applied sciences

MDPI

Article

Dynamic Job and Conveyor-Based Transport Joint Scheduling in Flexible Manufacturing Systems

Sebastiano Gaiardelli [1,†], Damiano Carra [1,*,†], Stefano Spellini [1] and Franco Fummi [2]

1 Department of Computer Science, University of Verona, 37134 Verona, Italy
2 Department of Engineering for Innovation Medicine, University of Verona, 37134 Verona, Italy
* Correspondence: damiano.carra@univr.it
† These authors contributed equally to this work.

Abstract: Efficiently managing resource utilization is critical in manufacturing systems to optimize production efficiency, especially in dynamic environments where jobs continually enter the system and machine breakdowns are potential occurrences. In fully automated environments, co-ordinating the transport system with other resources is paramount for smooth operations. Despite extensive research exploring the impact of job characteristics, such as fixed or variable task-processing times and job arrival rates, the role of the transport system has been relatively underexplored. This paper specifically addresses the utilization of a conveyor belt as the primary mode of transportation among a set of production machines. In this configuration, no input or output buffers exist at the machines, and the transport times are contingent on machine availability. In order to tackle this challenge, we introduce a randomized heuristic approach designed to swiftly identify a near-optimal joint schedule for job processing and transfer. Our solution has undergone testing on both state-of-the-art benchmarks and real-world instances, showcasing its ability to accurately predict the overall processing time of a production line. With respect to our previous work, we specifically consider the case of the arrival of a dynamic job, which requires a different design approach since there is a need to keep track of partially processed jobs, jobs that are waiting, and newly arrived jobs. We adopt a total rescheduling strategy and, in order to show its performance, we consider a clairvoyant scheduling approach, in which job arrivals are known in advance. We show that the total rescheduling strategy yields a scheduling solution that is close to optimal.

Keywords: scheduling; heuristic; makespan minimization

check for updates

Citation: Gaiardelli, S.; Carra, D.; Spellini, S.; Fummi, F. Dynamic Job and Conveyor-Based Transport Joint Scheduling in Flexible Manufacturing Systems. *Appl. Sci.* **2024**, *14*, 3026. https://doi.org/10.3390/app14073026

Academic Editors: Zoran Jurković, David Ištoković and Janez Gotlih

Received: 20 December 2023
Revised: 25 March 2024
Accepted: 29 March 2024
Published: 3 April 2024

1. Introduction

Industry 4.0 has significantly reshaped manufacturing system paradigms, emphasizing the move toward the complete automation of the production process [1,2]. In this transformative context, where the goal is to fully exploit the capabilities of machines, the precise planning of all production *tasks* becomes imperative, aiming to minimize machine idle time. Consequently, the scheduling problem has been the subject of extensive study in recent decades [3–5]. In its simplest form, this scheduling challenge is commonly referred to as the job shop scheduling (JSS) problem [6]. Given a set of *machines* and a set of *jobs*, where each job comprises a set of *tasks* to be processed in a specific order by different machines, the objective is to find a task assignment that minimizes an objective function, such as the total completion time or *makespan*.

The above problem formulation assumes that the time is divided into slots (e.g., each day is a time slot): the job requests are collected during a time slot and scheduled for the next one. By relaxing this assumption, we can insert the jobs as they arrive into the schedule. This scenario is usually referred to as *dynamic JSS* (DJSS). With the evolution of manufacturing technology, in which a single machine may perform multiple task types, we can further extend JSS to include this flexibility; in such a case, we have the flexible

DJSS problem (FDJSS or simply FJSS). The FJSS problem has been particularly interesting in recent years in the context of *Industry 4.0* [7], in which the production lines include advanced features that allow the machines to communicate and be reorganized and reconfigured to meet the increasingly challenging production constraints.

A crucial aspect that is not fully explored in the existing literature is the automation of *job transfers* between machines. While task execution on machines can be characterized by processing times, the transfer process involves various settings, including (i) the means of transportation, such as automatic guided vehicles (AGVs) or conveyor belts, and (ii) input and output buffers at the machines, determining their ability to store jobs for processing (e.g., while finishing another task). The store processed jobs (while awaiting the availability of the transport system) (Our definition of *transfer* pertains to the time between the end of one task and the start of the next for a given job).

Certain combinations of transport systems and buffers may be modeled using constant time [8], as is the case when the transport facility is always available, and the machines have sufficiently large input and output buffers. In such scenarios, incorporating transfer time into the processing time is straightforward, and numerous solutions proposed in the literature [4] can be applied. In other cases, transfer time depends on the availability of the transport system and the destination machine [1,9,10]. Overall, no single model can cover all alternatives, necessitating ad-hoc modeling for specific combinations.

In this study, we consider a scenario inspired by a *fully automated production line*, incorporating a conveyor belt as a means of transportation and no buffer at the machines. In this setup, once a task is completed, the job is immediately placed on the belt to free the machine. If the next machine is occupied, the job remains on the belt until the destination machine becomes available. The JSS problem is well-known to be NP-hard [11], and the variant considered here is at least as challenging [12]. In order to address this complexity, we propose a heuristic named SCHED-T, which falls under the *stochastic local search* (SLS) approach [13]. SLS encompasses well-known algorithms such as simulated annealing and tabu search.

A primary challenge is evaluating potential moves when exploring the solution space. A slight change in the scheduling sequence has a cascading effect on the remaining tasks, as the sequence depends on transfer times, which, in turn, depend on the execution sequence. While existing solutions rely on an approximate evaluation of each move (e.g., the computation of the critical path), we adopted a randomized approach, accurately assessing a few random neighbors. Given SCHED-T's ability to quickly compute complex schedules, especially in the case of dynamic job arrivals, we employed the "total rescheduling" strategy [14]. This strategy considers newly arrived jobs alongside those not yet scheduled, resulting in a new schedule that integrates with the current one. This approach is versatile, accommodating events beyond job arrivals, such as scheduled machine maintenance [15].

Here, we evaluate SCHED-T on a set of instances publicly available without transfer times [16]. In such cases, our heuristic achieves results comparable to those of the previously proposed versions. Subsequently, we analyze instances generated from a production line where the transfer times are available. Although the problem can be modeled using mixed linear integer programming (MILP) approaches, standard MILP solvers fail to find a solution in a reasonable time. Consequently, SCHED-T emerges as the only viable approach. Our results demonstrate that when applying the schedule found by SCHED-T to a real-world production line, the predicted makespan closely aligns with the actual outcome. Conversely, using a schedule derived without considering the transport system, as seen in the literature, results in a makespan that is up to 30% larger than that achieved using SCHED-T's schedule.

In our approach, we consider dynamic arrivals and compare the total rescheduling strategy using clairvoyant scheduling, i.e., a scheduling type wherby future job arrivals are known in advance, and show that the total rescheduling strategy produces close-to-optimal scheduling.

This article builds upon the findings presented in [17] in several ways. Specifically, we delve into the scenario of dynamic arrivals, where the schedule must be recalculated, which is in contrast to the static case examined in [17], where all jobs were available at the onset of the scheduling period. This necessitated a redesign of the scheduler to manage partially processed, waiting, and newly arrived jobs. Consequently, this updated definition encompasses the static case as a special instance, thus representing a generalization of the problem. Furthermore, we present a clear system model and problem formulation, along with an expanded comparison of our heuristics. This comparison includes additional instances, additional performance metrics, such as the running time of our scheduler, and diverse layouts for real-world experiments.

In summary, the contributions of our work are the following. We provide a heuristic for the JSS problem that takes into account the transport system based on a conveyor belt and no buffers at the stations. We consider a dynamic scenario where jobs continuously arrive while we managed the partially processed jobs and the newly arrived jobs, making for a total rescheduling strategy. We tested our solution in a real-world production line to show that if transport times are not considered, the scheduler produces results that may contain large errors.

In summary, the contributions of our work are as follows:

- We propose a heuristic for the Job Shop Scheduling (JSS) problem that incorporates a transport system based on a conveyor belt and no buffers at the stations.
- We address a dynamic scenario where jobs continuously arrive while we effectively manage partially processed jobs and newly arrived jobs by adopting a total rescheduling strategy.
- We validated our solution through experimentation on a real-world production line, demonstrating that neglecting transport times can lead to substantial errors in the scheduler's results.

The paper is structured as follows. In Section 2, we present the case study that serves as the motivation for our work. Specifically, we examine a production line where machines are interconnected through a conveyor belt. We then delve into the existing body of related work and highlight their limitations. Section 3 is dedicated to formalizing the problem, introducing a model that defines the key variables. This formalization enables us to cast the problem as a minimization challenge. Given the impracticality of finding an exact solution within a reasonable timeframe, Section 4 outlines our proposed heuristic, SCHED-T. This heuristic belongs to the class of *randomized iterated improvement* algorithms tailored to address the nuances of the specific problem. In Section 5, we conduct a comparative analysis of the solutions obtained with SCHED-T against those generated by state-of-the-art heuristics. We explore scenarios involving the transport system that were previously unsolvable with tools available in the literature. Finally, Section 6 encapsulates the key findings and draws conclusions from our study.

2. Background and Related Work

Production line with a conveyor belt: The manufacturing system, taken as a reference, consists of a set of machines dedicated to specific tasks disposed in a general *layout*. Each machine is connected with the others through a transport system that moves the materials between the machines circularly. Both raw and finished materials are stored in a vertical warehouse. The production line available at our research facility, *the ICE Laboratory* [18], follows the same layout. Figure 1 shows the plant configuration. It is composed of a set of "production cells", specifically tailored to a specific manufacturing process. Starting from the right, the laboratory includes a multi-tool milling machine, a robotic assembly station, a quality control cell, and a vertical warehouse. The conveyor belt moves materials on top of 10 pallets across various *belt segments*. Each pallet is identified by an RFID tag, which is detected by RFID sensors located near the switching mechanisms (blue squares in Figure 1). In particular, the belt segments can be differentiated into the following:

- *The four machine segments*: These are in charge of loading and unloading materials from the machines (i.e., moving pallets toward and away from the processing area). Since

these segments are the only access route to the machines, they can contain only one pallet at a certain time instance;
- *The main belt segments*: These are in charge of moving a pallet near the machine when it is ready to process a new task. The pallets waiting for a machine to become free must loop near the destination machine, implementing a circular buffer, while free pallets loop along the long main segments. The main belt is composed of two long belts running in opposite directions. Along these main segments, it is possible to traverse from one to the other by exploiting the switching mechanisms placed in certain positions.

The two types of belt segments differ in terms of their control policy. Machine segments are activated in the desired direction to perform loading or unloading pallet operations, and the main belt segments are always active, pushing in the same direction. A switching mechanism can move a pallet by choosing one of the following directions: (1) forward, following the direction of the actual belt; (2) backward, changing direction by switching onto the opposite conveyor; and (3) towards a machine working area. If two pallets must move in the same direction at the same time, the pallet exiting from an unloading segment or moving forward along the main belt has precedence.

Literature review: In the last decades, the JSS problem has been extensively analyzed by the research community, along with the different variants, such as DJSS and FJSS. The solutions that have been proposed over the years are summarized in [4,5] and [19]. Our work differs from the standard problem because it specifically addresses the transport time between machines and the impact of this on the schedule. Therefore, we concentrated our analysis of the related works on this specific topic.

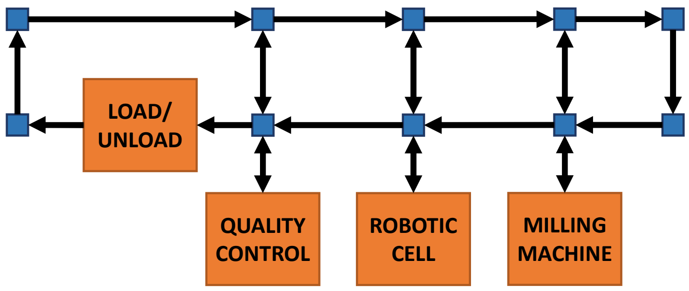

Figure 1. The layout of the production line used as a case study.

The transfer time between two machines is a variable delay that depends on the distance between the two machines and the availability of the destination machine. A solution proposed in [20] handles variable delays as sequence-dependent setup times. However, this solution considers the machines busy during the setup time, whereas, in our case, the destination machine can process other tasks while the materials of the next task are moving toward the machine.

The authors in [10,21–23] proposed a solution for the problem of job scheduling combined with transport time, but they consider only automated guided vehicles (AGVs) as a means of transportation. The limit of considering AGVs is that until a vehicle picks up a task from a machine, the machine itself cannot execute other activities. Similarly, when an AGV is waiting to perform a pick operation from a machine, it wastes transport resources. Moreover, if the machine is completing the previous task when the AGV reaches the destination, it must wait until the machine completes the task, freeing its working area. This is not a valid scenario in our case study, in which the machine immediately unloads the task to the conveyor belt once the task is completed. Another difference is that the jobs waiting to be processed by a machine do not impact on the unloading and loading capabilities of the tasks of other machines.

The aim of the different heuristics proposed in the literature is to find the best schedule that optimizes an objective function [12,20,24]. These state-of-the-art solutions are based on graph-based representations of the execution sequence. By evaluating the graph properties, such as dependencies and the critical path, it is possible to infer the feasibility of a solution by also approximating the objective function. In our case, these approaches cannot be applied due to the inter-dependency between transfer time and the execution sequence. There are also solutions, such as [16], based on MILP optimization models, which minimize the objective function. A major drawback of these approaches consists of the execution time. In fact, by modeling the constraints of our transport system, solving even small instances cannot be carried out in a reasonable amount of time.

In summary, none of the existing works take into account the influence of a transport system on comprehensive scheduling when machines lack input and output buffers. In this specific setup, the sequence of jobs affects the transport times, and conversely, the transport times impact the job sequence. This intricate circular dependency necessitates specialized tools that are currently absent in the literature.

3. Problem Description

3.1. Model

The flexible job shop problem concerning no buffers and a transportation system can be described as follows. The facility has a set of m machines ($M = \{1, 2, \ldots, m\}$) that are used to process a set of n jobs ($J = \{1, 2, \ldots, n\}$). Since we consider a dynamic environment in which jobs continuously arrive, the set of jobs ($J(t)$) depends on the time (t), i.e., we have $n(t)$. Such a dynamic set is composed of (i) the jobs that have arrived but have not been started yet and (ii) the newly arrived jobs.

Each job (i) comprises h_i operations or *tasks* (τ_{ij}, where i denotes the job, and $j = 1, 2, \ldots, h_i$), and each task can be executed on a subset of machines. The processing time (p_{ijk}) for a given task (j) of a job (i) on a machine (k) is considered known. If a task (τ_{ij}) cannot be performed on machine (k), we set $p_{ijk} = \infty$. The tasks within a job may have *precedence constraints*, meaning that a task cannot start until its preceding tasks (if any) are completed. For each job (i), the precedence constraints are summarized by a square matrix (U_i). An element ($U_{ijj'}$) is set to 1 if task τ_{ij} precedes task $\tau_{ij'}$, and this is 0 otherwise.

Each machine (k) is associated with a set of available times ($A_k(t)$), representing intervals during which tasks can be executed on that machine. Outside these intervals, the machine is unavailable either because it is busy with other tasks (e.g., a previous schedule is still being executed) or because it is undergoing maintenance. Each machine performs, at most, one task of any job at a time. With each new arrival, the availability sets for all machines are recomputed, considering the currently running jobs.

The time required to move a job from machine k to machine l, denoted by t_{kl}, is a key aspect of our work, particularly as we assume that machines lack input or output buffers and a conveyor belt serves as the means of transportation. More specifically,

$$t_{kl} = t^0_{kl} + t^c_{kl} \cdot n^c \tag{1}$$

Here, t^0_{kl} represents the minimum time to travel from machine k to machine l, t^c_{kl} is the cycle time in case machine l is unavailable, and n^c is the number of cycles the job needs to complete until the machine becomes available. These values are contingent on the layout of the production line (machines and conveyor belts) and can be readily measured once such a layout is defined. It is worth noting that by appropriately setting these values, we can encompass scenarios commonly explored in the literature: if $t^c_{kl} = 0 \; \forall k, l$, the transport time remains constant; if $t_{kl} = 0 \; \forall k, l$, the model excludes the transport system. Hence, our model is versatile, covering various layouts and accounting for cases where transport time is not considered.

Table 1 summarizes the notation used for the model and the formulation of the problem.

Table 1. Notation summary.

Inputs	
M	Set of machines, $\{1, 2, \ldots, m\}$
$J(t)$	Set of jobs $\{1, 2, \ldots, n(t)\}$
h_i	Number of tasks of job i
τ_{ij}	jth task of job i, $j \in [1, h_i]$
p_{ijk}	Processing time for τ_{ij} on machine k
U_i	Precedence matrix for job i: 1 if τ_{ij} precedes $\tau_{ij'}$; otherwise, it is 0
$A_k(t)$	Availability intervals for machine k
t_{kl}	Transfer time between machines k and l
Auxiliary Variables	
E_{ijk}	1 if τ_{ij} is executed on machine k; otherwise, it is 0
s_{ijk}	Execution start time of τ_{ij} on machine k
C_{ij}	Completion time of τ_{ij}

3.2. Problem Formulation

We consider a dynamic environment in which jobs continuously arrive. We adopt a total rescheduling strategy, i.e., we recompute the schedule at every job arrival; we discuss such a choice in detail in Section 4.4. The aim of the scheduling is to minimize the *makespan*, i.e., the total execution time required to process all the tasks of the current set of jobs $J(t)$. Formally, we have

$$\begin{aligned}
\underset{i \in [1, n(t)]}{\text{minimize}} \quad & \max \; C_{ij} && (2) \\
\text{s.t.} \quad & && \\
s_{ijk} + p_{ijk} = C_{ij} \quad & && (3) \\
s_{ijk} + p_{ijk} + t_{kl} \le s_{ij'l} \quad & \forall \tau_{ij}, \tau_{ij'} | U_{ijj'} = 1 && (4) \\
\sum_k E_{ijk} = 1 \quad & \forall \tau_{ij} && (5) \\
s_{ijk} + p_{ijk} < s_{i'j'k} \quad & \forall \tau_{ij}, \tau_{i'j'}, k \in [1, m] && (6) \\
\left[s_{ijk},\; s_{ijk} + p_{ijk} \right] \in A_k(t) \quad & \forall \tau_{ij},\; \forall k && (7)
\end{aligned}$$

Equation (3) specifies that the end time of each task must be equal to its start time plus its processing time. Here, we omit the necessary time to store the material produced by the last task of each job. Equation (4) models the dependencies between tasks within the same job. If a task (j) is assigned to a machine (k), it has precedence over another task (j') assigned to the machine (l); the starting time of the task (j') must be greater or equal to the end time of the task (j) plus the transport time between the two machines. Equation (5) imposes the maximum number of machines to which a task can be assigned, i.e., one. Equation (6) limits the number of tasks a machine can process in parallel. Finally, Equation (7) indicates that tasks can be executed on the machines only during their availability intervals.

The provided equations can be utilized to formulate and solve the constraints of a mixed linear integer programming (MILP) model. However, conventional MILP solvers, such as IBM CPLEX, face challenges in finding a solution within a reasonable time frame; more specifically, even with a modest number of jobs, they are unable to converge within a 12 h timeframe on standard hardware. This computational inefficiency arises due to the incorporation of transport time in Equation (4), where t_{kl} is contingent on the values derived from Equation (1), significantly escalating the computational complexity of the model. Consequently, we must turn to alternative methodologies that involve exploring the solution space.

4. Exploring the Solution Space

4.1. Overview

The general framework for any heuristic based on stochastic local search comprises the following steps [13]:

1. Construct a solution and compute the objective function;
2. Explore the neighborhood, assessing the objective function for each neighbor;
3. Choose the neighbor based on a specified criterion;
4. Repeat all steps starting from step 2 until a stop condition is satisfied.

In addition, given a solution, we must define (i) how we can explore its neighborhood (i.e., new solutions); (ii) how the objective function is computed (i.e., what are the parameters that we want to optimize); (iii) the policy used to choose the next solution; and (iv) when the exploration should terminate (i.e., stop conditions). The computational complexity of these approaches depends on the neighborhood size. For instance, given a task sequence (s_i) containing n elements, we may define (as a neighbor) any sequence (s_j) that differs from s_i for the position of two tasks (the minimum possible change). This would imply that the number of elements in the neighborhood is at least equal to $n(n-1)/2$.

Given the high number of neighbors used to explore each solution, a common procedure consists of limiting the exploration to the subset of the *most significant* neighbors. This set is composed of all the neighbors except those that are less likely to improve the current solution, e.g., all the neighbors that swap two tasks belonging to the same job since they have dependencies that limit the time interval in which they can be allocated.

The cost function must be computed for each explored neighbor. Therefore, its complexity heavily impacts the execution time. A cost function should be a trade-off between precision and complexity, exploring a wide range of neighbors in a certain time slot and choosing the most promising one. In our case, the complexity of the cost function is given by two main factors. The first one consists of the inner dependencies between tasks of the same job. This implies that small changes in one task can cause cascading effects on the subsequent portion of the schedule. The second factor is related to the transfer time between two machines. This transfer time is strictly dependent on both the availability of the destination machine (see Equation (1)) and the task execution order. While the first problem can be addressed by exploiting graph-based representations, the circular dependency between execution order and transport time makes it difficult to precisely or approximately compute the objective function.

As such, we are forced to compute the objective function for each neighbor explored from scratch, making the selection of the most promising neighbors to explore a critical step.

4.2. Randomized Approach

Given a set of jobs to schedule, the number of possible solutions increases exponentially as the number of jobs increases. Therefore, an entire exploration of the solution state space could require hours or could be unfeasible in a reasonable amount of time. For this reason, the exploration is always guided by heuristics that reduce the number of evaluated solutions. An idea inspired by the results proposed in [25–27] is to explore a set of neighbors chosen randomly. This is indeed a variant of the stochastic local search approach called *probabilistic random improvement*.

A random neighbor (i.e., solution) is described by the sequence of jobs. For each neighbor, we compute the allocation of tasks on the machines and evaluate the objective function. Given a solution, we consider its set of *randomly selected* neighbors. Then, we choose the best neighbor that improves the current solution. This process is repeated to increase the exploration's precision. Thus, we can also tune the precision of the exploration by increasing or decreasing the total number of random neighbors to explore.

During the exploration, we must avoid being stuck in a local minimum. We can do that by adopting known techniques used in tabu search or simulated annealing, in which a new solution is accepted with a certain probability and without considering the optimality degree of the solution. Instead, we adopt another methodology based on sampling, which is inspired by the fisheye view [28] and fisheye routing [29]. These techniques sample solutions at different distances to check if there are other *valleys* that exist to explore. In our context, a distant neighbor is obtained with more complex changes in the task execution order. In addition, for each remote neighbor, we perform a limited local exploration to test if such a remote neighbor could indeed improve the current solution. Figure 2 shows an example of the exploration process.

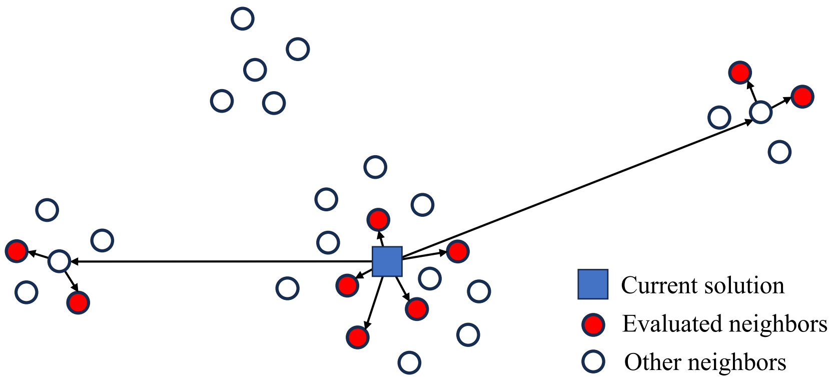

Figure 2. Example of the randomized exploration approach. We start from the current solution and evaluate some random close neighbors, along with remote neighbors (chosen randomly).

4.3. Detailed Solution

Our scheme for the exploration of the neighborhood uses a two-level hierarchical approach. At a higher level, we have jobs, while at the lower level, we have tasks.

Initialization: In the case of a cold start, in which the system is completely unloaded, and a set of jobs has been collected and are ready to be executed (e.g., during the night, the production line does not work and arriving orders are collected), we initialize the order of the jobs according to the *longest processing first* policy since it has been shown to improve the makespan compared other policies [30]. We define the processing time of a job as the sum of the processing time of its tasks. If a task (τ_{ij}) can be processed by more than one machine, then we consider the minimum (p_{ijk}) (see Algorithm 1, procedure INITTASKORDER, and parameter *BySize* set to TRUE). In the case of a running system, the order is given by the output of the previous schedule (considering only the jobs not yet launched), with the newly arrived job placed at the end.

We then establish the order of tasks within each job by grouping tasks with the same precedence and randomly arranging the tasks within each group. The final output is an ordered sequence of tasks, denoted as s_l. The makespan value is computed based on this sequence (refer to Algorithm 1 and procedure EVALSOLUTION). We allocate each task to a machine, considering machine availability and dependencies for previous tasks. If multiple machines can perform a task, we choose the machine where the task terminates earlier, including the transportation time.

Neighborhood: For a given sequence (s_l) of tasks, we define a *close neighbor* as the feasible solution where we work at a low level. In other words, we switch tasks to obtain a new sequence ($s_{l'}$) for evaluation. A random neighbor is obtained by (i) uniformly selecting a task (τ_{ij} in s_l) and (ii) uniformly selecting another task ($\tau_{i'j'}$). The latter must belong to the same job ($i' = i$) or to a job that comes before or after job i in the ordered set J. If the tasks belong to the same job, our choice is limited to tasks with no dependencies on task τ_{ij} (such a subset can be pre-computed upon job arrival).
We define a *remote neighbor* as a new solution in which jobs are swapped in the ordered set J (i.e., higher-level permutations). Thus, remote neighbors allow for the exploration of new solution areas, and close neighbors can be used to fine-tune the current solution.

Algorithm 1: Initialization and Evaluation

```
1  Procedure InitTaskOrder($J$, $BySize$):
2      if $BySize$ then
3          foreach job $i$ do
4              $i(size) \leftarrow \sum_j \min_k p_{ijk}$ ;
5          $J \leftarrow$ OrderBySize($J$);
6      $s_l \leftarrow \emptyset$;
7      foreach job $i \in J$ do
8          $G_i \leftarrow$ GroupByPrecedence($i$);
9          foreach group $g \in G_i$ do
10             $g \leftarrow$ RandomOrder($g$);
11             AddTasks($s_l, g$));
12     return $s_l$
13 Procedure EvalSolution($s, \{A\}$):
14     foreach task $\tau_{ij} \in s$ do
15         $q \leftarrow \arg\min_k(t_{lk} + p_{ijk})$;
16         Update($A, q, \tau_{ij}$);
17     return $\max C_{ij}$;
```

Exploration: The number of close and remote neighbors (solutions) explored during an iteration is governed by the budget, B. At each iteration, we update the current solution with a new one if it improves the current one (see Algorithm 2). The exploration starts by considering the current solution s_i and αB as close neighbors, chosen uniformly at random among the feasible close neighbors (lines 6–12). The parameter α is given as input, with $0 < \alpha < 1$. The exploration continues by evaluating R remote neighbors (lines 13–14), selected uniformly at random. For each of them, it explores its neighbors locally uniformly at random with a budget $(1-\alpha)B/R$ (lines 15–21).

The exploration stops when one of the following conditions is reached: (i) the maximum number of iterations $T_{\max}$; (ii) no improvements are found for t_{idle} iterations (not shown in Algorithm 2).

Figure 3 shows a high-level view of the processing flow.

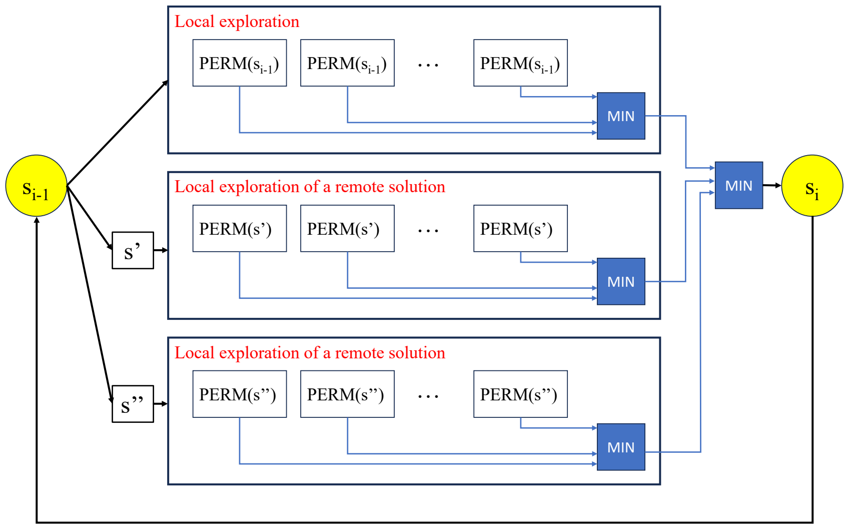

Figure 3. Exploration flow: starting from a solution at time $t-1$, we explore the local neighbors, along with a set of distant neighbors. If we find a better solution, we update the current solution.

Algorithm 2: SCHED-T

```
input: $J$, set of jobs to be scheduled
input: $M$, set of machines
input: $\{A_0\}$, availability intervals
input: $BySize$, True if jobs need to be ordered by size
input: $B, \alpha, R$, parameters for exploration
1  $s_0 \leftarrow$ InitTaskOrder($J$, $BySize$);
2  $\mathcal{O} \leftarrow$ EvalSolution($s_0, \{A_0\}$);
3  $i = 1$;
4  while $i \le T_{max}$ do
5      $s_i \leftarrow s_{i-1}$;
6      for $\alpha B$ times do
7          $s'_i \leftarrow$ RandLocalPerm($s_{i-1}$);
8          $\mathcal{O}' \leftarrow$ EvalSolution($s'_i, \{A_{i-1}\}$);
9          if $\mathcal{O}' < \mathcal{O}$ then
10             $s_i \leftarrow s'_i$;
11             $\mathcal{O} = \mathcal{O}'$;
12             $\{A_i\} \leftarrow$ UpdateAval($s_i$);
13     for $R$ times do
14         $s'_{i-1} \leftarrow$ RandRemotePerm($s_{i-1}$);
15         for $(1-\alpha)B/R$ times do
16             $s'_i \leftarrow$ RandLocalPerm($s'_{i-1}$);
17             $\mathcal{O}' \leftarrow$ EvalSolution($s'_i, \{A_{i-1}\}$);
18             if $\mathcal{O}' < \mathcal{O}$ then
19                 $s_i \leftarrow s'_i$;
20                 $\mathcal{O} = \mathcal{O}'$;
21                 $\{A_i\} \leftarrow$ UpdateAval($s_i$);
22     $i$++;
```

Complexity: When evaluating a single solution, since we build it from scratch, we run through the ordered list of tasks and assign the task to the available machine. Each task may be executed in more than one machine, but we assume that the number of alternative machines is bounded. The complexity of each evaluation, therefore, is $\mathcal{O}(H)$, with $H = \sum_i h_i$ total number of tasks considering all the jobs to be scheduled. We perform $T_{\max}$ iterations, and we evaluate B possible solutions; therefore, the complexity of SCHED-T is $\mathcal{O}(HBT_{\max})$. In Section 5, we show the running time for different instances.

4.4. Discussion

When considering dynamic job arrivals, there are different approaches for integrating the new job into the existing scheduling. For instance, one can exploit the gaps in the current schedule and run the newly arrived job during such gaps, with little or no impact on the other jobs. When we take into account the transport system, it is not easy to understand if the gaps can be fully exploited. In fact, even a small change in the current scheduling, such as shifting a task to accommodate the new one, results in a cascading effect that disrupts the whole scheduling.

Such interdependence between the job execution and the transport system is the reason behind our randomized approach to the exploration of the solution space. Every time we evaluate a possible solution, we need to compute the schedule from scratch. The solutions in the literature that adapt the current schedule to accommodate the new arrivals can not be easily extended when we consider the transport system, and the only available option is to recompute the whole schedule.

The use of a heuristic allows us to trade accuracy with speed, i.e., we accept sub-optimal solutions (with an error of less than 5%) that are obtained in a few seconds;

in Section 5, we show the running time for different instance sizes. Compared to the job processing time (e.g., tens of minutes), the scheduling processing time is very small, and therefore, the approach based on total rescheduling is justified. In case of high job arrival rates, rather than recomputing the scheduling at every arrival, it is possible to collect some new jobs before running SCHED-T. As a rule of thumb, if the scheduling processing time is P_{proc} seconds, and the average arrival rate (estimated considering the last arrivals) is V_{arr} jobs/s, then the new scheduling can be computed once $\lceil \beta \cdot V_{\mathrm{arr}} \cdot P_{\mathrm{proc}} \rceil$ jobs have arrived, with the parameter $\beta \geq 1$ that controls the trade-off between the delay and processing load.

Alternatively, at every job arrival, it is possible to consider only a subset of jobs waiting to be processed (e.g., the last W jobs in line) as part of the set of jobs that are included in the new schedule computation so that the other subset can be used to feed the production line during the computation.

5. Experimental Results

In this section, we demonstrate the validity of our proposed heuristic SCHED-T. First, we compare our heuristic with state-of-the-art approaches for solving the FJSS problem, using public benchmarks without the transport system. Then, we apply our heuristic to both a real-world scenario and a simulated scenario to evaluate its performance when the transport system is present.

5.1. Experimental Methodology and Settings

In order to compare SCHED-T with other heuristics, we consider the set of publicly available instances described in [12,16]. These instances have been created using the instance generator available at [31]. The generated instances contain information regarding the jobs, assuming that transportation is part of the task-processing time. We use these instances to evaluate the performance of our proposed scheduling heuristic with respect to the state-of-the-art. In all the comparisons, we use a cost function based on the *makespan*, i.e., the latest completion time of the entire set of scheduled tasks. The built-in parameters that guide our heuristics are (i) the number of iterations and (ii) the available budget at each iteration, divided between the budget dedicated to the exploration of local and remote neighbors. We analyzed the impact of these parameters on the makespan with a sensitivity analysis.

As for the transport system, unfortunately, public instances that include this aspect, such as the one used in [32], consider different means of transport (AGVs rather than conveyor belts). Therefore, they cannot be used in our comparison. Therefore, we consider a real-world use case related to our ICE lab, in which we were able to create a set of instances that include a transport system.

Finally, we consider the case of a sequence of arrivals. We assume, as is carried out in the literature [14], that a set of jobs is present at time zero and that another set randomly arrives. We evaluate the makespan in such a dynamic case (updated at every arrival), and we compare it with the makespan computed by an ideal clairvoyant scheduler, i.e., a scheduler that knows all the future arriving jobs and provides a single, optimized schedule.

SCHED-T is implemented in Python, and the experiments are carried out on a 3.3 GHz Intel Core i7 (Intel, Santa Clara, CA, USA) with 16 Gb of RAM (Microchip Technology Inc., Chandler, AZ, USA).

5.2. Instances with No Transport System

We compared SCHED-T with a set of heuristics that were presented in [12], in which authors applied iterated local search, genetic algorithms, differential evolution, and tabu search on a set of 50 *large* instances defined in the same paper, along with another set of 50 *large* instances proposed by the same authors in [16]. Each instance differs in the number of available machines (up to 97) on which tasks can be allocated, the number of jobs (up to 200), and the number of total tasks (up to 2000). None of these heuristics outperformed the others in all the instances: for some instances, differential evolution was better than the others, and for other heuristics, tabu search provides the best makespan. Rather than listing the values of each heuristic, we consider, for each instance, the best and the worst makespan obtained by these heuristics: we used such values to represent an interval for which the gap is much more narrow than the one found with the CP approach. Within this range lie the results of the four heuristics cited before (iterated local search, genetic algorithm, differential evolution, and tabu search). Therefore, we consider such a gap as a reference with which to compare. Instead of showing the absolute values for each instance, we can normalize the maximum makespan with respect to the minimum makespan so that the gaps of the different instances are comparable and can be put in a single graph. We also normalize the makespan obtained by SCHED-T so that it is simple to understand if it falls in the gap provided by other heuristics.

Makespan: In Figure 4, we compare the makespan obtained on the set of instances by SCHED-T with the one obtained with the other reference heuristics. The results show that SCHED-T is able to find a makespan within the gap defined by the other state-of-the-art heuristics. In some cases, it outperforms the other heuristics by 2%, whereas in two cases, it finds a larger makespan by 3%. We notice that every heuristic proposed in the literature, including our solution, provides an approximate solution, and there are no guarantees on the error bound. Therefore, we can only observe how well these heuristics perform on real datasets. Figure 4 shows that the approximate solutions have a 4–12% gap; therefore, such an error can be considered acceptable. From these results, SCHED-T performs similarly to other state-of-the-art heuristics on instances without considering the transport time.

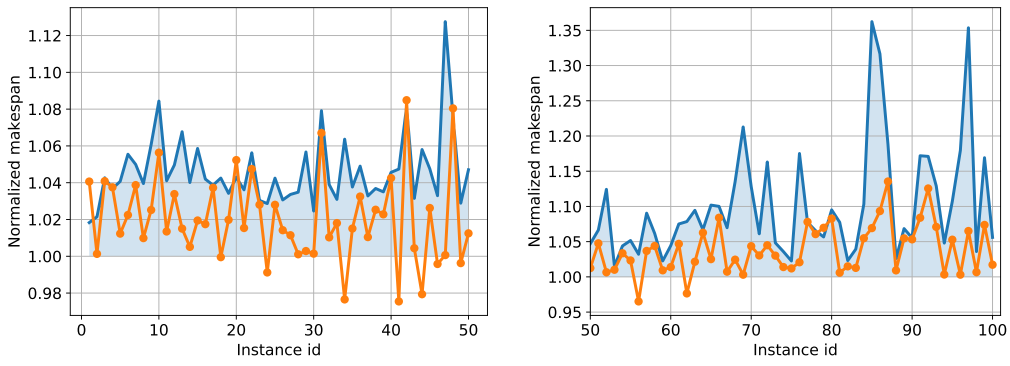

Figure 4. Normalized makespan for the set of the 50 + 50 large instances in [12,16]. The shaded area represents the gap (minimum and maximum makespan) found with different heuristics in [12], while the orange points are the results of SCHED-T.

Running time: Figure 5 shows the running time (in minutes) required to find the solutions of the 50 instances in [16]. The instances are ordered by increasing the number of tasks (shown at the top of the graph), e.g., instance 50 has almost 1000 tasks. Even with a large number of tasks, SCHED-T is able to obtain a solution that is comparable to the results found in the literature in less than 10 min.

Sensitivity analysis: SCHED-T has a set of parameters that tune the depth of the exploration phase. At the task level, we can exchange tasks belonging to the same job or to the next or previous job by considering the current scheduling order. At the job level, we can exchange jobs. During our tests, we noted that switching jobs provides most of the gain, while at the task level, the impact of changing the order on the makespan is less significant.

Therefore, the two main parameters we consider are the number of moves we explore $T_{\max}$ and the number of neighbors we evaluate in each iteration, R. These parameters influence the time required to compute the solution. Figure 6 shows two views for analyzing the impact of these parameters on the normalized makespan, i.e., the makespan found with a combination of $T_{\max}$ and R divided by the makespan found with the highest values of $T_{\max}$ and R used in our test. If we maintain $T_{\max}$ as fixed and we increase R (Figure 6, left), when

R is sufficiently large ($R > 10$), no additional improvement is observed. However, even for smaller values of R, the makespan only slightly increases (2–5%) with respect to the best makespan found. This is also confirmed if we keep R fixed and vary $T_{\max}$ (Figure 6, right); the two figures refer to two different instances, and they are representative of the general behavior that we observed for all the instances.

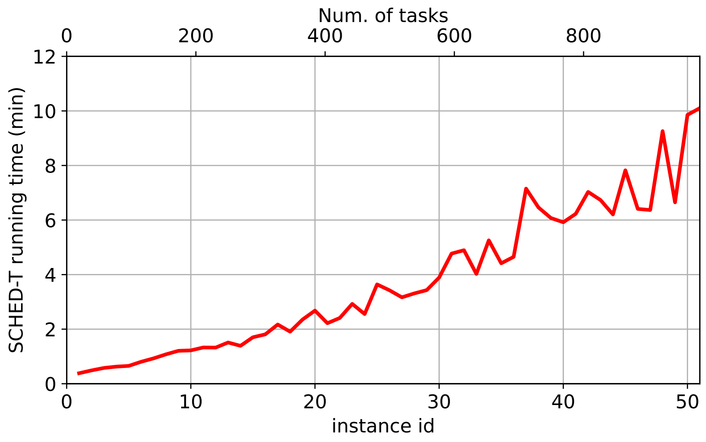

Figure 5. SCHED-T running time for finding the solution is shown in Figure 4. The top axis indicates the number of tasks associated with each instance.

Therefore, the two main parameters we consider are the number of moves we explore, denoted as $T_{\max}$, and the number of neighbors we evaluate in each iteration, denoted as R. These parameters influence the time required to compute the solution. Figure 6 provides two views for analyzing the impact of these parameters on the normalized makespan. In other words, it shows the makespan found with a combination of $T_{\max}$ and R divided by the makespan found with the highest values of $T_{\max}$ and R used in our test. If we keep $T_{\max}$ fixed and increase R (Figure 6, left), when R is sufficiently large ($R > 10$), no additional improvement is observed. Even for smaller values of R, the makespan only slightly increases (2–5%) compared to the best makespan found. This observation holds even if we keep R fixed and vary $T_{\max}$ (Figure 6, right). The two figures represent different instances but are indicative of the general behavior observed for all instances.

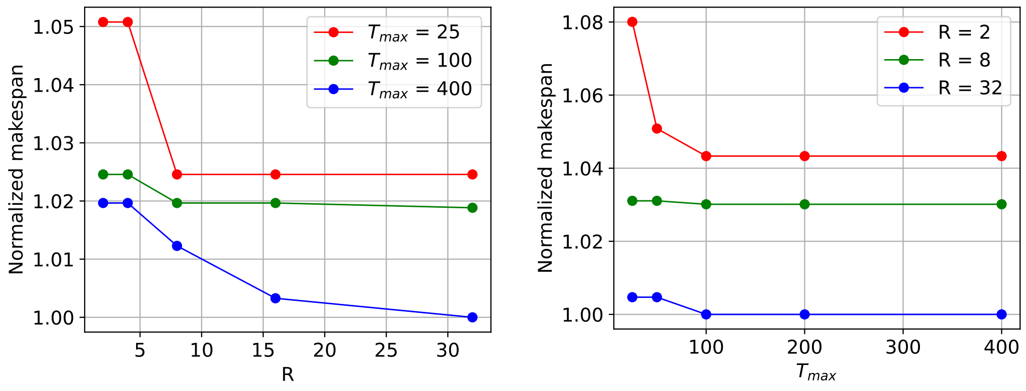

Figure 6. Sensitivity analysis of SCHED-T: impact of the parameters on the makespan (makespan normalized to the best value found). **Left** and **right** figures refer to different instances.

Overall, SCHED-T is not extremely sensitive to parameter settings and provides good results across a wide range of values. In terms of computational time, using small $T_{\max}$ and R allows for obtaining a solution in less than 30 s, while larger $T_{\max}$ and R may take up to 10 min. However, since large $T_{\max}$ and R do not significantly improve the results, the use of a small $T_{\max}$ and R is appropriate, especially in dynamic contexts where scheduling is recomputed when new jobs arrive.

In any case, the tuning process can be performed in a simulated environment where data collected from the production line is replayed, and different values of $T_{\max}$ and R are tested. If the production load does not vary much from one day to the next, this offline analysis should provide an indication of the best parameter settings for that specific production line.

ICE instances: In order to test SCHED-T in a real test case, we generated six instances that can be executed within our research laboratory (described in Section 2). The number of tasks and jobs for each instance are summarized in Table 2. Without considering the transport time, we were able to solve the problem using a MILP model and compare the results with SCHED-T (Table 2, column **Opt**).

Table 2. ICE instances: characteristics (cols. 2 and 3); makespan with no transport (cols. 4, 5, and 6).

Id	#Jobs	# Tasks	Opt	SCHED-T	Err.
1	5	32	900	900	0%
2	10	106	3580	3650	2.0%
3	15	100	2930	3014	2.9%
4	20	142	3530	3698	4.8%
5	25	174	4935	4998	1.3%
6	30	263	7475	7570	1.3%

By using SCHED-T, we are able to find a solution in less than 30 s. The difference between the local optimal and the global optimal solution is less than 5% (columns **SCHED-T** and **Err.** of Table 2). As mentioned earlier, the literature deems an error gap of up to 10% acceptable, acknowledging the trade-off between accuracy and computational speed provided by the approximate solutions. Regarding the variability in error, we were unable to pinpoint any specific job characteristic that might influence such variability. It can be regarded as an inherent variability intrinsic to the considered problem.

5.3. Instances with the Transport System

Introducing a transport system increases the complexity of the MILP model exponentially. In fact, a MILP formulation given to standard MILP solvers, such as IBM CPLEX, cannot be solved after hours of computation. As such, we do not have any values that can be used as a global optimal comparison reference to our results. The results presented in Table 3, column **SCHED-T** of **Experim.-I**, report the makespan found by SCHED-T. Next, we describe the test we performed on a real-world testbed to validate such results.

Table 3. Reference layout scheduling instances with transport.

Id	Experim.-I			Experim.-II		
	SCHED-T	**Actual**	**Err.**	**SCHED-T**	**Actual**	**Err.**
1	1511 s	1556 s	−2.89%	1221 s	1555 s	−21.48%
2	2814 s	2861 s	−1.64%	2243 s	2875 s	−21.98%
3	3557 s	3674 s	−3.18%	3321 s	4248 s	−21.82%
4	4654 s	4892 s	−4.87%	4366 s	5577 s	−21.71%
5	5820 s	5963 s	−2.40%	5473 s	7012 s	−21.95%
6	7025 s	7227 s	−2.80%	6461 s	8377 s	−22.87%

5.4. Real-World Experiments

We consider the set of instances described in Table 2, and we run them on an actual production line built in our lab (described in Section 2). The production line is governed by service-oriented manufacturing (SOM) software architecture similar to the one presented in [33,34], which automatically manages the production line, interacting with both the manufacturing execution system (MES) and the machines. On top, we have developed a module that implements our proposed scheduling heuristic, and the execution is forwarded to the SOM architecture. For each instance's job, we have fixed the task execution time. This allows for a fair evaluation of the accuracy of SCHED-T, removing processing time variability. The case of a stochastic execution time will be considered in future works.

Given the schedule returned by SCHED-T, we ran the jobs on the production line and recorded the makespan. The results are shown in Table 3, columns **Experim.-I**, and **Actual**. SCHED-T is able to predict the actual makespan obtained from a real-world production line with an error smaller than 5%. Additionally, in this case, there is no specific job characteristic that might influence the small variability for the different instances.

Then, in column **Experim.-II**, we consider what happens if the scheduler does not take into account the transport system, as most of the schedulers in the literature do. In this case, they would produce a schedule that is not optimal. In particular, we run SCHED-T by setting all the transport times to zero and we obtain a schedule that is fed to our production line. We record the makespan obtained in this case and compare the error obtained with the one computed in columns **Experim.-I**. If the scheduler does not include the transport times, the resulting schedule contains an error of up to 23%, which is 18% greater than the one obtained by the schedule considering the transport times. Thus, the makespan obtained using the real system by considering the transport time improves the makespan by up to 14% with respect to the one found without considering the transport time.

Table 4 shows the results obtained by applying SCHED-T to a different plant configuration depicted in Figure 7. This new configuration consists of a modified version of our real-world case study: we added a bay and a machine of the same type on the opposite side of each existing one to implement production redundancy. We built this new configuration by exploiting Tecnomatix Plant Simulation, a commercial state-of-the-practice discrete event simulation tool. The simulation allows for testing our proposed algorithm on a different scenario, enabling the calculation of simulated production times (e.g., makespan) and demonstrating that SCHED-T can also be applied to different plant configurations. We executed the same test using Table 3, generating new instances of the same size. The results show that when considering transport time, SCHED-T allows for reducing the error of the estimated makespan, keeping it within 5% and lowering it by 17% when not considering transport time.

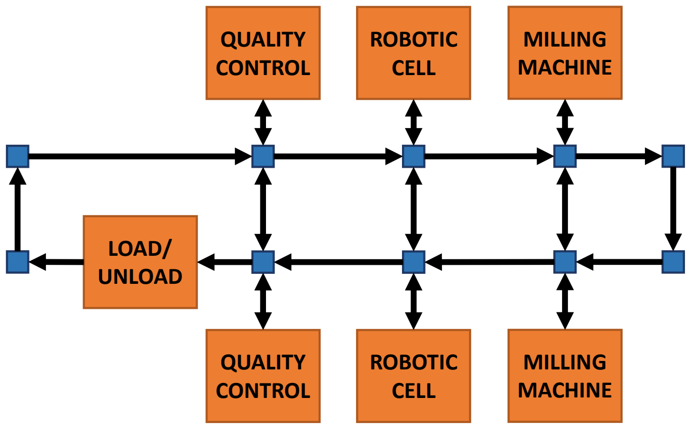

Figure 7. The modified layout of the production line used to test another scenario. We added a bay and a machine of the same type on the opposite side of each existing one.

Table 4. Modified layout scheduling instances with transport.

Id	Experim.-I			Experim.-II		
	SCHED-T	Actual	Err.	SCHED-T	Actual	Err.
1	1362 s	1343 s	+1.41%	1107 s	1412 s	−21.60%
2	2444 s	2436 s	+0.33%	2214 s	2721 s	−18.63%
3	3575 s	3685 s	−2.99%	3231 s	4239 s	−23.78%
4	4796 s	4775 s	+0.44%	4428 s	5557 s	−20.32%
5	6041 s	5929 s	+1.89%	5354 s	6945 s	−22.91%
6	6973 s	7189 s	−3.00%	6488 s	8428 s	−23.02%

To sum up, our proposed solution allows us to estimate the makespan within a reasonable time frame with high precision. By correctly estimating the transfer times time, we are able to build a more precise schedule.

5.5. Multiple Arrivals

The results discussed above focused on the efficiency of the solution at each single job arrival. In this section, we evaluate how the schedule changes as new jobs arrive. We consider the same set of instances without the transport system (Section 5.2) and with the transport system (Section 5.3). We remove p% of the jobs, compute the initial makespan without these jobs, and uniformly spread the arrival of the removed jobs at random between zero and the initial makespan. For each arrival, we update the schedule and the makespan.

In order to evaluate the quality of the final makespan obtained in this way, we consider an *ideal* scheduler that knows all the jobs (the initial ones and the future ones, with their arrival times) in advance and computes a single optimize schedule; we call this scheduler *clairvoyant*.

Figure 8 shows the difference (in percentage) between the final makespan obtained after updating the initial makespan at every job arrival and the makespan obtained with the clairvoyant scheduler for different percentages of dynamic jobs (jobs removed from the instances).

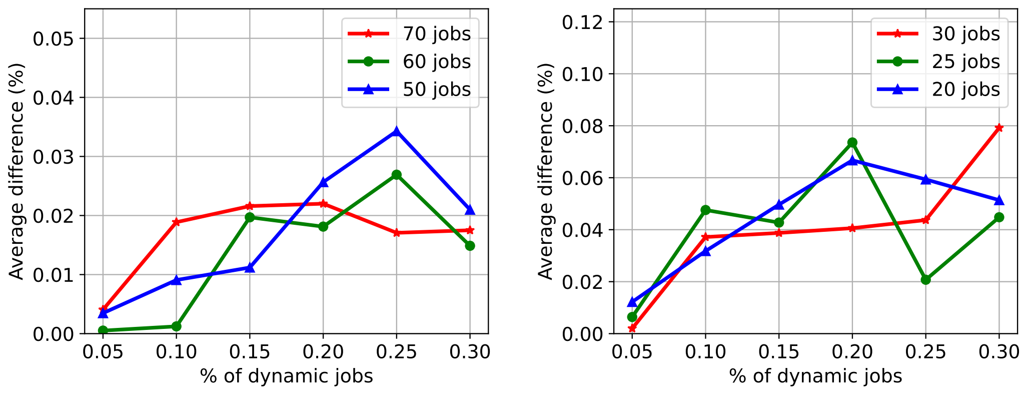

Figure 8. Difference (in percentage) between the makespan updated at every job arrival and the makespan from the ideal clairvoyant scheduler. Left and right figures refer to instances in [16] and the ICE instances.

In both cases (with and without the transport system), updating the schedule at every arrival provides a solution that is comparable to an ideal scheduler that has all the information in advance. The case with the transport system shows a higher deviation since the instances have a lower number of jobs. Overall, the total rescheduling strategy provides near-optimal solutions with a limited use of the processing resources.

6. Conclusions

Estimating the start and finish time of tasks belonging to jobs is fundamental for improving the efficiency of the manufacturing system. This is even more important in a fully automated environment, in which tasks are typically frequent but short. Integrating task duration with transport times enables more robust control of the overall process. To this aim, we designed a scheduler based on stochastic local search, which explores the solution space with a randomized approach.

Our proposed heuristic, SCHED-T, allows for building accurate and near-optimal schedules in a limited amount of time. This allows for efficiently managing the dynamic scenarios in which jobs continuously arrive or when unexpected events, such as machine breakdowns or maintenance activities, may occur.

In the future, we plan to extend our heuristic to include stochastically variable task-processing times and evaluate their impact on overall efficiency.

Author Contributions: Conceptualization, S.G. and D.C.; investigation, S.G. and D.C.; methodology, D.C.; software, S.G. and D.C.; writing—original draft preparation, S.G. and D.C.; writing—review and editing, S.S.; supervision, S.S. and F.F.; project administration, F.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially funded by the Italian Ministry for University and Research (MUR) under the project *"Joint Communication and Sensing: CSI-Based Sensing for Future Wireless Networks (CSI-Future)"*, PRIN 2022-NRRP Mission 4, CUP D53D23016040001. The research was partially supported by the Italian National Group for Scientific Computation (GNCS-INDAM). Thisstudy was carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5–D.D. 1058 23/06/2022, ECS_00000043). This manuscript reflects only the Authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflict of interest.

Terminology

Availability	Intervals of times when machines can be used (i.e., it is not under maintenance or booked for task execution).
Job	A set of operations or tasks that create a new product either directly from raw materials or components.
Machine	Equipment that processes, forms, or shapes raw materials or the output or other machines.
Makespan	Total execution time to process all the jobs currently submitted to the facility.
Precedence	Constraint on a task that can not be started before the completion of one or more tasks.
Processing time	Time required by a machine to perform a task.
Task	A unit of work performed on a single machine.
Transport times	Time required to move the partially processed materials from one machine to the next.

References

1. Wen, X.; Qian, Y.; Lian, X.; Zhang, Y.; Wang, H.; Li, H. Improved genetic algorithm based on multi-layer encoding approach for integrated process planning and scheduling problem. *Robot. Comput. Integr. Manuf.* **2023**, *84*, 102593. [CrossRef]
2. Grau, A.; Indri, M.; Bello, L.L.; Sauter, T. Industrial robotics in factory automation: From the early stage to the Internet of Things. In Proceedings of the IECON 2017—43rd Annual Conference of the IEEE Industrial Electronics Society, Beijing, China, 29 October–1 November 2017; pp. 6159–6164.
3. Luo, Q.; Deng, Q.; Xie, G.; Gong, G. A Pareto-based two-stage evolutionary algorithm for flexible job shop scheduling problem with worker cooperation flexibility. *Robot. Comput. Integr. Manuf.* **2023**, *82*, 102534. [CrossRef]
4. Chaudhry, I.A.; Khan, A.A. A research survey: Review of flexible job shop scheduling techniques. *Int. Trans. Oper. Res.* **2016**, *23*, 551–591. [CrossRef]
5. Zhang, J.; Ding, G.; Zou, Y.; Qin, S.; Fu, J. Review of job shop scheduling research and its new perspectives under Industry 4.0. *J. Intell. Manuf.* **2019**, *30*, 1809–1830. [CrossRef]
6. Pinedo, M.L. *Scheduling*; Springer: Berlin/Heidelberg, Germany, 2012; Volume 29.
7. Kourtis, G.; Kavakli, E.; Sakellariou, R. A rule-based approach founded on description logics for Industry 4.0 smart factories. *IEEE Trans. Ind. Informatics* **2019**, *15*, 4888–4899. [CrossRef]
8. Zhang, G.; Sun, J.; Liu, X.; Wang, G.; Yang, Y. Solving flexible job shop scheduling problems with transportation time based on improved genetic algorithm. *Math. Biosci. Eng.* **2019**, *16*, 1334–1347. [CrossRef]
9. Yao, Y.J.; Liu, Q.H.; Li, X.Y.; Gao, L. A novel MILP model for job shop scheduling problem with mobile robots. *Robot. Comput. Integr. Manuf.* **2023**, *81*, 102506. [CrossRef]
10. Li, Y.; Gu, W.; Yuan, M.; Tang, Y. Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep Q network. *Robot. Comput. Integr. Manuf.* **2022**, *74*, 102283. [CrossRef]
11. Pavlov, A.A.; Misura, E.B.; Melnikov, O.V.; Mukha, I.P. NP-Hard Scheduling Problems in Planning Process Automation in Discrete Systems of Certain Classes. In *Advances in Computer Science for Engineering and Education*; Hu, Z., Petoukhov, S., Dychka, I., He, M., Eds.; Springer: Cham, Switzerland, 2019; pp. 429–436.
12. Lunardi, W.T.; Birgin, E.G.; Ronconi, D.P.; Voos, H. Metaheuristics for the online printing shop scheduling problem. *Eur. J. Oper. Res.* **2021**, *293*, 419–441. [CrossRef]
13. Hoos, H.H.; Stützle, T. *Stochastic Local Search: Foundations and Applications*; Elsevier: Amsterdam, The Netherlands, 2004.
14. Wang, Z.; Zhang, J.; Yang, S. An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals. *Swarm Evol. Comput.* **2019**, *51*, 100594. [CrossRef]
15. Baykasoğlu, A.; Madenoğlu, F.S.; Hamzadayı, A. Greedy randomized adaptive search for dynamic flexible job-shop scheduling. *J. Manuf. Syst.* **2020**, *56*, 425–451. [CrossRef]
16. Lunardi, W.T.; Birgin, E.G.; Laborie, P.; Ronconi, D.P.; Voos, H. Mixed Integer linear programming and constraint programming models for the online printing shop scheduling problem. *Comput. Oper. Res.* **2020**, *123*, 105020. [CrossRef]
17. Gaiardelli, S.; Carra, D.; Spellini, S.; Fummi, F. On the Impact of Transport Times in Flexible Job Shop Scheduling Problems. In Proceedings of the 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation, Stuttgart, Germany, 6–9 September 2022; pp. 1–8.
18. Industrial Computer Engineering (ICE) Lab. Available online: https://www.icelab.di.univr.it/ (accessed on 1 June 2023).
19. Li, X.; Guo, X.; Tang, H.; Wu, R.; Wang, L.; Pang, S.; Liu, Z.; Xu, W.; Li, X. Survey of integrated flexible job shop scheduling problems. *Comput. Ind. Eng.* **2022**, *174*, 108786. [CrossRef]
20. Naderi, B.; Zandieh, M.; Balagh, A.K.G.; Roshanaei, V. An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. *Expert Syst. Appl.* **2009**, *36*, 9625–9633. [CrossRef]
21. Sun, Y.; Chung, S.H.; Wen, X.; Ma, H.L. Novel robotic job-shop scheduling models with deadlock and robot movement considerations. *Transp. Res. Part E Logist. Transp. Rev.* **2021**, *149*, 102273. [CrossRef]
22. Zhang, X.-j.; Sang, H.-y.; Li, J.-q; Han, Y.-y; Duan, P. An effective multi-AGVs dispatching method applied to matrix manufacturing workshop. *Comput. Ind. Eng.* **2022**, *163*, 107791. [CrossRef]
23. Lu, J.; Ren, C.; Shao, Y.; Zhu, J.; Lu, X. An automated guided vehicle conflict-free scheduling approach considering assignment rules in a robotic mobile fulfillment system. *Comput. Ind. Eng.* **2023**, *176*, 108932. [CrossRef]
24. Li, X.; Xing, K. Iterative Widen Heuristic Beam Search Algorithm for Scheduling Problem of Flexible Assembly Systems. *IEEE Trans. Ind. Inform.* **2021**, *17*, 7348–7358. [CrossRef]
25. Kleywegt, A.J.; Shapiro, A.; Homem-de Mello, T. The sample average approximation method for stochastic discrete optimization. *SIAM J. Optim.* **2002**, *12*, 479–502. [CrossRef]
26. Mitzenmacher, M. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.* **2001**, *12*, 1094–1104. [CrossRef]
27. Yang, J.; Wang, Y.; Wang, Z. Efficient Modeling of Random Sampling-Based LRU. In Proceedings of the 50th International Conference on Parallel Processing, Lemont, IL, USA, 9–12 August 2021; pp. 1–11.
28. Furnas, G.W. Generalized fisheye views. *ACM Sigchi Bull.* **1986**, *17*, 16–23. [CrossRef]

29. Pei, G.; Gerla, M.; Chen, T.W. Fisheye state routing: A routing scheme for ad hoc wireless networks. In Proceedings of the 2000 IEEE International Conference on Communications. ICC 2000. Global Convergence Through Communications, Conference Record, New Orleans, LA, USA, 18–22 June 2000; Volume 1, pp. 70–74.
30. Chen, J.; Chen, K.; Wu, J.; Chen, C. A study of the flexible job shop scheduling problem with parallel machines and reentrant process. *Int. J. Adv. Manuf. Technol.* **2008**, *39*, 344–354. [CrossRef]
31. FJS Instance Generator. Available online: https://github.com/willtl/online-printing-shop (accessed on 1 June 2023).
32. Zeng, C.; Tang, J.; Yan, C. Scheduling of no buffer job shop cells with blocking constraints and automated guided vehicles. *Appl. Soft Comput.* **2014**, *24*, 1033–1046. [CrossRef]
33. Gaiardelli, S.; Spellini, S.; Panato, M.; Lora, M.; Fummi, F. A Software Architecture to Control Service-Oriented Manufacturing Systems. In Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 14–23 March 2022; pp. 1–4.
34. Beregi, R.; Pedone, G.; Háy, B.; Váncza, J. Manufacturing Execution System Integration through the Standardization of a Common Service Model for Cyber-Physical Production Systems. *Appl. Sci.* **2021**, *11*, 7581. [CrossRef]