A Constraint Programming Model for Job-Shop Scheduling with Cooperative Transportation Resources

Amine Oussama, Jean-Philippe Gayon, and Philippe Lacomme

***Abstract*— This paper presents a Constraint Programming (CP) model for the Job-Shop Scheduling Problem with Cooperative Transportation Resources (JSPT-C), an extension of the Job-Shop Scheduling Problem with Transportation Resources (JSPT) in which transportation operations require synchronized teams of robots that cooperate to transport jobs between machines. The proposed CP formulation relies on interval variables, optional interval variables, and sequence variables, combined with global scheduling constraints to capture robot cooperation, synchronization and resource disjunctions. The effectiveness of the proposed approach is demonstrated through computational experiments on benchmark instances from the literature, showing that the CP model efficiently solves the problem and outperforms existing solution methods.**

I. Introduction

Modern manufacturing systems increasingly rely on fleets of mobile robots or automated guided vehicles to transport jobs between machines. In such environments, scheduling production operations and transportation activities are strongly interdependent, since the availability of transportation resources directly affects machine utilization and overall system performance. Efficient scheduling of both production and transportation tasks is therefore crucial for optimizing key performance measures such as throughput, resource utilization, and total production time. The classical Job-Shop Scheduling Problem (JSP) focuses on sequencing production operations on machines, while the Job-Shop Scheduling Problem with Transportation Resources (JSPT) extends this framework by explicitly modeling the transportation of jobs between machines using mobile robots or automated guided vehicles (see [1], [2] for surveys).

Recent developments in multi-robot systems have enabled cooperative object transport, where multiple robots cooperate to carry and move heavy, bulky or fragile items that cannot be handled by a single robot. Such cooperative robotic systems have attracted significant interest due to their wide range of applications, including agriculture, mining, logistics, space exploration, and warehouse automation [3]. In manufacturing environments, cooperative transportation naturally arises when large, heavy, or fragile components cannot be safely handled by a single robot. Examples include the transport of aircraft fuselage sections, automotive body parts, large industrial equipment, and oversized products in automated warehouses. In such settings, synchronization between multiple robots is a practical operational requirement rather than a purely theoretical modeling assumption.

Motivated by their applications in manufacturing environments, especially in factory intralogistics and automated material handling systems, Gayon et al. [4], [5] introduced the Job-Shop Scheduling Problem with Cooperative Transportation Resources (JSPT-C), in which transportation operations may require the cooperation of multiple robots that must execute the task simultaneously. In this problem, each transportation task requires a specific number of robots, and all the robots assigned to the task must start the operation at the same time and remain jointly engaged for its entire duration. The JSPT-C therefore integrates machine scheduling, robot assignment, routing, and synchronization constraints, which makes it significantly more complex than the classical JSPT.

To solve the JSPT-C, Gayon et al. [5] proposed a Mixed-Integer Linear Programming (MILP) formulation and a hybrid metaheuristic that combines a Greedy Randomized Adaptive Search Procedure (GRASP) with Evolutionary Local Search (ELS). However, the combinatorial complexity introduced by robot cooperation and synchronization makes large instances particularly challenging to solve, especially when using the MILP formulation as an exact approach. Although the metaheuristic can produce good feasible schedules, the MILP fails to find feasible solutions within reasonable computational time as the instance size increases. In parallel, recent studies have shown that Constraint Programming (CP) is particularly well suited for complex scheduling problems and highly constrained problems [6]. For instance, Ham [7] demonstrated that CP significantly outperforms traditional methods on the classical JSPT.

Motivated by these results, this paper proposes the first CP model for the JSPT-C. The aim of this paper is to investigate whether CP provides an effective solution approach for the JSPT-C and to evaluate its efficiency in solving medium- and large-scale instances of the problem. The main contributions of this paper are: (1) a mathematical formulation capturing robot cooperation and synchronization in a job-shop environment; (2) a CP model using interval variables, optional interval variables, and sequence variables within CP Optimizer; and (3) a computational study on benchmark instances comparing the proposed CP approach with the MILP formulation and the GRASPxELS metaheuristic.

II. Problem Definition

The JSPT-C considers a set of n jobs J to be processed on a set of nm machines M. Each job is composed of n_i production operations, each assigned to a specific machine and must be processed in a predefined technological order. A

The authors are with Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France (e-mails : amine.oussama@doctorant.uca.fr, j-philippe.gayon@uca.fr, philippe.lacomme@isima.fr).

machine can process only one job at a time, and a job is processed on a specific machine at most once. Additionally, jobs must be transported between machines by a fleet of *nr* cooperative robots that can temporarily combine and coordinate to form a team of robots that acts as a single transportation unit to adapt to factors such as job size, weight or fragility. All jobs and robots are initially located at the depot *D.* A job must first be transported from the depot to the machine processing its first production operation (in the technological order). It is then processed on that machine for an uninterrupted duration after which it is transported to the next required machine, and so on. No transportation operation is considered after the last production operation of a job (no return to the depot). Hence, each job alternates between transportation and production operations. For a job *i,* the total number of operations is $o_i = 2 \times n_i$. The ordered sequence of operations of job *i* is therefore denoted $(O_{i1}, O_{i2}, \ldots, O_{i,oi})$, where odd-indexed operations correspond to transportation operations and even-indexed operations correspond to production operations.

Production operation O_{ij} must be processed on machine m_{ij} for an uninterrupted duration d_{ij}. Transportation operation O_{ij} moves job *i* from machine $m_{i,j-1}$ to machine $m_{i,j+1}$ (where $m_{i,0} = D$) and must be carried out by a team of robots of size r_{ij}, meaning that exactly r_{ij} robots must cooperate to perform the task. Transportation operation O_{ij} may be assigned to any combination of r_{ij} robots among the set of robots *R*, and the cooperation of the selected team members for this operation implies that they must all be simultaneously available at the pickup location and initiate the transportation at the same time and remain jointly engaged for its entire duration. Besides transportation operations which correspond to loaded travel movements, robots also perform empty travel movements to reposition themselves between consecutive transportation tasks, as the delivery location of a completed transport generally differs from the pickup location of the next transportation operation. The travel time from machine *s* to machine *d,* denoted l_{sd}, is assumed to be independent of the size and composition of robots involved in the transport and does not differ between loaded and empty movements. It follows that the duration of transportation operation O_{ij} is $d_{ij} = l_{m\_\{i,j-1\},m\_\{i,j+1\}}$. A team of robots is assumed to transport only one job at a time; although larger teams could, in principle, carry multiple jobs simultaneously, such multi-job handling is not considered in this study. Assembly and disassembly times of teams are considered negligible. This assumption is realistic when robots are physically homogenous and cooperation is achieved through software coordination. When reconfiguration times are significant, they can be incorporated as setup times between transportation operations, which constitutes an interesting extension of the problem.

The objective of the JSPT-C is to determine (1) the assignment of each transportation operation to the required number of robots (robot assignment), and (2) the interdependent schedules of production operations on machines and transportation operations on robots (production and transportation scheduling), so as to minimize the makespan, defined as the completion time of the last finished job. The JSPT-C therefore integrates three interconnected subproblems: assigning transportation operations to robots, sequencing production operations on machines, and routing the robots, which makes it a particularly challenging optimization problem. Furthermore, the JSPT-C is NP-hard since it generalizes the classical JSP, which is known to be NP-hard [8].

III. Constraint Programming Model

A. Mathematical model

This section presents the complete mathematical model for the JSPT-C, including the definition of all parameters, decision variables, constraints, and the objective function.

1) Problem parameters

The parameters used throughout this paper to describe the JSPT-C problem are:

- J : Set of jobs.
- M : Set of machines.
- R : Set of robots.
- O: Set of all operations.
- P : Set of production operations.
- T : Set of transportation operations.
- n : Number of jobs.
- nm : Number of machines.
- nr : Number of robots.
- n_i : Number of production operations of job $i \in J$.
- o_i : Number of operations of job $i \in J$ ($o_i = 2 \times n_i$).
- m_{ij} : Machine required for production operation $O_{ij} \in P$.
- r_{ij} : Number of robots required for transportation operation $O_{ij} \in T$.
- l_{sd} : Travel time from machine s to machine d where $s,d \in M \cup \{D\}$.
- d_{ij} : Duration of operation $O_{ij} \in O$.
- h : Scheduling horizon.

2) Decision variables

The model uses the following decision variables:

- ST_{ij} : Starting time of operation $O_{ij} \in O$.
- FT_{ij} : Finishing time of operation $O_{ij} \in O$.
- A_{ijr} : Binary variable that equals 1 if robot $r \in R$ is assigned to transportation operation $O_{ij} \in T$ and equals 0 otherwise.
- C_{max} : Makespan.

3) Constraints

The mathematical formulation consists of the following constraints:

$$\forall O_{ij} \in O: ST_{ij} \in [0, h] \quad (1)$$

$$\forall O_{ij} \in O: FT_{ij} \in [0, h] \quad (2)$$

$$C_{max} \in [0, h] \quad (3)$$

$$\forall O_{ij} \in O: FT_{ij} = ST_{ij} + d_{ij} \quad (4)$$

$$\forall i \in J, \forall j = 1..o_i - 1: FT_{ij} \leq ST_{i,j+1} \quad (5)$$

$$\forall O_{ij}, O_{kp} \in P \text{ such that } i < k:$$

$$m_{ij} = m_{kp} \Rightarrow (FT_{ij} \leq ST_{kp}) \vee (FT_{kp} + \leq ST_{ij}) \quad (6)$$

$$\forall O_{ij} \in T: ST_{ij} \geq l_{D,m\_\{i,j-1\}} \quad (7)$$

$$\forall O_{ij} \in T: \sum_{r=1..nr} A_{ijr} = r_{ij} \quad (8)$$

$$\forall r \in R, \forall O_{ij}, O_{kp} \in T \text{ such that } i < k:$$

$$A_{ijr} = A_{kpr} = 1 \Rightarrow (FT_{ij} + l_{m\_\{i,j+1\},m\_\{k,p-1\}} \leq ST_{kp}) \vee (FT_{kp} + l_{m\_\{k,p+1\},m\_\{i,j-1\}} \leq ST_{ij}) \quad (9)$$

$$\forall i \in J: FT_{i,o\_i} \leq C_{max} \quad (10)$$

Constraint sets (1)-(3) define the temporal domain in which all operations must occur, restricting starting times, finishing times, and the makespan to lie within the scheduling horizon. Constraint set (4) links the starting and finishing times of each operation by fixing the operation's duration. Constraint set (5) represents the technological precedence constraints between operations within each job. For every job $i \in J$, the operation O_{ij} must finish before the next operation $O_{i,j+1}$ can begin. Constraint set (6) enforces machine capacity constraints for production operations. For each machine $m \in M$, the time intervals of all production operations $O_{ij} \in P$ requiring that machine ($m_{ij} = m$) must not overlap, ensuring that a machine can process at most one operation at a time and cannot perform multiple operations simultaneously. Constraint set (7) ensures that any transportation operation $O_{ij} \in T$ cannot begin until the robot has had enough time to travel from the depot to the machine where the job is located; this constraint is only relevant when the transportation task is the first operation performed by that robot, and it is automatically satisfied with no practical effect otherwise. Constraint set (8) ensures that each transportation operation $O_{ij} \in T$ is assigned to exactly r_{ij} robots, as required by the cooperative transportation task. Constraint set (9) enforces robot capacity constraints for transportation operations. If two transportation operations O_{ij}, $O_{kp} \in T$ are assigned to the same robot r ($A_{ijr} = A_{kpr} = 1$), then they cannot be performed simultaneously and their time intervals must not overlap, and an appropriate empty travel time must be inserted between them. This constraint expresses a temporal disjunction for robot usage: either O_{ij} is scheduled before O_{kp} and in that case O_{kp} cannot start until O_{ij} finishes and the shared robots have traveled empty to the pickup location of O_{kp}, or the reverse must hold. Constraint set (10) defines the makespan as an upper bound of the completion times of the last operation of each job.

4) Objective

The objective is to minimize the makespan:

$$\text{Minimize} \quad C_{max}$$

B. OPL Implementation

The mathematical model presented in the previous subsection is implemented in OPL (Optimization Programming Language) using IBM ILOG CP Optimizer. The complete OPL model is presented in Fig. 1. The instruction at line 1 specifies that the model uses CP Optimizer as its solving backend. Our CP formulation relies on the modeling primitives of CP Optimizer, in particular interval variables, optional intervals, sequence variables, and global scheduling constraints. These concepts, together with their semantic properties and propagation mechanisms, are described in detail in [9], which provides the theoretical foundation for the scheduling capabilities of CP Optimizer. In Fig. 1, the section between line 2 and line 16 corresponds to the declaration and reading of input data. The number of jobs n is read from the data file at line 2, the number of machines nm at line 3 and the number of robots nr at line 4. Line 8 defines an array o to store the total number of operations for each job, while lines 5, 6, 7 and 9 define the index ranges used throughout the model. Line 10 defines an array t that indicates the type of each operation (0 for production and 1 for transportation). Line 11 defines an array m to store the machine required for each production operation, and line 12 defines an array r to store the number of robots required for each transportation operation. Line 13 defines an array l to store the travel time between any source and destination machine, including the depot. The duration of each operation is stored in an array d at line 14. For a transportation operation O_{ij}, this duration corresponds to the travel time from machine $m_{i,j-1}$ to machine $m_{i,j+1}$. Line 15 defines a transition matrix tm to store the unloaded travel time required between any two transportation operations. More precisely, for two transportation operations O_{ij} and O_{kp}, the entry $tm[i][j][k][p]$ corresponds to the empty travel time required for a robot to move from the delivery location of O_{ij} to the pickup location of O_{kp}, i.e., $tm[i][j][k][p] = l_{m\_\{i,j+1\},m\_\{k,p-1\}}$. This matrix is later used to determine the unloaded travel time that must be inserted between two consecutive transportation operations assigned to the same robot.

Figure 1. Full OPL implementation of the CP model for the JSPT-C.

Model 1 – OPL model for JSPT-C

```
1:  using CP;
2:  int n = ...;
3:  int nm = ...;
4:  int nr = ...;
5:  range Jobs = 1..n;
6:  range Machines = 0..nm;
7:  range Robots = 1..nr;
8:  int o[Jobs] = ...;
9:  range Ops[i in Jobs] = 1..o[i];
10: int t[i in Jobs][j in Ops[i]] = ...;
11: int m[i in Jobs][j in Ops[i]] = ...;
12: int r[i in Jobs][j in Ops[i]] = ...;
13: int l[s in Machines][d in Machines] = ...;
14: int d[i in Jobs][j in Ops[i]];
15: int tm[i in Jobs][j in Ops[i]][k in Jobs][p in Ops[k]] = ...;
16: dvar interval I[i in Jobs][j in Ops[i]] size d[i][j];
17: dvar interval Ir[i in Jobs][j in Ops[i]][r in Robots] optional;
18: dvar sequence Sr[r in Robots] in { Ir[i][j][r] | i in Jobs, j in Ops[i]:
19:                                   t[i][j] == 1};
20: dexpr int Cmax = max(i in Jobs) endOf(I[i][0[i]]);
21: minimize Cmax;
22: subject to {
23:   forall(i in Jobs, j in 1..(o[i]-1)) {
24:     endBeforeStart(I[i][j], I[i][j+1]);
25:   }
26:   forall(m in Machines:m > 0) {
27:     noOverlap( { I[i][j] | i in Jobs, j in Ops[i]:
28:                  t[i][j] == 0 && m[i][j] == m } );
29:   }
30:   forall(i in Jobs, j in Ops[i]:t[i][j] == 0, r in Robots) {
31:     presenceOf(Ir[i][j][r]) == 0;
32:   }
33:   forall(i in Jobs, j in Ops[i]:t[i][j] == 1) {
34:     startOf(I[i][j]) >= l[0][m[i][j-1]];
35:     sum(r in Robots) presenceOf(Ir[i][j][r]) == r[i][j];
36:     forall(r in Robots) {
37:       startsAtStart(Ir[i][j][r], I[i][j]);
38:       endsAtEnd(Ir[i][j][r], I[i][j]);
39:     }
40:   }
41:   forall(r in Robots) {
42:     noOverlap(Sr[r], tm);
43:   }
44: }
```

The decision variables of the model are declared in the section between line 16 and line 20. Line 16 defines an array I of interval variables indexed by job i and operation j. Each interval I_{ij}, represents the time window during which the operation O_{ij} is executed and has a size fixed to the operation's duration, corresponding to constraint (4) in the mathematical model. Line 17 defines an array Ir of optional interval variables indexed by job i, operation j and robot r. Each interval Ir_{ijr} corresponds to the time window during which robot r performs transportation operation O_{ij}, provided that the operation is assigned to that robot. Because these variables are optional, the solver decides the presence of each interval Ir_{ijr}, thereby encoding the robot assignment: If Ir_{ijr} is present then transportation operation O_{ij} is assigned to robot r; otherwise, it is not. Lines 18-19 define an array Sr of sequence variables, one for each robot r. Each sequence Sr_r contains the optional interval variables Ir_{ijr} corresponding to the transportation operations O_{ij} that robot r can perform and may be assigned to. This sequence models the ordered list of tasks that robot r may participate in, enabling CP Optimizer to impose non-overlapping robot usage and to account for the unloaded travel times between consecutive transportation operations, as specified in the transition matrix tm. Line 20 defines the expression C_{max} as the maximum completion time among the last operations of all jobs which corresponds to constraint set (11). This value becomes the objective to minimize, as stated in line 22.

The other constraints of the model are specified in the section between line 22 and line 44. Lines 23-25 impose the precedence constraints within each job, corresponding to constraint set (5) in the mathematical model. Lines 26-29 represent the machine disjunctive constraints for production operations, corresponding to constraint set (6). Lines 30-32 ensure that the interval variables Ir_{ijr} are defined and may be present only if O_{ij} is a transportation operation ($t_{ij} = 1$). Lines 33-40 specify the constraints associated with transportation operations. Line 34 corresponds to constraint set (7), while line 35 corresponds to constraint set (8) by enforcing that exactly r_{ij} interval variables Ir_{ijr} must be present for each transportation operation O_{ij}, meaning that that r_{ij} robots must participate in the operation. Lines 36-39 enforce the synchronization constraints for transportation operations: If the optional interval variable Ir_{ijr} is chosen to be present by the solver then it means that robot r participates in transportation operation O_{ij}, and in that case its execution interval Ir_{ijr} must be synchronized with the operation interval I_{ij}, meaning that both intervals start and end at the same time. This ensures that any robot used for a transportation task remains engaged for its entire duration. Finally, lines 41-43 represent the robot disjunctive constraints for transportation operations, corresponding to constraint set (9). For each robot r, the instruction at line 17 ensures that the transportation tasks assigned to that robot do not overlap in time, and that the appropriate unloaded travel time is inserted between any two consecutive tasks performed by that robot, as specified in the transition matrix tm.

All the experiments were performed using the default search strategy of CP Optimizer. No custom search phases or variable ordering heuristics were introduced. Investigating dedicated search strategies that prioritize machine sequencing or robot assignment decisions constitutes an interesting direction for future research.

IV. Numerical Study

This section presents the computational experiments conducted to evaluate the proposed CP formulation for the JSPT-C and to compare its performance with the existing MILP model and GRASPxELS metaheuristic for the same problem from literature. The goal of these experiments is to assess the model's effectiveness in producing optimal or near-optimal schedules and to determine whether constraint programming constitutes a more suitable solution method for this problem. All experiments are conducted on a laptop equipped with an Intel® Core™ i7-1365U CPU @ 1.80 GHz and 32 GB of RAM. The CP model was implemented in C++ and solved using the IBM ILOG CPLEX Optimization Studio package with the CP Optimizer engine. To ensure reproducibility and facilitate future research, all benchmark instances, detailed computational results, and corresponding solutions are publicly available at: https://perso.isima.fr/~amoussam/index_files/pages/Research/JSPT-C.html

A. Notations

The following notations are used in Tables I and II:

- T_{opt} : Time (in seconds) to proven optimality, if reached. The symbol "/" indicates that optimality was not proven.
- T_{best} : Time (in seconds) at which the best solution found is first obtained.
- *GAP* : Relative percentage difference between the best-found feasible solution and the optimal solution.
- *GAP (vs MILP 60s)* : Percentage gap of the CP solution relative to the MILP solution computed under a computational time limit of 60 seconds. The symbol '/' indicates that the gap cannot be calculated because no reference MILP solution was found.
- *GAP (vs MILP 1hr)* : Percentage gap of the CP solution relative to the MILP solution computed under a computational time limit of 1 hour. The symbol '/' indicates that the gap cannot be calculated because no reference MILP solution was found.
- *GAP (vs GRASPxELS)* : Percentage gap of the CP solution relative to the GRASPxELS solution.

B. Comparative study with [5] on medium-scale instances

The proposed CP formulation is evaluated on 50 medium-scale JSPT-C instances from [5], adapted from the classical JSPT dataset of [10]. Each of the instances is designated by the prefix EX followed by two digits indicating the job-set and layout, and then the suffix C with a digit specifying the

number of robots. These instances involve between 5 and 8 jobs, 4 machines and between 2 and 10 robots. For example, EX11C5 designates the JSPT-C instance with job-set 1 and layout 1, featuring 5 robots and transportation operations that require multiple robots simultaneously.

The GRASPxELS approach proposed for the JSPT-C in [5] is based on the GRASPxELS framework introduced in [11], which combines a Greedy Randomized Adaptive Search Procedure (GRASP) [12] with an Evolutionary Local Search (ELS) [13, 14]. GRASP generates diversified solutions through randomized greedy construction and local search, while ELS explores the attraction basins of local optima through mutation and local search operators.

To assess the performance of our CP approach, we compare it with both the MILP and GRASPxELS approaches reported in [5]. Since those methods were evaluated on the same laptop configuration and under a computational time limit of 65 seconds, we apply the same computational time limit to our CP model, ensuring that the results are directly comparable without requiring any speed normalization factor. The average results obtained on the 50 medium-scale instances are summarized in Table I.

TABLE I. AVERAGE PERFORMANCE ON THE 50 MEDIUM-SCALE JSPT-C INSTANCES

Method	Avg. GAP (%)	Avg. T_{best} (s)	Avg. T_{opt} (s)
MILP	0.00	-	4.72
GRASPxELS	1.19	7.76	-
CP	0.00	1.37	2.80

Both the CP and the MILP models achieve optimality for all instances, but the CP approach reaches optimality more quickly, with an average time of 2.80 seconds, approximately 1.7 times faster than the MILP model, which requires 4.72 seconds on average. Table I also shows that while the GRASPxELS metaheuristic achieves an average gap to optimality of 1.19% in 7.76 seconds, it is significantly outperformed by our CP approach, which reaches a 0% gap and finds its best solution in only 1.37 seconds on average, making it approximately 5.8 times faster than the metaheuristic.

Although both exact methods prove optimality for all medium-scale instances, the MILP formulation exhibits increasing computational times on instances involving larger robot fleets, particularly those with 10 robots. This behavior can be explained by the larger number of potential robot teams, assignment decisions, and synchronization constraints that must be considered. In contrast the CP approach maintains consistently low times to proven optimality across the benchmark instances, even as the number of available robots increases, indicating that CP Optimizer is less sensitive to the combinatorial growth induced by additional robot assignment and synchronization decisions. Detailed instance-by-instance results are available on the publicly available project webpage.

C. Comparative study with [5] on large-scale instances

The CP model is also evaluated on 9 large-scale JSPT-C instances from [5], designated with prefix JL followed by the instance number. These instances involve between 8 and 20 jobs, 8 to 12 machines, and 5 to 15 robots, with a total number of operations to schedule ranging from 128 to 480. We evaluate the CP model under a time limit of 60 seconds, and we compare its performance with the results reported in [5] for (1) their MILP model under 60 seconds, (2) the same MILP model under 1 hour, and (3) their GRASPxELS metaheuristic under 60 seconds. The average results obtained on the 9 large-scale instances are summarized in Table II.

TABLE II. AVERAGE PERFORMANCE ON THE LARGE-SCALE JSPT-C INSTANCES

Metric	Value
Avg. *GAP (vs MILP 60s)* (%)*	-26.6%
Avg. *GAP (vs MILP 1hr)* (%)*	-23.8%
Avg. *GAP (vs GRASPxELS)* (%)	-17.8%
Avg. T_{best} *(GRASPxELS)* (s)	36.95 s
Avg. T_{best} *(CP)* (s)	45.33 s

*. Average computed only over instances for which the corresponding MILP solution was available

As shown in Table II, the CP model provides high-quality solutions on the large-scale benchmark instances within the computational time limit. Compared with the MILP model evaluated under the same 60 second time limit, the CP approach achieves average improvements of 26.6%. Moreover, the MILP model is only able to find feasible solutions for a small subset of the large-scale benchmark instances, indicating that the problem size rapidly overwhelms its search process. Even when granted 1 hour of computational time (i.e., 60 times more than the time allocated to the CP model), the MILP formulation still produces solutions that are significantly worse than those obtained by CP, with average improvements of 23.8%. The CP model also outperforms the GRASPxELS metaheuristic, achieving an average improvement of 17.8%. Although the metaheuristic finds its best solutions more quickly on average (36.95 seconds versus 45.33 seconds for CP), the quality of the solutions obtained by the CP approach remains substantially better. Detailed instance-by-instance results are available on the project webpage.

D. Discussion

These trends reflect fundamental differences in how each method scales with problem size. MILP must explicitly encode every robot-operation assignment possibility through binary variables. As the number of jobs, machines and robots increases, the linear relaxation weakens significantly. Large-scale instances therefore generate an enormous branch-and-bound tree, and the solver struggles to prune inefficient regions early because the Linear Programming relaxations provide limited guidance. Furthermore, synchronization between multiple robots requires additional linear constraints that do not propagate strongly within the MILP framework, leading to weaker relaxations and slower convergence. As a result, MILP often cannot identify even one feasible solution within short time limits and remains far from high-quality solutions even after 1 hour of computation. Although the metaheuristic is fast at generating feasible schedules, it faces

a rapid growth of the search space on large instances. As the number of robots increases, the number of possible robot teams expands combinatorially. This dilutes the effect of local search operators, slows convergence, and frequently leads to stagnation before high-quality regions of the search space can be explored. In contrast, the CP formulation relies on optional interval variables and powerful global constraints such as *noOverlap* and interval synchronization (*startsAtStart*, *endsAtEnd*,…). These constructs allow CP Optimizer to propagate temporal and resource constraints very efficiently. Increasing the number of robots does not introduce combinatorial branching at a variable level; instead, it provides more optional intervals and more opportunities for propagation. The solver can quickly eliminate infeasible robot assignments through constraint propagation without enumerating all possibilities. Moreover, cooperation and synchronization constraints are handled natively through interval alignment, which is far more expressive and computationally efficient than linear constraints. As a result, CP Optimizer scales significantly better than both MILP and GRASPxELS and maintains robust and consistent performance even as instance size increases. An interesting observation is that the synchronization requirements introduced by cooperative transportation affect the different solution approaches in fundamentally different ways. In the MILP formulation, robot cooperation generates additional binary assignment decisions and synchronization constraints that weaken the linear relaxation and increase the difficulty of the branch-and-bound search. In contrast, the CP formulation benefits from stronger propagation opportunities generated by optional interval variables and synchronization constraints, allowing many infeasible robot assignments to be eliminated early during the search process.

V. Conclusion and Future Research

This paper introduced the first constraint programming model for the Job-Shop Scheduling Problem with Cooperative Transportation Resources (JSPT-C). Computational experiments demonstrate the effectiveness of the proposed approach. On medium-scale instances, the CP model proves optimality for all benchmark instances while requiring less computational time than the existing MILP formulation and consistently outperforming the GRASPxELS metaheuristic. On large-scale instances, the CP approach achieves substantially better solution quality than competing methods, even when the MILP formulation is granted significantly longer computational times. These results highlight the scalability of the proposed CP model and suggest that constraint programming constitutes a particularly effective exact solution approach for the JSPT-C.

Several research directions emerge from this work. From a methodological perspective, the proposed CP model could be further enhanced through the design of dedicated search strategies or hybrid approaches combining constraint programming with MILP or metaheuristic frameworks. In particular, CP could be used as a powerful improvement or post-optimization phase within hybrid solution methods. Another promising direction concerns the study of stochastic variants of the JSPT-C, where uncertainties such as machine breakdowns, robot failures, or congestion may affect processing and travel times. Integrating the CP model within rolling-horizon or reactive scheduling frameworks therefore represents an interesting avenue for future research. Although the proposed CP model scales effectively on the benchmark instances considered in this study, its behavior on substantially larger instances involving several dozens of jobs, hundreds of transportation operations, or larger robot fleets may eventually generate a significantly larger number of optional intervals and synchronization constraints, potentially increasing the difficulty of the search process. Evaluating the performance of the proposed CP formulation on such larger-scale instances constitutes an interesting direction for future research.

Acknowledgment

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

References

[1] H. Xiong, S. Shuangyuan, R. Danni and H. Jinjin, “A survey of job shop scheduling problem: The types and models,” *Computers & Operations Research*, vol. 142, Art. No. 105731, 2022.

[2] H. E. Nouri, O. B. Driss and K. Ghédira, “A classification schema for the job shop scheduling problem with transportation resources : state-of-the-art review,” in *Artificial Intelligence Perspectives in Intelligent Systems: Proc. 5th Computer Science On-line Conf. (CSOC)*, vol. 1, pp. 1-11, 2016.

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

[4] J.-P. Gayon, P. Lacomme and A. Oussama, “Job-Shop Scheduling with Robot Synchronization for Transport Operations,” in *Metaheuristics International Conference*, Cham, Switzerland: Springer Nature, pp. 28-42, 2024.

[5] J.-P. Gayon, P. Lacomme and A. Oussama, “Job-shop scheduling with cooperative transportation resources,” *International Transactions in Operational Research*, 2025.

[6] E. B. Edis and I. Ozkarahan, “A combined integer/constraint programming approach to a resource-constraint parallel machine scheduling problem with machine eligibility restrictions,” *Engineering Optimization,* vol. 43, no. 2, pp. 135-157, 2011.

[7] A. Ham, “Transfer-robot task scheduling in job-shop,” *International Journal of Production Research*, vol. 59, no. 3, pp. 813-823, 2021.

[8] M. R. Garey, D. S. Johnson and R. Sethi, “The complexity of flowshop and jobshop scheduling,” *Mathematics of operations research*, vol. 1, no. 2, pp. 117-129, 1976.

[9] P. Laborie, “IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems,” in *Proc. Int. Conf. on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR),* Heidelberg, Germany: Springer, pp. 148-162, 2009.

[10] Ü. Bilge and G. Ulusoy, “A time window approach to simultaneous scheduling of machines and material handling system in an FMS,” *Operations Research*, vol. 43, no. 6, pp. 1058-1070, 1995.

[11] C. Prins, “A GRASPx evolutionary local search hybrid for the vehicle routing problem,” in *Bio-inspired algorithms for the vehicle routing problem*, Berlin, Heidelberg, Germany: Springer, pp. 35-53, 2009.

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

[13] S. Wolf and P. Merz, “Evolutionary local search for the super-peer selection problem and the p-hub median problem,” in *International workshop on hybrid metaheuristics*, Berlin, Heidelberg, Germany: Springer, pp. 1_15, 2007.

[14] C. Prins, “A simple and effective evolutionary algorithm for the vehicle routing problem,” *Computers & operations research*, vol. 31, no. 12, pp. 1985-2002, 2004.