On the Impact of Transport Times in Flexible Job Shop Scheduling Problems

Sebastiano Gaiardelli, Damiano Carra, Stefano Spellini, Franco Fummi
University of Verona, Italy
{firstname.lastname}@univr.it

***Abstract*—Manufacturing systems require a careful scheduling of the resource usage to maximize the production efficiency. In a completely automated environment, the transport system should be orchestrated to work smoothly with the other resources. While the impact of job characteristics, such as fixed or variable processing times of the tasks composing the jobs, or task dependencies, has been extensively studied, the role of the transport system has received less attention.**

In this paper we consider a conveyor belt as a mean of transportation among a set of production machines. In this scenario, there is no input or output buffer at the machines, and the transport times depend on the availability of the machines. We propose a heuristic based on randomization, called SCHED-T, which is able to find a near optimal joint schedule for job processing and transfer in few seconds. We test our solution on known benchmarks, along with real-world instances, showing that our scheduler is able to predict accurately the overall processing time of a production line.

I. INTRODUCTION

The continuous evolution of manufacturing systems aims at reaching the complete automation of the production process [1]. This allows for an effective scheduling of all the operations involved in the production, which in turn improves the production efficiency. For this reason, the scheduling problem has been studied extensively in the last decades [2], [3]. In its simplest form, such a problem is usually referred to as the job shop scheduling (JSS) problem [4]. Given a set of *machines* and a set of *jobs*, considering that each job is composed by a set of *tasks* that must be processed in a given order by different machines, the problem consists in finding an assignment for the tasks that minimizes an objective function, such as the total completion time or *makespan*.

The introduction of different features in the manufacturing technology, such as the ability for a machine to perform different types of tasks, led to various extensions to the JSS problem – the flexible JSS (FJSS) problem being an example. A key element that has only been taken into consideration in recent years is the automation of the *transfer* of jobs from one machine to the next. While the execution of the tasks on machines can be characterized by their processing time, the transfer has a variety of settings that include (i) the means of transportation, such as automatic guided vehicle (AGV) or a conveyor belt, and (ii) the input and output buffers at the machines, *e.g.*, their ability to store the jobs to be processed (e.g., while finishing another task) and to store the processed job (while waiting for the transport system to be available)[1].

Some specific combinations of transport system and buffer may be modeled with a constant time [5]. This is the case in which the transport facility is always available and the machines have sufficiently large input and output buffers. In such a scenario, it is easy to include the transfer time into the processing time and use the numerous solutions proposed in the literature [2]. In other cases, the transfer time depends on the availability of the transport system and the destination machine [6]. Overall, no single model could cover all the alternatives, so specific combinations require ad-hoc modeling.

In this work, we consider a setting inspired by a completely automated production line that includes a conveyor belt as a mean of transportation, and no buffer at the machines. In this scenario, once a task is finished, the job will be immediately put on the belt to free the machine. If the next machine is busy, the job will remain on the belt until the destination machine is ready. It is well known that the JSS problem is NP-hard, and the variant considered here is at least as difficult. Therefore, we propose a heuristic, called SCHED-T, which is able to provide an approximate solution in reasonable time. SCHED-T represents an instance of the *Stochastic Local Search* (SLS) approach [7] – SLS includes well known algorithms such as Simulated Annealing, Evolutionary Algorithm and Tabu Search.

The main issue we need to face is the evaluation of the potential move when exploring the solution space. A small change in the scheduling sequence, in fact, has a cascading effect on the remaining tasks, since the sequence depends on the transfer times, which in turn depends on the execution sequence. While the solutions proposed in the literature are based on approximate evaluation of each move (*e.g.*, computation of the critical path), we take a randomize approach, in which we assess accurately few random neighbors.

We evaluate SCHED-T on a publicly available set of instances that do no include transfer times [8]. In such a case, our heuristic is able to obtain the same results as other previously proposed heuristics. We then analyze a set of instances generated from a production line for which the transfer times are available. In this case, when the instances are small, one may find the optimal solution using standard approaches based on Mixed Linear Integer programming (MILP). We show that

[1]Our definition of *transfer* refers to the time between the end of task and the start of the next one for a given job.

these approaches are not able to find a solution in reasonable time, so SCHED-T is the only viable approach. We also show that, when the scheduling is actually executed on a the production line, we achieve the makespan predicted by SCHED-T.

The remainder of the paper is organized as follows. In Section II we provide some background on the scenario we consider and discuss the related work. In Section III we formulate the problem, and we discuss our solution in Section IV. We evaluate SCHED-T in Section V and conclude the paper in Section VI.

II. BACKGROUND AND RELATED WORK

Production line with a conveyor belt. We consider a manufacturing system with a general *layout*, in which different machines dedicated to specific tasks are placed close one to another. The machines are connected to a conveyor belt that is used to circulate the job. In addition, a storage facility contains both the raw material and the finished products. This layout has been adopted in the production line available at our research facility [9]. The structure, depicted in Figure 1, is organized in a set of "production cells", each dedicated to a specific manufacturing process. In particular, from left to right, the laboratory includes a vertical warehouse, a quality control cell, a robotic assembly station and a multi-tool machining station. The transportation system is based on RFID readings and it is implemented by multiple conveyors, on which a maximum of *nine* pallets are transported. In particular, the set of conveyors can be differentiated in:

- *four unloading belts*, transporting the pallets (*i.e.*, the materials) near the machines for processing. Each production cell has an unloading belt and, therefore, a single input/output access route. Furthermore, one single pallet can be present at a time in such conveyors;
- *the main belt*, transporting the pallets towards the different production cells and, therefore, to the unloading belts. The main conveyor is composed of two long belts running in opposite directions. Furthermore, at the end of each long belt, a switching mechanism is in place for the pallets to traverse from one belt to the other. The main conveyor acts an active buffer for pallets in a waiting state, by continuously circulating them.

An important difference between the main conveyor and the unloading belt is that the former cannot be stopped while the latter can be stopped and activated when a pallet must be released or admitted. Therefore, the main conveyor has particular mechanical components to handle the routing of pallets at junction points. Here, the possible directions of the pallets are: (1) direct to the unloading bay, (2) continue straight, along the actual belt and (3) switch to the opposite long conveyor. The lead is always given to pallets exiting from unloading bays.

Literature review. The JSS problem, along with the different variants (such as FJSS) that reflect additional settings, has been extensively studied in the last decades. Since our work considers the impact of the transfer times on the scheduling, we focus our attention to the related work on this specific topic. For a general overview of the solutions to the JSS problem, the interested reader may refer to the surveys presented in [2] and [3].

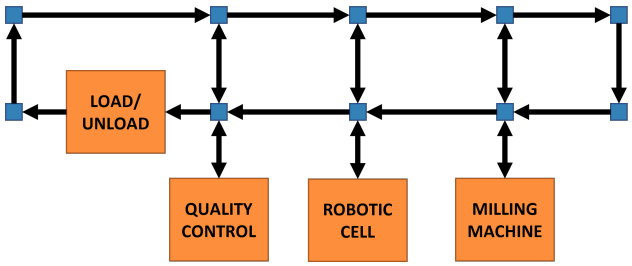

Fig. 1. The layout of the production line used as a case-study.

Transfer times depend on the task execution sequence, and there are numerous works, *e.g.* [10], that deal with the JSS problem with sequence-dependent setup times. Nevertheless, during the setup, the machine is not available. In our case, instead, the machine can be used while the job is travelling, so we can not cast our problem as JSS with sequence-dependent setup times.

Many works consider as a mean of transportation a set of automated guided vehicles (AGVs) [5], [6], [11], [12], and solve jointly the problem of job scheduling and transport scheduling. The use of AGVs implies that, at the end of a task, the machine needs to wait for an available AGV to pick up the job. Subsequently, when the AGV arrives at the destination machine, either the machine has a buffer, or the AGV needs to wait before unloading the job, so the AGV is not available for other pickups. In our case, instead, when a task is completed, the machine immediately unloads the job on the conveyor belt. In addition, if a job is waiting for a machine to be available, its presence on the conveyor belt is not limiting other machines to unload other jobs on the belt.

The heuristics proposed in the literature for the different variants of the JSS problem need to solve a common issue: the estimation of the objective function during the local exploration of the solution space – see for instance [10], [13]. The proposed solutions are based on a graph-based representation of the execution sequence: through the evaluation of the graph properties, such as the critical path, it is possible to understand if a different solution is feasible, along with an approximate estimation of the objective function. In our case, the inter-dependency between the transfer time and the execution sequence makes the graph-based approaches infeasible, and we adopt a randomized local exploration.

By formalizing the problem as a minimization problem with linear constraints, some works, such as [8], aim at finding the optimal solution using MILP solvers. While the details of our transportation system can be indeed be expressed with these constraints, even the solution of very small instances could not be found in a reasonable time.

III. Problem description

The Flexible Job Shop Scheduling (FJSS) problem consists in scheduling a set of n jobs, $J_1, J_2, \ldots, J_n$, on m machines. Each job J_i is composed by h_i operations or *tasks*, and each task can be executed on a subset of machines. For a given task of a job and a given machine, the processing time is considered known.

Such a baseline definition is augmented with a set of assumptions that are related to the specific environment we consider. In particular, we assume that:

- All jobs are ready at time zero;
- A machine performs at most one task of any job at a time;
- Tasks may have precedence constraints, *i.e.*, a task can not start until its preceding tasks (if any) are completed;
- Machines have no input or output buffers.

The latter assumption comes from the layout of the production line we consider, in which a conveyor belt is responsible for moving the job parts from one machine to the next. Since the conveyor belt is always available, there is no need of an output buffer, and the parts can be released as soon as they are processed by a machine. In this completely automated environment, input buffers at each machine are difficult to implement: instead, the parts keep circulating on the conveyor belt until the destination machine is ready.

While the above assumptions are specific to the context we study in this paper, the proposed solution is general and can be easily adapted in case of some of these assumptions change.

Table I summarizes the notation used for the problem formulation.

TABLE I
NOTATION SUMMARY

Inputs	
M_k	Machine k, $k \in [1, m]$
J_i	Job i, $i \in [1, n]$
h_i	Number of tasks of job i
t_{ij}	jth task of J_i, $j \in [1, h_i]$
p_{ijk}	processing time for t_{ij} on machine k
$U_{ijj'}$	1 if t_{ij} precedes $t_{ij'}$, 0 otherwise
σ_{kl}	transfer time between machines k and l, $k, l \in [1, m]$
Auxiliary variables	
E_{ijk}	1 if t_{ij} is executed on machine k, 0 otherwise
s_{ijk}	execution start time of t_{ij} on machine k
C_{ij}	completion time of t_{ij}

The tasks that compose a job i are identified with t_{ij}. Each task can be performed by different machines. We identify the processing time of task t_{ij} on machine k with p_{ijk}. If a task t_{ij} can not be done on machine k, we set $p_{ijk} = \infty$. The precedence constraints are summarize by a square matrix U whose rows and columns identify the set of all the tasks (of all jobs), and each element is either 0 or 1 depending if one task precedes the other. The transfer time σ_{kl} is the information that characterizes our work. In particular, it is defined as follows:

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

where σ^0_{kl} is the minimum time to go from machine k to machine l, σ^c_{kl} is the cycle time in case the machine l is not available, and n^c is the number of cycles that the job needs to do until the machine becomes available. Equation 1 can only be computed *while building the scheduling*, since its value depends on the availability of the destination machine. This scheduling-dependent formulation makes it difficult to find a simple way of estimating the transfer time, which in turn has an impact on the estimation of the makespan. The only available choice is to build the entire schedule, and compute the makespan while doing so.

We are now ready to formulate our problem. The aim of the scheduling is to minimize the *makespan*, *i.e.*, the total execution time required to process all the tasks. Formally, we have:

$$\underset{i \in [1,n]}{\text{minimize}} \quad \max\ C_{ij} \tag{2}$$

$$\text{s.t.}$$

$$s_{ijk} + p_{ijk} = C_{ij} \tag{3}$$

$$s_{ijk} + p_{ijk} + \sigma_{kl} \leq s_{ij'l} \qquad \forall t_{ij}, t_{ij'} | U_{ijj'} = 1 \tag{4}$$

$$\sum_k E_{ijk} = 1 \qquad \forall t_{ij} \tag{5}$$

$$s_{ijk} + p_{ijk} < s_{i'j'k} \qquad \forall t_{ij}, t_{i'j'}, k \in [1, m] \tag{6}$$

Equation 3 defines the relation between the start and finish times for each task. Here we do not consider the transfer time, and we do not count the time required to bring the job back to the storage facility after the last operation in each job. Equation 4 represents the precedence constraints: once two tasks have been assigned to machine k and l, if one task precedes the other, then the next task can not start before the finish time of the previous one, including the transfer time. Equation 5 imposes that each task can be assigned to exactly one machine. Equation 6 represents the fact that each machine can process one task at a time.

While the above formulation can be used to model the problem and solve it with approaches based on MILP, the introduction of the transport system, with the scheduling-dependent computation of the transfer times, makes such a task computationally intensive. Even with a small number of jobs and tasks, standard MILP solvers are not able to complete the computation after 12 hours. For this reason, we need to resort to alternative approaches based on the exploration of the solution space.

IV. Exploring the solution space

A. Overview

The general scheme of any heuristic based on Stochastic Local Search is composed by the following steps [7]:

1) Build a solution and compute the objective function;
2) Explore the neighborhood, evaluating the objective function for each neighbor;
3) Select the neighbor according to a given criteria;
4) Repeat from step 2 until a stop condition is met.

The solution, therefore, needs to specify how the neighborhood exploration is done, along with the evaluation of the objective function, the criteria used to select next solution, and the stop condition. From the computational point of view, the complexity depends on the size of the neighborhood. Given a sequence s_i of tasks assigned to the set of machines, we may define as neighbor any sequence s_j that differs from s_i for few tasks. For instance, one may swap two or three tasks. This would translate into a neighborhood at least quadratic in the number of tasks.

To decrease the number of neighbors to assess, it is common to discard beforehand some combinations that, with high probability, will not improve the solution. For instance, if we consider the makespan as objective function, since the tasks belonging to the same job have strong dependencies, they should be scheduled almost sequentially. Swapping a task from the first job with a task from the last job would probably not improve the makespan.

In addition to the size of the neighborhood, we need to face another challenge: for each neighbor we should be able to compute efficiently the objective function. This is indeed very difficult in the specific problem setting we consider for two reasons. The tasks within a job have dependencies, so moving one task may have a cascading impact on the other tasks. In addition, the transfer time depends on the availability of the destination machine (see Eq.1), but the availability depends on the execution order. While the impact of the first issue can be mitigated with known techniques on graph-based representation of the problem, the circular dependency between the transfer time and execution order makes it difficult to estimate, even roughly, the objective function.

The only option is, then, to compute from scratch the objective function for each neighbor. Since the evaluation of a single neighbor is so expensive, then the neighbor set should be chosen accurately.

B. Randomized approach

Limiting the neighbor set to be explored is a technique already adopted by many solutions. Nevertheless, the number of neighbors still depends on the number of tasks. The main idea we adopt is the introduction of a randomized selection of neighbors with the aim of evaluating only a few neighbors, This approach is inspired by the results proposed in [14], [15], along with some application in other fields [16].

In particular, we consider a randomly selected set of neighbors, we compute the allocation for each sequence and we evaluate the corresponding objective function. We then select the best option and, if it improves over the current state, we change the current solution, otherwise we repeat the process with a different random subset.

In order to avoid to be stuck in a local minimum, we can adopt known techniques, such as Tabu search or Simulated Annealing, in which we accept a solution with a given probability even if it does not improve over the current solution. Another option is to adopt a sampling approach in testing solutions that are not part of the neighborhood. This solution, inspired by the fisheye view [17] and fisheye routing [18], considers neighbors with a probability that depends on their distance. In other words, close neighbor are sampled with higher rate, while distant neighbors are sampled with lower rate. In our context, a distant neighbor is obtained with more complex changes in the task execution order.

C. Detailed solution

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

Initialization. We start from a random sequence of jobs, $s_J^l = \{J_i^l\}$, where J_i^l indicates that job i is executed in the sequence l. We then assign the tasks of each job to the different machines, following the order of s_J^l, *i.e.*, we allocate the tasks of job J_i^l only after having allocated all the tasks of job J_{i-1}^l. For a given job, the task execution order is chosen considering the dependencies among tasks, and selecting randomly in case of tasks with the same priority. For instance, if a job has three tasks, and the first two tasks precede the last task, then the first two tasks has the same priority, and the initial solution will schedule randomly one the two, then the other one and finally the last task. For a given task, if there are more than one machine that can perform that task, we choose the machine in which the task terminates earlier (including the transportation time). The initial allocation is a sequence $s_T^l = \{t_{ij}^l\}$ of tasks, where t_{ij}^l indicates that task j of job i is executed in the sequence l.

Neighborhood. For a given sequence $s_T^l = \{t_{ij}^l\}$ of tasks assigned to the set of machines, we define a *close neighbor* the solution in which we work at low level, *i.e.*, we switch tasks to obtain a new sequence $s_T^{l'}$ to be evaluated. A random neighbor is obtained by (i) selecting a random task t_{ij}^l in s_T^l, and (ii) selecting another random task $t_{i'j'}^l$ that must belong to the same job ($i' = i$) or to a job that comes before or after job i in the sequence s_J^l.
We also define a *remote neighbor* the solution in which we work at higher level, *i.e.*, we switch jobs in the sequence s_J^l – the order of the tasks can be further explored, and we may change the allocation on the machines depending on their availability in the new sequence.

Exploration. In each iteration we explore close and remote neighbors, and we select the solution that improves over the current sequence – see Algorithm 1. The scheme has a number of parameters. The budget B represents the number of solutions (neighbors) to be evaluated in each iteration. This budget is split between the close neighbor (αB) and the remote neighbors ($(1-\alpha)B$), with $0 < \alpha < 1$. We consider R remote neighbors: once selected a remote neighbor, we explore locally its neighbors with a budget $(1-\alpha)B/R$. The exploration concludes is no improvements are observed for T_{idle} iterations (not shown in Algorithm 1), or if we reach the maximum number of iterations T_{max}

Algorithm 1: SCHED-T

input: $\{J_i\}$, jobs to be scheduled
input: $\{M_i\}$, machines
input: B, α, R, parameters for exploration

```
s_0 ← Rand_Init({J_i});
O ← Eval_solution(s_0);
i = 1;
while i ≤ T_max do
    s_i ← s_{i−1};
    foreach αB close neighbor do
        s'_i ← Rand_Local_Perm(s_{i−1});
        O' ← Eval_solution(s'_i);
        if O' < O then
            s_i ← s'_i;
            O = O';
    foreach R remote neighbor do
        s'_{i−1} ← Rand_Remote_Perm(s_{i−1});
        foreach (1 − α)B/R of its close neighbor do
            s'_i ← Rand_Local_Perm(s'_{i−1});
            O' ← Eval_solution(s'_i);
            if O' < O then
                s_i ← s'_i;
                O = O';
    i++;
```

V. EXPERIMENTAL RESULTS

We compare SCHED-T with the state-of-the-art approaches for solving the FJSS problem based on heuristics. We consider public benchmarks, and a real-world scenario that includes the transport system.

A. Experimental Methodology and Settings

In SCHED-T we use a sampling-based heuristic because the cost of evaluating the objective function for a single job sequence is high. Since we are introducing a new heuristic, we need to compare it with other heuristics. Publicly available instances, such as the ones described in [13] and available in [19], do not contain information about the transport system. Instances that contain the transport system, such as the ones used in [11], consider AGVs rather than a conveyor belt, so it is not possible to directly compare to their results.

For this reason, we consider the instances provided in [19] with no transport system, in order to check if our heuristic is equivalent to state-of-the-art heuristics. The objective function we evaluate is the *makespan*, *i.e.*, the time to complete the processing of all jobs. In addition, we show the machine utilization, *i.e.*, the cumulative amount of time in which the machine is used for processing tasks, divided by the makespan.

SCHED-T has three parameters – the budget at each iteration, the fraction of the budget dedicate to local and remote neighbor exploration, the number of remote neighbors, and the number of iterations. We show the sensitivity analysis with respect to these parameters and observe their impact on the quality of the solution.

SCHED-T has been implemented in Python, and the experiments are done on a 3.3 GHz Intel Core i7 with 16 Gb of RAM.

B. Instances with no transport system

We consider the set of 50 *large* instances whose main features are described in [8] (Section 5.2.3, Table 8). In particular, the number of machines across the different instances ranges from 10 to 55, and the number of jobs ranges from 13 to 106, with up to 978 total number of tasks. In [8] the optimization problem is formulated using a *Constraint Programming* (CP) model, although the high number of constraints limits the precision of the solution. The output is an interval, but such an interval has a wide gap, so comparing with that gap would provide little information. In [13], the same authors propose and evaluate the same instances with the following set of heuristics: Differential Evolution, Genetic Algorithm, Iterated Local Search, and Tabu Search. For a given instance, the solutions provided by these heuristics represent an interval whose gap is much narrow than the one found with the CP approach. Therefore we consider such a gap as a reference to compare to. Note that, given any two instances, the lower values of their gaps may have been obtained by different heuristics. For ease of comparison, we normalize the makespan of each instance with the lower value of the gap.

Makespan. Figure 2 shows the results obtained by SCHED-T. In all the cases, we set $T_{\max} = 250$ and $R = 20$ (see the sensitivity analysis described later for a detailed discussion on these parameters). For most of the instances, SCHED-T provides a makespan that is within the gap obtained with other heuristics. In some cases, SCHED-T is able to improve over other heuristics by 2%. In two cases, our heuristic obtained a larger makespan by 3%. Overall, when instances do not consider the transportation system, our solution is able to obtain a schedule with a makespan that is comparable to the one obtained by state-of-the-art heuristics.

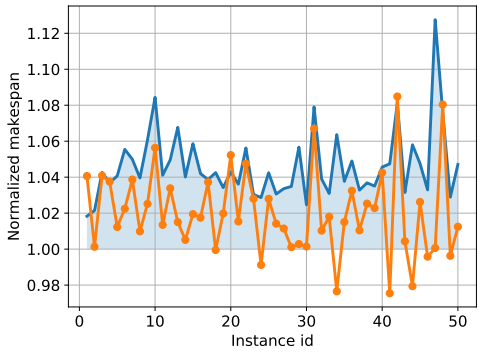

Fig. 2. Normalized makespan for the set of the 50 large instances in [8]. The shaded area represents the minimum and maximum makespan found with different heuristics in [13], while the points are the results of SCHED-T.

Machine load. While the makespan is the primary performance index, an interesting aspect to look at is how efficiently the machines are used. This may provide an indication if there is indeed room for improvement, or if some machines are underutilized. By analyzing the scheduling provided by SCHED-T, we record the average machine usage, along with the machine with the highest usage, and show the values in Figure 3. In almost all the instances, the most used machine is busy 93%-95% of the time: this indicates that probably such a machine represents the bottleneck of the system, and the potential improvement would be limited by such a bottleneck. In any case, the average machine utilization is always above 75% (80% for most of the instances), so overall the solution is able to exploit the available resources.

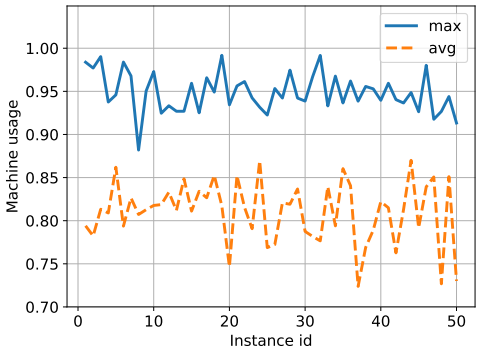

Fig. 3. Machine load for the set of the 50 large instances in [8]. The maximum value is the highest loaded machine, while the average is computed over all machines.

Sensitivity analysis. SCHED-T uses a two-level hierarchical scheme to explore the solution space. At the low level, we switch some tasks within a job or between jobs that are scheduled closely. At the higher level, we switch jobs. In our evaluation, we observed that most of the improvement is due to the higher level moves, while the scheme is less sensible to the lower level exploration. In other words, once the job order has been identified, a limited low level exploration is necessary to provide the maximum possible improvement. For this reason, the two main parameters we consider are the number of iterations $T_{\max}$ (how many moves we explore) and the number of remote neighbors R (how many neighbors we evaluate in each iteration). Both have a direct impact on the time required to compute the solution. Figure 4 shows two views in 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 fix $T_{\max}$ and we increase R (Fig. 4, left), we notice that, with values of $R > 10$ we reach a plateau. Note that, even for smaller values of R, the makespan only slightly increases (2%-5%) with respect to the best makespan found. This is confirmed also if we fix R, the number of remote neighbors, and we change $T_{\max}$, the number of iterations (Fig. 4, right) – the two figures refer to two different instances, and they are representative of the general behaviour that we observed for all the instances.

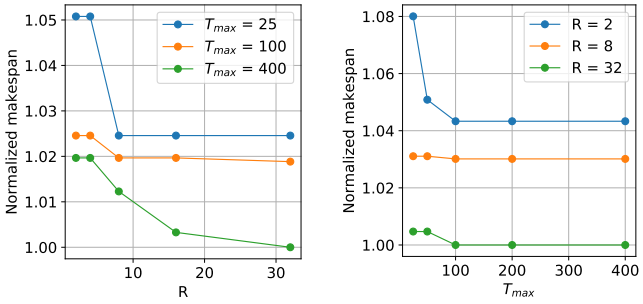

Fig. 4. Sensitivity analysis of SCHED-T: impact of the parameters on the makespan (makespan normalized to the best value found). Left and right figures refer to different instances.

In summary, SCHED-T works well with a wide range of parameter values. The solution with small $T_{\max}$ and R is obtained in less than 30 seconds, while with larger $T_{\max}$ and R it may take up to 10 minutes. But since the difference between the best makespan is limited, one may consider to use small $T_{\max}$ and R, especially in dynamic contexts, where the scheduling may need to be recomputed due to changes in the production line (machine faults) or new job arrivals – a case we do not consider here, but which is part of our future research directions.

ICE instances. We consider a set of six instances generated for our lab described in Section II. The number of jobs and tasks for each instance are provided in the Table II. Since our lab has only one machine for each specific task, we artificially doubled each machine and assume that a specific task can be performed by one of the duplicates. In this way, SCHED-T has some degree of freedom in exploring the solution locally. For these settings, we were able to solve the problem using a MILP model – see the **Opt** column in Table II.

TABLE II
ICE INSTANCES: CHARACTERISTICS (COLS. 2 AND 3), MAKESPAN WITH NO TRANSPORT (COLS. 4 AND 5), MAKESPAN WITH TRANSPORT (COL. 7).

			SCHED-T			SCHED-T	
Id	**jobs**	**tasks**	**Opt**	**no tr.**	**Err**	**with tr.**	**Diff**
1	5	32	900	900	0%	1216	35.1%
2	10	106	3580	3650	2.0%	4882	36.4%
3	15	100	2930	3014	2.9%	3109	6.1%
4	20	142	3530	3698	4.8%	4236	20.0%
5	25	174	4935	4998	1.3%	5159	4.6%
6	30	263	7475	7570	1.3%	9286	24.2%

Using SCHED-T we are able to find an approximate solution in less than 30 seconds whose difference with the optimal solution is less than 5% (columns 5 and 6 of Table II). The results are also shown in Figure 6, where the makespan is normalized with respect to the optimal solution, so the line represents the error. By looking at the machine load (see Figure 5), we observe that at least one machine is used almost 100% of the time, while the average is between 55% and 60%.

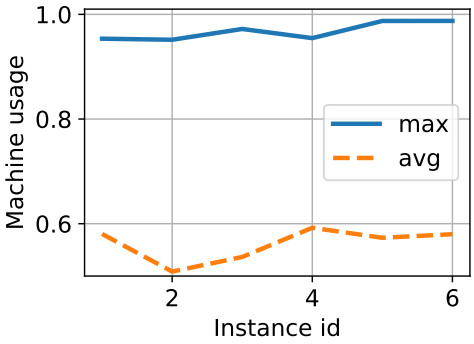

Fig. 5. Machine load for the set of ICE instances (no transportation).

C. Instances with the transport system

After having validated SCHED-T without the transport system, we are ready to evaluate it when we introduce the conveyor belt. In this case, we do not have any optimal solution to compare to. We formulated the problem with a MILP model, but we were not able to find a solution after hours of computation. Therefore, besides the makespan, we use the machine load as an indication of how efficiently the resources are used by our schedule.

Table II, column 7, shows the makespan obtained with SCHED-T. We compare this value with the optimal solution, which does not include the transport system (last column). The results are also shown in Figure 6: here the comparison can be interpreted as a difference between the case in which one computed the makespan with no transfer time and the actual makespan that will be obtained by the real system.

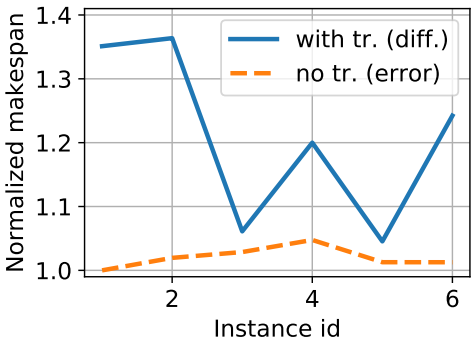

Fig. 6. Makespan for the ICE instances. When the transport system is not considered, the normalization shows the error with respect to the optimum. With the transport system, it shows the difference with respect to the optimum computed without transfer times.

In some cases (instances 3 and 5), the makespans with or without transfer times are indeed close. The analysis of the trace Gantt chart (not shown here for space constraints) reveals that the system is able to exploit the transportation while the machine are working, and it synchronizes smoothly with the processing tasks. In the other cases, the availability of the machines has a strong impact on the transfer times. These results show that the transfer has a strong impact on the overall makespan, therefore not considering them may lead to a highly imprecise estimate of the time required to process all the jobs.

If we consider the load on the machines (Figure 7) we notice that both the maximum and the average utilization are indeed decreased. The intervals in which the machines are unused are not sufficiently large to accommodate the processing of a task. These intervals could be exploited in case of heterogeneous task processing time – given a task, its processing time is not constant, and it could depend on the specific job type.

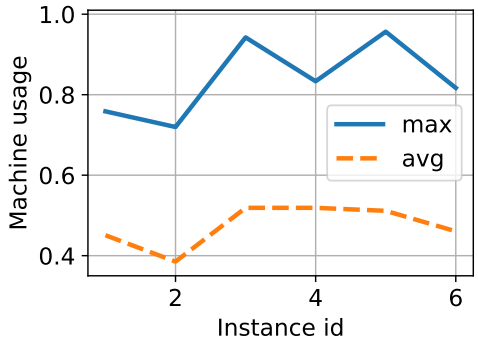

Fig. 7. Machine load for the set of ICE instances, with transportation.

As a future work, we plan to analyze the impact of heterogeneous task processing times on the efficiency of machine usage.

D. Real-world experiments

To show the accuracy of SCHED-T, we consider the set of instances described in Table II (with no machine duplication) and we run them sequentially on an actual production line built in our lab (described in Section II). For each task, we register the starting processing time and we compare it with the one computed by SCHED-T.

For fair comparison, the tasks performed by the actual machines were kept simple so that to obtain a constant processing time – the case of a stochastic execution time will be considered in the future work. For the execution, we exploit a Service-oriented Manufacturing (SOM) software architecture capable of interacting both with the Manufacturing Execution System (MES) [20] and the machines. On top of this architecture, we developed a new module that takes as input a sequence of tasks to execute and returns as output the schedule, updated with the transport time.

In addition to the real-world production line, we have also modeled our production line (*i.e.*, we have created a virtual replica) with Plant Simulation [21], a commercial state-of-the-practice discrete event simulation tool. The simulation helps us exploring cases that would otherwise take a very long time to run on the actual line.

Figure 8 shows the accuracy of the task start time, normalized to the actual start time. In the first 30 minutes, we

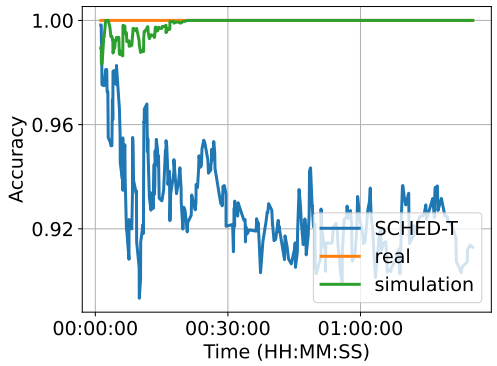

Fig. 8. Accuracy of the task start time compared to simulation and real execution.

predict the execution times (which compose the makespan) with SCHED-T, we run the same set of tasks on the actual production line, and we run the simulation. The simulation provides a high accuracy, with at most a 2% error. After 30 minutes, we kept running the simulation and take the results as a reference. As for the task start time predicted by SCHED-T, we can see that the precision of our proposed approach continuously rises and drops. This is due to the fact that our SCHED-T sometimes overestimates or underestimates the task starting times. Therefore, these negative effects balance each other, slowing the decrease of accuracy, and keeping the precision near 95% in the first 30 minutes. After 30 minutes, the accuracy of our proposed approach decreases following the same behavior of the first part. In the following hour, it slowly decreases, reaching an accuracy of 92%. This means that the variability of the system can not be predicted without taking into account the stochastic nature of the process we model. Summing up, our proposed solution allows estimating with high precision the makespan within a reasonable time frame. By estimating correctly the transfer times time, we are able to build a more precise scheduling.

VI. Conclusion

In a completely automated production environment, the transport system plays a crucial role in moving efficiently and timely the jobs among the different machines. By jointly scheduling the execution of the tasks on the machines along with their transfer, it is possible to have a fine-grained control on the overall process. In this work we propose a heuristic for solving such a problem, which adopts a randomized approach for exploring the solution space. Our solver is able to find near-optimal schedules in a limited time. This in turn opens the possibility to consider not only a static scenario, in which the jobs to be processed are known at the beginning, but also a dynamic one, in which jobs continuously arrive and the scheduling decisions are updated.

In our future work, we plan to explore these aspects, along with the differentiation of the processing time of each task, in order to exploit the available resources.

Acknowledgment

This work has been partially supported by "Progetto di Eccellenza" on Industry 4.0 of the Computer Science Department, University of Verona, Italy.

References

[1] A. Grau, M. Indri, L. L. Bello, and T. Sauter, "Industrial robotics in factory automation: From the early stage to the internet of things," in *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2017, pp. 6159–6164.
[2] I. A. Chaudhry and A. A. Khan, "A research survey: review of flexible job shop scheduling techniques," *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016.
[3] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, "Review of job shop scheduling research and its new perspectives under industry 4.0," *Journal of Intelligent Manufacturing*, vol. 30, no. 4, pp. 1809–1830, 2019.
[4] M. L. Pinedo, *Scheduling*. Springer, 2012, vol. 29.
[5] G. Zhang, J. Sun, X. Liu, G. Wang, and Y. Yang, "Solving flexible job shop scheduling problems with transportation time based on improved genetic algorithm," *Mathematical Biosciences and Engineering*, vol. 16, no. 3, pp. 1334–1347, 2019.
[6] Q. Zhang, H. Manier, and M.-A. Manier, "A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times," *Computers & Operations Research*, vol. 39, no. 7, pp. 1713–1723, 2012.
[7] H. H. Hoos and T. Stützle, *Stochastic local search: Foundations and applications*. Elsevier, 2004.
[8] W. T. Lunardi, E. G. Birgin, P. Laborie, D. P. Ronconi, and H. Voos, "Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem," *Computers & Operations Research*, vol. 123, p. 105020, 2020.
[9] "Industrial Computer Engineering (ICE) Lab," https://www.icelab.di.univr.it/.
[10] B. Naderi, M. Zandieh, A. K. G. Balagh, and V. Roshanaei, "An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness," *Expert systems with Applications*, vol. 36, no. 6, pp. 9625–9633, 2009.
[11] C. Zeng, J. Tang, and C. Yan, "Scheduling of no buffer job shop cells with blocking constraints and automated guided vehicles," *Applied Soft Computing*, vol. 24, pp. 1033–1046, 2014.
[12] Y. Sun, S.-H. Chung, X. Wen, and H.-L. Ma, "Novel robotic job-shop scheduling models with deadlock and robot movement considerations," *Transportation Research Part E: Logistics and Transportation Review*, vol. 149, p. 102273, 2021.
[13] W. T. Lunardi, E. G. Birgin, D. P. Ronconi, and H. Voos, "Metaheuristics for the online printing shop scheduling problem," *European Journal of Operational Research*, vol. 293, no. 2, pp. 419–441, 2021.
[14] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello, "The sample average approximation method for stochastic discrete optimization," *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 479–502, 2002.
[15] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.
[16] J. Yang, Y. Wang, and Z. Wang, "Efficient modeling of random sampling-based lru," in *50th International Conference on Parallel Processing*, 2021, pp. 1–11.
[17] G. W. Furnas, "Generalized fisheye views," *Acm Sigchi Bulletin*, vol. 17, no. 4, pp. 16–23, 1986.
[18] G. Pei, M. Gerla, and T.-W. Chen, "Fisheye state routing: A routing scheme for ad hoc wireless networks," in *2000 IEEE International Conference on Communications. ICC 2000. Global Convergence Through Communications. Conference Record*, vol. 1. IEEE, 2000, pp. 70–74.
[19] "FJS instance generator," https://github.com/willtl/online-printing-shop.
[20] B. Saenz de Ugarte, A. Artiba, and R. Pellerin, "Manufacturing execution system–a literature review," *Production planning and control*, vol. 20, no. 6, pp. 525–539, 2009.
[21] S. Bangsow, *Tecnomatix plant simulation*. Springer, 2020.