield a better solution. If such an operator is found, it is applied to generate a new neighboring solution, updating the current solution. This process repeats on the new solution until a locally optimal solution is found or a maximum number of iterations is reached.

The local search mainspring is given in Fig. 14. The current solution is referred to $S_{best} = (OS_{best}, OA_{best})$, which is initialized by the initial solution $S_{best} = S_0$. The algorithm is based on a main loop which iterates along the vertices of the critical path $P^{S_{best}}$ from the last operation to the first one. The algorithm stops when the first vertex is reached or when the maximal number of iterations I_{max} is reached. At each iteration, the type of block is identified to find a move that would result in a better neighboring solution:

If (O_{ij}, O_{kp}) belong to the same job ($i = k$ and $p = j + 1$), then there is nothing that can be done and the search continues (next block). In case of a machine-block, the only possible move is to invert the processing order of the two operations. This is done using $Call\_Swap$ procedure, which locates in OS_{best} vector the indices corresponding to O_{ij} and O_{kp}, and swaps the respective elements. This produces a new vector OS, leading to the neighboring solution $S = (OS, OA_{best})$. The new solution S is then evaluated and if the new makespan $C_{max}(S)$ is better than the current makespan $C_{max}(S_{best})$, the current solution is updated ($S_{best} = S$) and the process restarts. Otherwise, the current solution is retained, and the search continues.

If (O_{ij}, O_{kp}) forms a transport-block, we can similarly attempt to invert the processing order of the two operations using $Call\_Swap$ procedure. If this move improves the current solution, it is updated, and the process restarts. However, if the move wouldn't result in a better neighboring solution, we can still attempt to resolve the robot disjunctions between O_{ij} and O_{kp}, provided the transport-block is resolvable.

Algorithm 2: Local Search

```
Input  : ($S_0$ : Initial solution, $I_{max}$ : Maximum number of iterations)
Output: $S_{best}$ : Best solution found
Function $LOCAL\_SEARCH(S_0, I_{max})$:
    $iter \leftarrow 0$            // Initialize number of iterations
    $S_{best} \leftarrow S_0$            // Initialize best solution found
    $(k,p) \leftarrow S_{best}.P(*)$ // $O_{kp}$ is the last operation on the critical path $P^{S_{best}}$
    while $((k,p) \neq (0,0)$ and $iter \leq I_{max})$ do
        $(i,j) \leftarrow S_{best}.P(k,p)$ // $O_{ij}$ is the predecessor of $O_{kp}$ on the critical path
        /* If $O_{ij}$ and $O_{kp}$ involve different jobs
        if $(i \neq k)$ then
            /* If $(O_{ij}, O_{kp})$ is a machine-block
            if ($O_{ij}$ and $O_{kp}$ are machine operations) then
                $S \leftarrow S_{best}$            // Initialize neighboring solution
                $Call\_Swap(S,i,j,k,p)$
                $Call\_Evaluate(S)$
                /* If Makespan improved
                if $S.C_{max} < S_{best}.C_{max}$) then
                    $S_{best} \leftarrow S$            // Update best solution found
                    $(k,p) \leftarrow S_{best}.P(*)$ // $O_{kp}$ is the last operation on the new critical path
                else
                    $(k,p) \leftarrow (i,j)$        // Do nothing.  Continue exploration
                end if
            else
                /* If $(O_{ij}, O_{kp})$ is a transport-block
                if ($O_{ij}$ and $O_{kp}$ are transport operations) then
                    $S \leftarrow S_{best}$
                    $Call\_Swap(S,i,j,k,p)$
                    $Call\_Evaluate(S)$
                    /* If Makespan improved
                    if $S.C_{max} < S_{best}.C_{max}$) then
                        $S_{best} \leftarrow S$
                        $(k,p) \leftarrow S_{best}.P(*)$ // $O_{kp}$ is the last operation on the new critical path
                    else
                        /* Check if $(O_{ij}, O_{kp})$ is resolvable
                        $res \leftarrow False$ // Boolean that indicates if the transport-block is resolvable
                        $Call\_Check\_Resolvable(S,i,j,k,p,res)$
                        /* If the transport-block is resolvable
                        if $(res = True)$ then
                            $S \leftarrow S_{best}$
                            $Call\_Resolve(S,i,j,k,p,res)$
                            $Call\_Evaluate(S)$
                            /* If Makespan improved
                            if $(S.C_{max} < S_{best}.C_{best})$ then
                                $S_{best} \leftarrow S$
                                $(k,p) \leftarrow S_{best}.P(*)$ // $O_{kp}$ is the last operation on the new critical path
                            else
                                $(k,p) \leftarrow (i,j)$        // Do nothing.  Continue exploration
                            end if
                        else
                            $(k,p) \leftarrow (i,j)$ // Do nothing.  Continue exploration
                        end if
                    end if
                end if
            end if
        else
            $(k,p) \leftarrow (i,j)$ // Do nothing.  Continue exploration
        end if
        $iter \leftarrow iter + 1$
    end while
    return $S_{best}$
end
```

Fig. 14. Local Search algorithm

To determine whether the transport-block is resolvable we use the $Call\_Check\_Resolve$ procedure. If the transport-block is not resolvable, there is nothing more that can be done, and the search continues. However, if the block is resolvable, it is resolved by $Call\_Resolve$ procedure, which updates OA_{best} by unassigning from O_{ij} or O_{kp} the robots in conflict, and assigning new compatible robots to either operation until all disjunctions are resolved. This produces a new vector OA, resulting in the neighboring solution $S = (OS_{best}, OA)$. The new solution S is then evaluated, and if the makespan $C_{max}(S)$ is better than the current makespan $C_{max}(S_{best})$, the current solution is updated ($S_{best} = S$) and the process restarts. Otherwise, the current solution is retained, and the search continues.

6.4 GRASPxELS metaheuristic

We employ the GRASP×ELS metaheuristic (Prins, 2009), a hybrid approach combining GRASP (Greedy Randomized Adaptive Search Procedure) (Feo and Resende, 1995) and ELS (Evolutionary Local Search) (Wolf and Merz, 2007; Prins, 2004), leveraging the strengths of both methods. GRASP follows a multi-start strategy, generating Np initial solutions using a greedy randomized heuristic, which are then improved through local search. ELS (Prins, 2004), an extension of Iterated Local Search (ILS) (Lourenço et al., 2003), begins with the solution obtained from a GRASP iteration. Each iteration involves creating multiple copies of the current solution, applying mutation operators, and improving them via local search. This strategy enables ELS to thoroughly explore attraction basins near the local optimum before transitioning to a new region in the search space. The GRASP×ELS hybrid effectively balances diversification, driven by GRASP's global exploration, and intensification, ensured by ELS's local refinement.

In our case, instead of using a randomized greedy heuristic, initial solutions are generated purely at random. Furthermore, rather than fixing the number of GRASP iterations, we generate as many solutions as possible within the given computational time limit. The number of ELS iterations after a GRASP iteration is denoted Ne. For the mutation process, we apply the $Call\_Swap$ procedure to swap two distinct randomly selected values in the OS vector of the solution. This mutation produces a child solution, which may be worse than the current one. The number of child solutions generated by mutation in each ELS iteration is denoted by Nc. To ensure that only new neighbors are considered, we track previously found solutions using a hash table. The hash key of each solution is computed as the sum of the squared starting times of all operations.

7 Numerical experiments

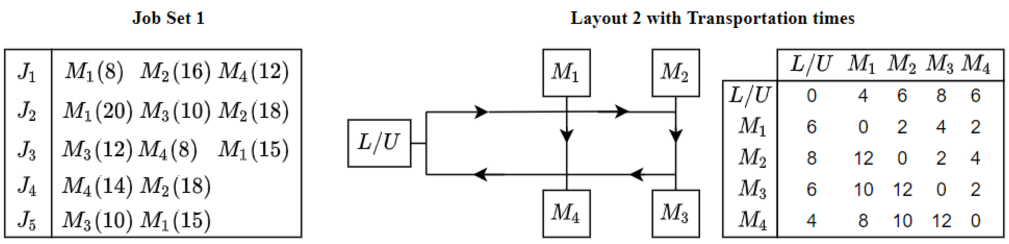
In this section, we first compare our MILP approach to the Constraint Programming (CP) approach developed by Ham (2021) for JSPT, which significantly outperforms all other benchmark approaches in the literature. We show that our MILP has very good performance, although not dedicated to JSPT, on the instances of Bilge and Ulusoy (1995). We then adapt the JSPT instances of Bilge and Ulusoy to create JSPT-C instances including multiple robots and synchronization among robots. We show that our metaheuristic provides optimal or near-optimal solutions in a short computational time. Finally, we present new large JSPT-C instances and show that the metaheuristic outperforms the MILP.

All experiments were conducted on a laptop equipped with an Intel® Core™ i7-1365U CPU @ 1.80 GHz and 32 GB of RAM. To favour fair future research on the topic, all the data sets and all the solutions can be download at: https://perso.isima.fr/~amoussam/index_files/pages/Research/JSPT-C.html

7.1 Comparative study between MILP and CP2 (Ham, 2021) for solving classical JSPT instances

The classical Job-Shop Problem with Transportation resources (JSPT) does not involve robot cooperation and can be considered a special case of our problem where all transportation operations require only a single robot to be performed. Therefore, our model for the JSPT-C is expected to solve classical JSPT instances. We conducted an initial experimental study aimed at validating our proposed MILP and demonstrating its effectiveness in solving the classical JSPT compared to the best-published methods.

The proposed MILP is benchmarked using the first set of instances provided by Bilge and Ulusoy (1995). This dataset includes 40 instances where the transportation-to-processing time ratio is greater than 0.25. Each instance is labeled with the prefix EX followed by two digits representing the job-set and layout (e.g., Fig. 15 illustrates the job-set and layout used in instance EX12).



J_1	$M_1(8)$ $M_2(16)$ $M_4(12)$
J_2	$M_1(20)$ $M_3(10)$ $M_2(18)$
J_3	$M_3(12)$ $M_4(8)$ $M_1(15)$
J_4	$M_4(14)$ $M_2(18)$
J_5	$M_3(10)$ $M_1(15)$

	L/U	M_1	M_2	M_3	M_4
L/U	0	4	6	8	6
M_1	6	0	2	4	2
M_2	8	12	0	2	4
M_3	6	10	12	0	2
M_4	4	8	10	12	0

Fig. 15. Job-set and layout for instance EX12

To adapt a classical JSPT instance from Bilge and Ulusoy's formulation to our model, we consider that all transportation operations require a single robot, and all robots are compatible with each transportation operation. Furthermore, as they do not differentiate between loaded and unloaded transportation times, we consider that for each source machine s and destination machine d, the loaded transportation time $T_{s,d}$ is equal to the unloaded travel time $V_{s,d}$.

We compare our results with one of the most recent published methods denoted CP2 (Ham, 2021) which is based on a Constraint Programming (CP) formulation using IBM ILOG CP Optimizer. To ensure a fair comparative study, the computational time of each method has been scaled by the speed factor presented in Table 1. This coefficient takes into account the CPU Mark of each processor which can be found in https://www.cpubenchmark.net/compare/1919vs5266/Intel-i7-4770K-vs-Intel-i7-1365U. Table 1 also provides the information available on the computer, the operating system and the programming language.

Table 1
Relative performances of computers

	(Ham, 2021)	Our framework
Computer	Intel Core i7-4770	Intel Core i7-1365U @1.80 GHz
OS	/	Windows 10 Pro
Language	IBM ILOG CP Optimizer	IBM ILOG CPLEX (Visual C++)
CPU Mark	7129	14799
Speed factor	1	2.07

We evaluate the performance of our proposed MILP under a computational time limit equal to the maximum time spent by CP2 method (Ham, 2021) to converge to optimality for an instance which is 792.88 seconds as reported for instance EX74. Given the speed factor of 2.02 for our computer, a computational time of 792.88s corresponds to 383 seconds on the computer used for this study ($383 = 792.88 \times \frac{1}{2.07}$).

The following notations are used in Tables 2, 4 and 9:

- S Best Makespan found. The bold indicates that the solver converged to optimality
- T Total computational time (in seconds)
- T_{scale} Scaled computational time (in seconds);
- T_{best} Time to best found solution in T_{limit}
- T^* Time required for the MILP to converge to optimality (in seconds);
- GAP Linear relaxation GAP reported by CPLEX Solver (in %);
- GAP^* Gap to optimal solution ($GAP^* = 100 \times (\frac{S}{S^*} - 1)$) where S is the best found solution and S^* is the optimal solution.

As stressed in Table 2 below, with a computational time limit of 383 seconds, our proposed MILP converges to optimality for 38 instances and is close to convergence for instances EX71 and EX74. The proposed MILP reports an average relative GAP of 1.56% in an average computational time of 36.26 seconds which corresponds to 75 seconds given the speed factor, approximately 2 times slower than the CP2 method. This shows that our model, which is dedicated to a more general problem, has some merit in solving the classical JSPT. The results are also coherent with the expectations of Ham (2021), where he points that Constraint

Programming (CP) has been shown to outperform MIP significantly particularly in sequencing, scheduling, and highly constrained problems, which leads us to consider CP as potentially a more suitable exact approach for tackling our problem effectively, especially with the increased number of constraints related to robot synchronization for cooperation.

Table 2
Detailed comparative study between our proposed MILP and CP2 (Ham, 2021) on (Bilge and Ulusoy, 1995) instances

	CP2 (HAM,2021)		OUR PROPOSED MILP			
INSTANCE	***S***	***T***	***S***	***GAP***	***T***	T_{scale}
EX11	**96**	0.83	**96**	0	2.98	6.17
EX12	**82**	0.44	**82**	0	0.49	1.01
EX13	**84**	0.39	**84**	0	0.77	1.59
EX14	**103**	1.26	**103**	0	2.24	4.64
EX21	**100**	2.21	**100**	0	5.53	11.45
EX22	**76**	0.17	**76**	0	1.09	2.26
EX23	**86**	1.59	**86**	0	4.41	8.82
EX24	**108**	5.02	**108**	0	10.88	22.52
EX31	**99**	1.26	**99**	0	4.40	9.11
EX32	**85**	0.49	**85**	0	1.66	3.44
EX33	**86**	0.26	**86**	0	0.94	1.95
EX34	**111**	2.08	**111**	0	9.14	18.92
EX41	**112**	22.42	**112**	0	142.35	294.66
EX42	**87**	3.88	**87**	0	29.21	60.46
EX43	**89**	2.37	**89**	0	22.48	46.53
EX44	**121**	7.60	**121**	0	26.01	52.02
EX51	**87**	1.39	**87**	0	6.68	13.83
EX52	**69**	0.46	**69**	0	0.95	1.97
EX53	**74**	1.23	**74**	0	1.44	2.98
EX54	**96**	2.18	**96**	0	1.83	3.79
EX61	**118**	4.40	**118**	0	55.25	114.37
EX62	**98**	0.39	**98**	0	3.82	7.91
EX63	**103**	0.66	**103**	0	4.72	9.77
EX64	**120**	3.72	**120**	0	15.96	33.04
EX71	**111**	406.28	112	23.21	383	792.81
EX72	**79**	1.22	**79**	0	16.84	34.86
EX73	**83**	2.58	**83**	0	18.76	38.83
EX74	**126**	792.88	127	39.21	383	792.81
EX81	**161**	0.07	**161**	0	4.47	9.25
EX82	**151**	0.02	**151**	0	2.25	4.66
EX83	**153**	0.01	**153**	0	4.84	10.02
EX84	**163**	0.14	**163**	0	3.83	7.93
EX91	**116**	0.95	**116**	0	2.91	6.02
EX92	**102**	0.27	**102**	0	3.06	6.33
EX93	**105**	0.38	**105**	0	4.70	9.73
EX94	**120**	1.30	**120**	0	9.92	20.53
EX101	**146**	2.35	**146**	0	42.67	88.33
EX102	**135**	0.67	**135**	0	7.37	15.26
EX103	**137**	1.30	**137**	0	17.95	37.16
EX104	**157**	12.80	**157**	0	189.78	392.84
AVERAGE		32.24		1.56	36.26	75.01

7.2 MILP vs GRASPxELS in solving JSPT-C instances adapted from (Bilge and Ulusoy, 1995)

Since the JSPT-C has never been studied before, we created an initial set of instances by adapting 10 instances from the previous study. For each instance we created 5 new instances with the same job-set and layout for which we introduced cooperation requirements with a total number of robots $NR \in \{2,3,5,7,10\}$. Each JSPT-C instance is denoted by the name of the JSPT instance used followed by prefix "C" and the number of robots. For example, EX11C3 is a JSPT-C instance with 3 robots and based on the same job-set and layout of EX11.

We report the performance of our GRASPxELS metaheuristic, executed under a computational time limit of 65 seconds to solve the 50 generated JSPT-C instances, in comparison with our exact approach in terms of the time required to achieve the best solution within the given time limit. The metaheuristic was executed 5 times, and the best solution across the 5 runs was selected based on solution quality. The parameters for the GRASPxELS are given in Table 3. These parameters were not fine-tuned through an extensive experimental plan, as the primary focus of this study is methodological. However, they were selected based on preliminary experiments aiming to ensure a reasonable balance between computational effort and solution quality. To improve transparency, we now explicitly mention this rationale in Section 7.2 of the manuscript. We also acknowledge that further parameter tuning could enhance performance and consider this a valuable direction for future work.

Table 3
GRASP×ELS parameters for the JSPT

Parameter	Definition	Value
Ne	Number of ELS iterations	15
Nc	Number of child solutions	10
Nl	Number of local search iterations	10

As reported in Table 4, we can see that the MILP converges to optimality for all the 50 instances in an average computational time of 4.72 seconds, and the GRASPxELS competes well with the exact approach providing high quality solutions in an average computational time of only 7.76 seconds with an average GAP of 1.19% to optimality. Although these instances are relatively small with a maximum of 8 jobs and 4 machines, we can observe that the metaheuristic begins to struggle for the instances with 7 and 10 robots.

.

Table 4
Comparative study between GRASP×ELS and MILP resolution

	MILP			GRASPXELS		
INSTANCES	S	GAP^*	T^*	S	GAP^*	T_{best}
EX11C2	113	0	0.74	113	0	0.04
EX11C3	88	0	0.25	88	0	0.02
EX11C5	92	0	0.42	92	0	0.31
EX11C7	94	0	0.50	94	0	0.02
EX11C10	100	0	1.07	100	0	3.52
EX24C2	138	0	0.96	138	0	2.17
EX24C3	142	0	1.25	142	0	6.49
EX24C5	94	0	0.59	94	0	0.26
EX24C7	96	0	0.48	96	0	0.30
EX24C10	119	0	3.03	120	0.84	0.80
EX31C2	123	0	0.72	123	0	2.08
EX31C3	132	0	1.30	132	0	0.16
EX31C5	117	0	0.84	117	0	1.32
EX31C7	101	0	1.66	101	0	0.23
EX31C10	102	0	1.58	102	0	6.93
EX41C2	139	0	1.50	139	0	1.14
EX41C3	124	0	1.42	125	0.81	0.28
EX41C5	118	0	1.56	118	0	14.62
EX41C7	103	0	3.16	104	0.97	4.90
EX41C10	101	0	14.07	102	0.99	11.39
EX54C2	127	0	0.58	127	0	0.04
EX54C3	130	0	0.55	130	0	0.44
EX54C5	123	0	0.67	123	0	0.08
EX54C7	100	0	0.35	100	0	0.09
EX54C10	107	0	3.12	107	0	1.57
EX61C2	134	0	1.90	134	0	15.86
EX61C3	119	0	1.56	119	0	33.52
EX61C5	112	0	1.28	112	0	2.85
EX61C7	116	0	1.84	116	0	28.80
EX61C10	129	0	14.31	133	3.10	31.66
EX71C2	148	0	30.76	156	5.40	2.53
EX71C3	124	0	10.47	128	3.22	2.93
EX71C5	98	0	2.90	101	3.06	12
EX71C7	135	0	2.89	143	5.92	2.77
EX71C10	99	0	67.74	106	7.07	18.84
EX84C2	186	0	4.82	193	3.76	10.41
EX84C3	163	0	2.79	163	0	9.58
EX84C5	165	0	5.20	174	5.45	40.73
EX84C7	163	0	4.10	163	0	0.07
EX84C10	163	0	6.80	163	0	9.63
EX94C2	179	0	2.19	179	0	5.83
EX94C3	123	0	0.45	123	0	0.43
EX94C5	140	0	0.98	142	1.43	9.80
EX94C7	137	0	1.59	137	0	2.49
EX94C10	116	0	2.29	116	0	0.13
EX102C2	144	0	3.18	155	7.64	2.38
EX102C3	138	0	4.60	138	0	62.35
EX102C5	145	0	4.16	153	5.51	3.97
EX102C7	131	0	3.39	131	0	0.07
EX102C10	134	0	11.56	140	4.48	19.33
AVERAGE		0	4.72		1.19	7.76

7.3 Impact of robot compatibility constraints

In this section, we briefly consider how the influence of the number of compatible robots influences the objective function. To illustrate the impact of robot compatibility constraints, we consider the JSPT instance EX41, which we adapt to the JSPT-C setting by introducing cooperation requirements. Specifically, we fix the total number of robots to 10 and set the required number of robots for each transportation operation to 3. We begin with no flexibility, setting exactly 3 compatible robots per transportation operation. Then, we incrementally increase the number of compatible robots by one, randomly selecting the compatible robot sets. This resulted in 8 instances, with the number of compatible robots per transportation operation ranging from 3 (restricted compatibility) to 10 (full compatibility). We evaluate the performance of our MILP model on each instance under a time limit of 65 seconds. Table 5 reports the makespan and computational time when the number of compatible robots per transportation operation increases.

Table 5
Impact of the number of compatible robots per transportation operation on solution quality and computational time

Number of compatible robots per transportation operation	Best found solution	Computational time (s)	Linear relaxation GAP (%)
3	139	1.09	0
4	105	11.59	0
5	97	65	17.53
6	92	65	16.13
7	91	65	15.22
8	94	65	19.15
9	94	65	17.02
10	94	65	17.02

A substantial improvement in solution quality—approximately 25%—is observed when increasing the number of compatible robots from 3 to 4, albeit with a corresponding rise in computational time. Beyond this point, further increases in compatibility yield more modest improvements in solution quality, while the MILP model fails to converge to optimality within the imposed time limit, as evidenced by a persistent linear relaxation gap.

This behavior is expected, as increasing the number of compatible robots expands the set of feasible robot assignments, thereby enlarging the solution space. However, the benefits of added compatibility diminish beyond a certain threshold, indicating that additional flexibility does not necessarily translate into better scheduling outcomes. These findings confirm that restricted compatibility constrains scheduling flexibility and reduces the diversity of feasible robot assignments, which can exacerbate resource bottlenecks. Conversely, relaxing compatibility constraints improves solution quality, but at the cost of increased computational effort due to the larger and more complex solution space.

7.4 MILP vs GRASPxELS in solving new large-scale JSPT-C instances

The benchmark instances of Bilge and Uluosy (1995) that we adopted in the previous study have only 8 jobs and 4 machines. Although we increased the number of robots, the size of instances remains relatively small and doesn't challenge our model enough. In the following study, we have created large benchmark instances for the JSPT-C. We used 3 layouts depicted in Fig. 16, 17 and 18 and 9 job-sets to produce 9 large-scale instances denoted by prefix "JL" followed by the instance number. As highlighted in Table 6, the total number of operations ranges from 128 to 480. The largest instance, JL9, consists of 480 operations, including 240 production operations and 240 transportation operations.

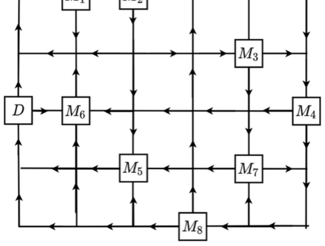


Fig. 16. Layout 1 (8 machines)

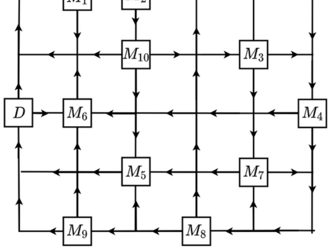


Fig. 17. Layout 2 (10 machines)

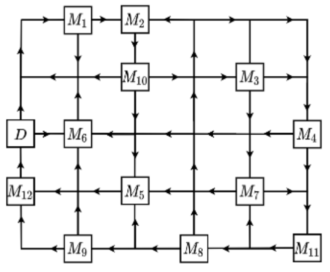


Fig. 18. Layout 3 (12 machines)

Table 6
Instances details

Instances	Job-set	Layout	N	M	NR	Total number of operations
JL1	1	1	8	8	5	128
JL2	2	1	10	8	7	160
JL3	3	1	12	8	10	192
JL4	4	2	10	10	7	200
JL5	5	2	12	10	10	240
JL6	6	2	15	10	12	300
JL7	7	3	12	12	10	288
JL8	8	3	15	12	12	360
JL9	9	3	20	12	15	480

For instance, we introduce in Table 7 the operations of job 1 in job-set 1, which is the smallest job set of the benchmark. The first job of the job-set is composed of 16 operations. The first operation is a transportation operation where 2 robots have to be selected from the set $\mathbb{C}_{11} = \{R1,\ R5\}$, and the second operation is a production operation on machine $M7$ for a processing time of 37 units.

Table 7
Job 1 of the Job-Set 1

Sequence of operations	Description of the operations
Op 1	2 robots $\in \{R1,\ R5\}$
Op 2	M7(37)
Op 3	1 robot $\in \{R_3\}$
Op 4	M6(39)
Op 5	2 robots $\in \{R_1, R_2\}$
Op 6	M1(43)
Op 7	3 robots $\in \{R_1, R_2, R_3, R_4, R_5\}$
Op 8	M3(38)
Op 9	2 robots $\in \{R_2, R_4, R_5\}$
Op 10	M8(41)
Op 11	3 robots $\in \{R_3, R_4, R_5\}$
Op 12	M2(43)
Op 13	4 robots $\in \{R_1, R_2, R_3, R_4, R_5\}$
Op 14	M4(43)
Op 15	1 robot $\in \{R_1\}$
Op 16	M5(35)

The parameters of the GRASP×ELS are tuned to address the instances scale with a higher number of ELS and local search iterations, and a higher number of child solutions during evolutionary search (Table 8).

Table 8
GRASP×ELS parameters for the JSPT-C large scale instances

Parameter	Definition	Value
Ne	Number of ELS iterations	40
Nc	Number of child solutions	30
Nl	Number of local search iterations	50

We report the performance of our GRASPxELS metaheuristic, executed under a computational time limit of 60 seconds to solve the 9 generated JSPT-C large instances, in comparison with our exact approach which was executed with computational time limits of 60 seconds and 3600 seconds. The metaheuristic was executed 5 times, and the best solution across the 5 runs was selected based on solution quality.

As shown in Table 9, under a computational time limit of 60 seconds, the metaheuristic produces high-quality solutions, whereas the MILP struggles to find feasible solutions for nearly all instances within the same time frame. Even after one hour of computation, the MILP yields significantly worse solutions than those obtained by the metaheuristic in just one minute. This highlights the effectiveness of both our modeling and local search strategies.

Table 9
GRASP×ELS resolution for the large instances

				MILP 60s		MILP 3600s		GRASPXELS Best run	
INSTANCES	N	M	NR	S	GAP	S	GAP	S	T_{best}
JL1	8	8	5	780	37	668	22	758	15.34
JL2	10	8	7	1158	59	816	39	792	41.49
JL3	12	8	10	/	/	1260	61	1049	40.32
JL4	10	10	7	/	/	1157	45	1235	19.42
JL5	12	10	10	/	/	1574	59	1421	35.56
JL6	15	10	12	/	/	2714	76	1893	59.86
JL7	12	12	10	/	/	1942	58	1808	29.00
JL8	15	12	12	/	/	3032	73	2416	52.55
JL9	20	12	15	/	/	/	/	3265	39.00
AVERAGE									**36.94**

8 Conclusion

In this paper, we introduce a new extension of JSPT by considering the cooperation of multiple robots to perform a transportation operation. We model this problem using a Mixed Integer Linear Programming (MILP) approach and illustrate how it can be represented by a disjunctive graph. We describe a feasible solution through an operation sequence vector and a robot assignment vector, along with an efficient algorithm for evaluating such solutions. Next, we introduce a GRASP×ELS metaheuristic integrated with a local search procedure. Finally, in a numerical study, we adapt benchmark instances to fit our problem framework and generate new large-scale instances tailored to our problem. While the metaheuristic performs competitively with exact methods for small to medium instances, it surpasses the MILP approach for larger instances.

In future research, it would be interesting first to improve the procedure for assigning robots to transportation operations, which is random in our metaheuristic. We could, for example, try to balance the workload among the robots. In practice, transportation times are random variables, due to congestion, and depend on the number of robots involved in the cooperation. Moreover, cooperation influences transportation times. Depending on the robots considered, transport times can increase or decrease with the number of robots involved in the cooperation. It would therefore be interesting to consider an extension where transportation times are stochastic and depend on cooperation. Another possible avenue for future research would be to explore the performance of the proposed algorithms on a broader range of JSPT instances, which could provide additional insights into their generality and effectiveness.

Acknowledgment

This work was supported by the International Research Center "Innovation Transportation and Production Systems" of the I-SITE CAP 20-25.

References

Albert, B., Chaikovskaia, M., Gayon, J.P. and Quilliot, A., 2025. Sizing and scheduling for a fleet of reconfigurable mobile robots. *International Journal of Production Research*, pp.1-18.

An, X., Wu, C., Lin, Y., Lin, M., Yoshinaga, T. and Ji, Y., 2023. Multi-robot systems and cooperative object transport: Communications, platforms, and challenges. *IEEE Open Journal of the Computer Society*, *4*, pp.23-36.

Berghman, L., Kergosien, Y. and Billaut, J.C., 2023. A review on integrated scheduling and outbound vehicle routing problems. *European Journal of Operational Research*, *311*(1), pp.1-23.

Berterottière, L., Dauzère-Pérès, S. and Yugma, C., 2024. Flexible job-shop scheduling with transportation resources. *European Journal of Operational Research*, *312*(3), pp.890-909.

Bierwirth, C., 1995. A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum*, *17*(2), pp.87-92.

Bilge, Ü. and Ulusoy, G., 1995. A time window approach to simultaneous scheduling of machines and material handling system in an FMS. *Operations Research*, *43*(6), pp.1058-1070.

Brucker, P., Jurisch, B. and Sievers, B., 1994. A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, *49*(1-3), pp.107-127.

Chaikovskaia, M., Gayon, J.P. and Marjollet, M., 2022, August. Sizing of a fleet of cooperative and reconfigurable robots for the transport of heterogeneous loads. In *2022 IEEE 18th international conference on automation science and engineering (CASE)* (pp. 2253-2258). IEEE.

Chaikovskaia, M., Gayon, J.P. and Quilliot, A., 2025. Optimization of a fleet of reconfigurable robots. *Flexible Services and Manufacturing Journal*, pp.1-23.

Deroussi, L., Gourgand, M. and Tchernev, N., 2008. A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles. *International Journal of Production Research*, *46*(8), pp.2143-2164.

Drexl, M., 2012. Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science*, *46*(3), pp.297-316.

Feo, T.A. and Resende, M.G., 1995. Greedy randomized adaptive search procedures. *Journal of global optimization*, *6*, pp.109-133.

Fontes, D.B.M. and Homayouni, S.M., 2019. Joint production and transportation scheduling in flexible manufacturing systems. *Journal of Global Optimization*, *74*(4), pp.879-908.

Fontes, D.B., Homayouni, S.M. and Gonçalves, J.F., 2023. A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources. *European Journal of Operational Research*, *306*(3), pp.1140-1157.

Garey, M.R., Johnson, D.S. and Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, *1*(2), pp.117-129.

Gayon, J.P., Lacomme, P. and Oussama, A., 2024, June. Job-Shop Scheduling with Robot Synchronization for Transport Operations. In *Metaheuristics International Conference* (pp. 28-42). Cham: Springer Nature Switzerland.

Grabowski, J., Nowicki, E. and Zdrzałka, S., 1986. A block approach for single-machine scheduling with release dates and due dates. *European Journal of Operational Research*, *26*(2), pp.278-285.

Ham, A., 2021. Transfer-robot task scheduling in job shop. *International Journal of Production Research*, *59*(3), pp.813-823.

Hurink, J. and Knust, S., 2002. A tabu search algorithm for scheduling a single robot in a job-shop environment. *Discrete applied mathematics*, *119*(1-2), pp.181-203.

Hurink, J. and Knust, S., 2005. Tabu search algorithms for job-shop problems with a single transport robot. *European journal of operational research*, *162*(1), pp.99-111.

Knust, S., 2000. Shop-scheduling problems with transportation.

Kuhpfahl, J. and Bierwirth, C., 2016. A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research*, *66*, pp.44-57.

Lacomme, P., Larabi, M. and Tchernev, N., 2007. A disjunctive graph for the job-shop with several robots. In *MISTA conference* (Vol. 20, pp. 285-292).

Lacomme, P., Larabi, M. and Tchernev, N., 2013. Job-shop based framework for simultaneous scheduling of machines and automated guided vehicles. *International Journal of Production Economics*, *143*(1), pp.24-34.

Lourenço, H.R., Martin, O.C. and Stützle, T., 2003. Iterated local search. In *Handbook of metaheuristics* (pp. 320-353). Boston, MA: Springer US.

MecaBotiX, https://www.mecabotix.com/ (2023).

Nguyen, C.T., Gayon, J.P., Nguyen, V.H., Quilliot, A. and Ta, A.S., 2023, December. Pickup and Delivery Problem with Cooperative Robots. In *Proceedings of the 12th International Symposium on Information and Communication Technology* (pp. 695-700).

Nouri, H.E., Driss, O.B. and Ghédira, K., 2016. A classification schema for the job shop scheduling problem with transportation resources: state-of-the-art review. In *Artificial Intelligence Perspectives in Intelligent Systems: Proceedings of the 5th Computer Science On-line Conference 2016 (CSOC2016), Vol 1* (pp. 1-11). Springer International Publishing.

Pinedo, M. (2012). Scheduling - Theory, algorithms and systems. Springer.

Prins, C., 2004. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & operations research*, *31*(12), pp.1985-2002.

Prins, C., 2009. A GRASP× evolutionary local search hybrid for the vehicle routing problem. In *Bio-inspired algorithms for the vehicle routing problem* (pp. 35-53). Berlin, Heidelberg: Springer Berlin Heidelberg.

Raman, N., 1986. Simultaneous scheduling of machines and material handling devices in automated manufacturing. In *Proc. of the Second ORSA/TIMS Conference on Flexible Manufacturing Systems: Operations Research Models and Applications, 1986.*

Roy, B. and Sussmann, B., 1964. Les problemes d'ordonnancement avec contraintes disjonctives. *Note ds*, *9*.

Seo, J., Paik, J. and Yim, M., 2019. Modular reconfigurable robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, *2*(1), pp.63-88.

Ulusoy, G. and Bilge, Ü., 1992. Simultaneous scheduling of machines and material handling system in an FMS. *IFAC Proceedings Volumes*, *25*(8), pp.15-25.

Van Laarhoven, P.J., Aarts, E.H. and Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. *Operations research*, *40*(1), pp.113-125.

Wolf, S. and Merz, P., 2007, October. Evolutionary local search for the super-peer selection problem and the p-hub median problem. In *International workshop on hybrid metaheuristics* (pp. 1-15). Berlin, Heidelberg: Springer Berlin Heidelberg.

Yao, Y., Gui, L., Li, X. and Gao, L., 2024. Tabu search based on novel neighborhood structures for solving job shop scheduling problem integrating finite transportation resources. *Robotics and Computer-Integrated Manufacturing*, *89*, p.102782.