Scheduling Single AGV in Blocking Flow-Shop with Identical Jobs

Erik Boom[1,2], Matúš Mihalák[1][a], Frank Thuijsman[1][b] and Mark H. M. Winands[1][c]
[1]*Department of Advanced Computing Sciences, Maastricht University, Maastricht, The Netherlands*
[2]*VDL Nedcar, Born, The Netherlands*

Keywords: Scheduling, Flow-Shop, Makespan, AGV, Manufacturing, Integer Linear Programming (ILP), Heuristics.

Abstract: We consider a flow-shop with m stations (machines) and n identical jobs that need to be processed on each station. The processing time of every job on station i is p_i. After a job is processed on a station i, it needs to be transported by an automated guided vehicle (AGV) to the next station $i+1$. There is only one AGV. We assume no buffers, i.e., when the AGV transports a job to a station, the station needs to be empty. Furthermore, an AGV can transport at most one job at a time, non-preemptively, i.e., it cannot leave the job in the middle of transportation. The transportation times between the stations are given and are independent of whether the AGV carries a job or not. We study the problem of scheduling the single AGV such that all jobs are processed and the makespan is minimized. We provide a characterization of feasible schedules, and use it to derive an integer linear program (ILP) for the problem. We observe that solving the ILP requires a rather large amount of computation time even for very small instances. We use the ILP-formulation to design a rolling-window based heuristic that scales up and provides close-to-optimum schedules, as demonstrated by experimental evaluation that also involves comparison to two natural greedy algorithms.

1 INTRODUCTION

Transportation robots such as automated guided vehicles (AGVs), autonomous mobile robots (AMRs), or autonomous intelligent vehicles (AIVs) are being increasingly used in manufacturing plants to accommodate the so-called high-mix low-volume manufacturing, in which many different product types are manufactured in small quantities. In a manufacturing plant with several workstations, the variety of products often manifest itself in different production paths through the workstations. AGVs, AMRs, and AIVs are unconstrained in their movements between the workstations, compared to transportation by conveyor belts, and thus are an ideal mean to transport jobs along their individual production paths. Manufacturing environments that offer flexibility in production routes between stations are often called *flexible assembly lines*. This paper discusses only AGVs, but all the results apply to AMRs and AIVs as well.

The inherent flexibility of AGVs in movement poses scheduling challenges that are not present in the classic conveyor-belt based manufacturing: at any time, the AGV has to decide which of the possible *transportation tasks* it executes as next. Here, a transportation task is a transportation of a product from one station to the next station along the product's production path. Such decisions can quickly chain and lead to a combinatorial explosion of possible outcomes.

In this paper, we study a conceptually straightforward setting for scheduling AGVs in flexible manufacturing systems: use a *single* AGV for all transportation tasks in a manufacturing that processes n *identical jobs* (products) on m workstations, where every job $j \in \{1,\dots,n\}$ has the same production path $(1,2,\dots,m)$ through the m workstations, and the time every job j needs at workstation i is $pr(i)$, i.e., the jobs are identical and each operation of a job has a specific processing time. The single AGV can at any moment *transport at most one product*, can drop-off or pick-up a product only at the workstations, the workstations have no buffer (to store products before they are processed or after they are processed on the station) and thus a product on station i blocks this station from being used until the AGV picks-up the product at the station and moves it to the next station $i+1$. In particular, if the AGV carries a product and arrives at a station which has a finished product on it, there is no mean for the AGV to swap the two products (between

[a] https://orcid.org/0000-0002-1898-607X
[b] https://orcid.org/0000-0001-8139-8003
[c] https://orcid.org/0000-0002-0125-0824

Boom, E., Mihalák, M., Thuijsman, F. and Winands, M.
Scheduling Single AGV in Blocking Flow-Shop with Identical Jobs.
DOI: 10.5220/0012398200003639
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 13th International Conference on Operations Research and Enterprise Systems (ICORES 2024)*, pages 325-332
ISBN: 978-989-758-681-1; ISSN: 2184-4372
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

the AGV and the station).

We study the problem of minimizing the makespan – the completion time of the last job on the last workstation. Motivated by solving the problem to optimality, we first develop a non-trivial integer-linear-programming (ILP) formulation of the problem. The formulation heavily uses a simple characterization of a feasible schedule that we provide in this paper. The experiments demonstrate that even state-of-the-art commercial ILP-solvers do not scale up to solve even modestly sized instances of 25 jobs and 6 stations. To mitigate this computational hurdle, we design a heuristic that uses a moving-time-window and solves iteratively smaller sub-instances corresponding to the time-windows using the ILP formulation, obeying decisions about jobs that were made in previous iterations of the heuristic. For comparison, we also implement two straightforward greedy algorithms, and experimentally compare the algorithms.

2 RELATED WORK

There are many papers dealing with scheduling and/or routing of AGVs in manufacturing environments (Vis, 2006; Hosseini et al., 2023). The published results deal with different scheduling or routing aspects of operating manufacturing processes where transportation happens by AGVs. For example, the scheduling environment, as coined by the scheduling literature, may be parallel machines, flow shops, job shops, or open shops; the machines may have input/output buffers or not (where jobs may be queuing to be processed by a machine); one or more AGVs may be used; collisions of AGVs are/are-not taken into account; the capacity of AGVs is one or more jobs. Finally, the optimization goal can be the makespan, total completion time, latency, or tardiness.

Our scheduling problem is to minimize the makespan in a flow-shop setting with m machines and n identical jobs, without buffers, and where the jobs are transported between the stations (machines) by a single AGV, with given transportation times between any two stations. Our problem is a special case of the so-called *robotic cell problem (RCP)* (Carlier et al., 2010; Kharbeche et al., 2011). RCP is a flow-shop with no buffers and with one AGV of capacity one for the transportation tasks. Our problem is specific in that all the jobs are identical, and thus we do not need to schedule the order (permutation) of the jobs on the machines. The main results are two heuristics based on a decomposition of the problem and genetic programming (Carlier et al., 2010) and an exact branch-and-bound algorithm (Kharbeche et al., 2011). We provide an ILP formulation, and use it to derive a heuristic that scales with the number of jobs.

While the complexity of our problem is not known, it is worth noting that the makespan minimization in a flow shop with two machines, one AGV, and *unlimited buffers* is NP-hard (Kise, 1991; Hurink and Knust, 2001). For the case of zero travel times, the problem reduces to the flow-shop with blocking constraints, which was proved to be NP-hard for $m \geq 3$ by Hall and Sriskandarajah (Hall and Sriskandarajah, 1996). For the case of no buffers, three stations, and arbitrary processing times (and travel times), Hall et al. show that given a fixed periodic schedule of period 6 for a robotic arm (that can be seen as a transportation robot), the problem of sequencing the jobs such that the used fixed periodic schedule of the AGV minimizes the makespan is NP-hard (Hall et al., 1998). Obviously, this is not directly related to our problem, since do not consider sequencing of jobs (we have identical jobs), nor periodic schedules, nor constant number of stations.

In this paper, besides others, we develop an ILP formulation of the problem. There are several papers that deal with different variants of the scheduling problem with AGVs that also provide ILP-formulations. None of the ILP-formulations covers our problem. It is noteworthy to mention that the literature often reports non-scalability of using ILP-formulations to solve instances beyond tiny ones. There is one exception, and that is the work of (Fontes and Homayouni, 2019), who consider a scheduling problem with equivalent machines (any job-task can be processed on any machine) and infinite buffers and demonstrate reasonable runtimes for modestly large ILPs (4 stations, 8 jobs, 23 operations in total).

3 PROBLEM DEFINITION

We consider the makespan minimization problem in the following flow-shop setting. There are n identical jobs and m workstations. Every job needs to be processed on each workstation in the order $1, 2, \ldots, m$. In the beginning, all jobs are at a pick-up station (also called the loading station), which we refer to as station 0. Before a job can be processed at workstation i, $i \in \{1, \ldots, m\}$, it needs to be transported from station $i-1$ to station i. Transportation is performed by a single AGV. When the AGV arrives at station i, the job is moved from the AGV to the station only if the station has no other job. Upon arriving, the processing of the job starts immediately and takes $pr(i)$ time units (i.e., the moving of a job from the AGV to the station happens in no time). The processing is non-preemptive.

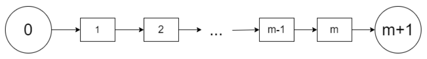

Figure 1: All jobs are at pick-up station 0, and need to be processed on the workstations $1,2,\dots,m$ in the order indicated by the arrows. All jobs finish in delivery station $m+1$.

Every workstation can only handle one job at a time, and there are no buffers at any workstation (where the jobs could be stored to wait for being processed by the workstation). This implies that whenever there is a job at a workstation, the workstation is blocked until the job is transported to the next workstation. A job that has been processed on the last workstation m needs to be eventually transported to the delivery station (also called an unloading station), which we refer to as station $m+1$. We stress that stations 0 and $m+1$ are no real workstations as no processing happens at these stations. When a job arrives at station $m+1$, the job is treated as finished and away from the overall system. See Figure 1 for a schematic overview of the flow of the jobs throughout the flow-shop.

The jobs are transported between the stations by a single AGV. The time it takes for the AGV to move from station i_1 to station i_2 is $d(i_1,i_2)$, and is independent of whether the AGV carries a job, or not. Thus, transporting a job from station i to station $i+1$ takes $d(i,i+1)$ time. Because there can be at most one job at any station, because the AGV can carry at most one job, and because we are minimizing the makespan, the only meaningful operations of the AGV are "go from current location (a station) to a station i, pick-up the job at this station, and transport it to station $i+1$", for $i \in \{0,1,\dots,m\}$. We call such an operation a *transportation task i*.

Observe that because of the blocking nature of the flow-shop, the jobs are processed on each of the stations in the same order given by the order in which the jobs enter station 1. Furthermore, because the jobs are identical, the order in which they are picked-up from station 0 has no influence on the makespan. Thus, only the order in which the jobs are transported between the stations influences the overall makespan. The time it takes to complete a transportation task i is a sum of three values: the time $d(c,i)$ that is needed to travel from the current station c to station i, the waiting time w at station i from the arrival until the job on station i is processed (it can be zero, when the processing of the job on machine i has been completed before the arrival of the AGV), and the time $d(i,i+1)$ that is needed to travel from station i to station $i+1$. It is obvious that c depends on the previous transportation task of the AGV, and w also depends on the previous transportation task(s). Thus, we want to schedule the transportation tasks for the single AGV so that the resulting makespan is minimum.

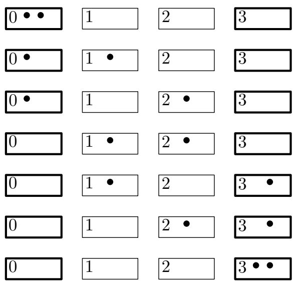

Figure 2: Execution of schedule $S = (0,1,0,2,1,2)$ for $n = 2$ jobs and $m = 2$ stations. Every row k corresponds to a state before execution of the k-th transportation task. A dot depicts a job and its position at a workstation. For example, the third row corresponds to the state $(1,0,1,0)$.

Since all jobs are identical, they can be picked from station 0 in any order, say in the order $1,2,\dots,n$, and a transportation task "transport job from station i to station $i+1$ is uniquely defined by the origin station i (the identity of a job is simply the job currently present at station i, if $i \geq 1$, or it is simply the next job in the order if $i = 0$). Hence, a schedule of transportation tasks can be represented as a sequence (ordering) $S = (s_1, s_2, \dots, s_k)$ of all transportation tasks, where $s_p \in \{0,1,\dots,m\}$, $p = 1,\dots,k$, stands for the transportation task "transport the (next) job from station s_p to station $s_p + 1$". Naturally, the AGV executes the transportation tasks in this order, as soon as possible, thus inducing completion times of jobs at the stations and thus also the overall makespan. Figure 2 gives an example of a schedule and its execution. Here, we define a state to reflect the position of all jobs just before an execution of a transportation task. Formally, for a schedule $S = (s_1, s_2, \dots, s_k)$, a state at schedule position $p \in \{1,2,\dots,k\}$ is a vector $state(p) = (n_0, n_1, \dots, n_{m+1})$, where n_a is the number of jobs at station a, $a = 0,\dots,m+1$, after the completion of the first $p-1$ transportation tasks $s_1, s_2, \dots, s_{p-1}$ from schedule S. Clearly, $0 \leq n_0, n_{m+1} \leq n$, $0 \leq n_a \leq 1$, for $a = 1\dots,m$, and $n_0 + n_1 + \cdots + n_{m+1} = n$. We will also use the notation $state(p,i) := n_i$ to denote the number of jobs at station i in $state(p)$.

Finally, we denote the *completion time* of the transportation task s_p in schedule $S = (s_1, \dots, s_p, \dots, s_k)$ by $c_p(S)$, or simply c_p if the schedule S is clear from the context.

3.1 Characterization of Schedules

Recall that every job needs to be processed on every machine i, $i = 1,\dots,m$. Thus, the transportation task $i = 0,\dots,m$ appears exactly n times in S. Hence, the length of every schedule S is $k = n \cdot (m+1)$, and every feasible schedule S is a permutation of

$(\underbrace{0,\dots,0}_{n \text{ times}},\underbrace{1,\dots,1}_{n \text{ times}},\dots,\underbrace{m,\dots,m}_{n \text{ times}})$. Not every permutation is a feasible schedule. For example, S always needs to start with the transportation task 0. In general, schedule $S=(s_1,\dots,s_k)$ is feasible, if and only if, for every $p=1,\dots,k$,

1. there is a job at station s_p in $state(p)$, i.e., $state(p,s_p)\geq 1$, and
2. if $s_p \neq m$, there is no job at station s_p+1 in $state(p)$, i.e., $state(p,s_p+1)=0$.

The claim follows, because we can transport a job from station s_p to station s_p+1 only if there is a job at station s_p and the station s_p+1 is empty (unless station s_p+1 is the delivery station).

We provide an equivalent characterization of feasible schedules. For this, we denote by $\#_p(i)$ the number of occurrences of transportation task i in S until and including position p.

Theorem 1. *Let $S=(s_1,\dots,s_k)$ be a permutation of $(\underbrace{0,\dots,0}_{n \text{ times}},\underbrace{1,\dots,1}_{n \text{ times}},\dots,\underbrace{m,\dots,m}_{n \text{ times}})$. S is feasible, if and only for every $p=1,\dots,k$, and for every $i=1,\dots,m$,*

$$\#_p(i+1)+1 \geq \#_p(i) \geq \#_p(i+1). \tag{1}$$

Proof. Let S be a feasible schedule. Let $p\in\{1,\dots,k\}$ and $i\in\{1,\dots,m\}$. Clearly, a transportation task $i+1$ can only be made if there is a job at station $i+1$. This requires that for every transportation task $s_j=i+1$ in S there needs to be in S before position j a unique transportation task i (which brings job from station i to station $i+1$), and thus $\#_p(i)\geq \#_p(i+1)$. To show that $\#_p(i+1)+1\geq \#_p(i)$, observe first that in $state(p+1)$, $state(p+1,i+1)=\#_p(i)-\#_p(i+1)$ (the number of jobs on station $i+1$ is the number of times we bring a job to $i+1$ minus the number of times we take a job from $i+1$). Since $state(p+1,i+1)\leq 1$, the inequality follows.

To prove the second implication, assume that the necessary condition is true. We will show that S is a feasible schedule. We want to show that whenever $s_p=i$, there is a job at station i and there is no job at station $i+1$ in $state(p)$. First, recall that the number of jobs at station i in $state(p)$ is $\#_{p-1}(i-1)-\#_{p-1}(i)$. We show that this quantity is at least one. Since $s_p=i$, we get that $\#_p(i)=\#_{p-1}(i)+1$. For the same reason, $\#_{p-1}(i-1)=\#_p(i)$. Using these equalities, we derive

$$\#_{p-1}(i-1)=\#_p(i-1)\geq \#_p(i)=\#_{p-1}(i)+1,$$

where the inner inequality follows from Eq. (1). Thus, the number of jobs at station i in state $state(p)$ is at least one.

Second, recall that the number of jobs at station $i+1$ in $state(p)$ is $\#_{p-1}(i)-\#_{p-1}(i+1)$. We show that this quantity is zero, or, equivalently, that $\#_{p-1}(i)\leq \#_{p-1}(i+1)$ and $\#_{p-1}(i)\geq \#_{p-1}(i+1)$. The first inequality follows directly from Eq. (1). We obtain the second inequality as follows. Since $s_p=i$, we get that $\#_{p-1}(i)=\#_p(i)-1$ and $\#_p(i+1)=\#_{p-1}(i+1)$. Then, using Eq. (1) again, we get

$$\#_{p-1}(i)=\#_p(i)-1\leq \#_p(i+1)+1-1=\#_{p-1}(i+1).$$

This concludes the proof. □

3.2 Challenges

We want to formulate the problem as an ILP. Theorem 1 gives a way to check whether a schedule $S=(s_1,\dots,s_k)$ is feasible. The characterization can be easily translated to linear inequalities in an ILP, as we will see in the next sections. The challenge is to encode in the ILP the completion time $c_p(S)$ of a transportation task s_p, $p=1,\dots,k$. Recall that $c_p(S)=d(c,i)+w+d(i,i+1)$, where c is the current position of the AGV when the transportation task s_p is to be executed, and w is the waiting time of the AGV at station s_p for the station to finish processing its job. Expressing w as a linear function of previous decisions $s_1,s_2,\dots,s_{p-1}$ for the AGV is a challenge as it needs to compare the arrival time of the AGV at station s_p with the processing time $pr(s_p)$ plus the time when the current job at station s_p arrived.

4 ALGORITHMS

4.1 ILP Formulation

In this section, we provide an ILP for our problem. We first define the variables and the objective function. We then continue with the constraints regarding the feasibility of the schedule. We finish with constraints that concern completion times and arrival times.

4.1.1 Decision Variables & Objective Function

We view the problem of computing an optimal schedule $S=(s_1,\dots,s_k)$ as an assignment problem that assigns to every of the k positions of schedule S one of the $n(m+1)$ transportation tasks. We model such an assignment by binary decision variables. We define for every $i=0,1,\dots,m$ and $p=1,2,\dots,n(m+1)$ a binary variable $x_{i,p}$ with the obvious meaning

$$x_{i,p}=\begin{cases}1 & \text{if } s_p=i,\\ 0 & \text{otherwise.}\end{cases}$$

We define c_p to be a variable that equals the completion time (in schedule encoded by variables $x_{i,p}$) of the task that has been assigned in S to position p. The objective is to minimize the makespan. Thus, the objective of the ILP is to minimize $c_{n(m+1)}$.

4.1.2 Validity Constraints

We need to ensure that variables $x_{i,p}$ encode an assignment and that it encodes a permutation of $(\underbrace{0,\ldots,0}_{n\text{ times}},\underbrace{1,\ldots,1}_{n\text{ times}},\ldots,\underbrace{m,\ldots,m}_{n\text{ times}})$. First, we make sure that for every position p, exactly one of transportation tasks is assigned to position p by constraints:

$$\forall p: \sum_{i=0}^{m} x_{i,p} = 1. \tag{2}$$

Furthermore, to make sure that variables $x_{i,p}$, $i = 0,\ldots,m$, $p = 1,\ldots,n(m+1)$, encode the desired permutation, we make sure that every transportation task $i = 0,1,\ldots,m$ is assigned exactly n times (this is equivalent to saying that all n jobs were transported from each of the $m+1$ stations $0,1,\ldots,m$):

$$\forall i: \quad \sum_{p=1}^{n(m+1)} x_{i,p} = n. \tag{3}$$

Until now, the constraints assure that the decision variables encode a permutation of $(\underbrace{0,\ldots,0}_{n\text{ times}},\underbrace{1,\ldots,1}_{n\text{ times}},\ldots,\underbrace{m,\ldots,m}_{n\text{ times}})$. We define a variable s_p that expresses the transportation task that is assigned to position p by the decision variables. We can assure this by the following linear constraints:

$$\forall p: \quad s_p = \sum_{i=0}^{m} i \cdot x_{i,p}. \tag{4}$$

We now provide linear constraints that ensure that the decision variables encode a feasible schedule. We use the characterization of feasible schedules from Theorem 1, which requires that for every position p and every station i, $\#_p(i+1)+1 \geq \#_p(i) \geq \#_p(i+1)$. Thus, we define variables $c(i,p)$ for $i = 0,\ldots,m$ and for $p = 1,\ldots,n(m+1)$ that we want to be equal to $\#_p(i)$. We can assure this by the following linear constraints:

$$\forall i \forall p: \quad c(i,p) = \sum_{q=1}^{p} x_{i,q}. \tag{5}$$

Assuring the feasibility of the computed schedule according to Theorem 1 is now straightforward:

$$\forall p \forall i < m: \quad c(i+1,p)+1 \geq c(i,p) \tag{6}$$

$$\forall p \forall i < m: \quad c(i,p) \geq c(i+1,p) \tag{7}$$

4.1.3 Time-Related Constraints

Recall that the completion time c_p of the transportation task s_p that is assigned to position p depends on the pick-up station i that is assigned to s_p, but it also depends on the completion time and delivery location of the preceding transportation task s_{p-1}, and on the time when the job that is going to be transported from station s_p has been processed. Following this observation, we define two new variables. We define for every position p the variable $l(p)$ that equals the station (location) at which the AGV is present when it starts executing the transportation task s_p. Obviously, $l(p)$ is the delivery station of the previous transportation task s_{p-1}, i.e., $l(p)$ should equal to $s_{p-1}+1$. We can ensure this with the following constraints:

$$\forall p: \quad l(p) = s_{p-1}+1, \tag{8}$$

where $s_{-1} := 0$ is the initial position of the AGV before we start any transportation.

The AGV can start the transportation task s_p as soon as possible, but not earlier than (i) the arrival time of the AGV at station s_p, i.e., time $c_{p-1} + d(l(p),s_p)$, and not earlier than (ii) the time when the job at station s_p has been processed, i.e., time $c_{pred(p)} + pr(s_p)$, where $pred(p)$ is the last previous transportation task s_q that delivers a job to station s_p. We define a $pred(p)$ for every $p > 1$ as follows:

$$pred(p) = \begin{cases} \max_{q<p} q : s_q = s_p - 1 & \text{if } s_p \neq 0 \\ p-1 & \text{if } s_p = 0 \end{cases} \tag{9}$$

We can now express the completion time c_p of the task s_p at position p. Since the first task s_1 is always to take a job from station 0 to station 1, we set

$$c_1 = d(0,1). \tag{10}$$

For $p > 1$, we define

$$\forall p > 1: c_p = \max\{c_{p-1} + d(l(p),s_p), \; c_{pred(p)} + pr(s_p)\} + d(s_p, s_p+1). \tag{11}$$

In this equation, $d(s_p,s_p+1)$ is the time it takes to do the actual transportation. The max operator gives the earliest time when the task can be started. The job arrives at station s_p at time $c_{pred(p)}$ and takes $pr(s_p)$ time to be processed. Thus, the transportation task cannot start before the processing is finished at time $c_{pred(p)} + pr(s_p)$. Expression $c_{p-1} + d(l(p),s_p)$ is the earliest time when the AGV is able to start the task, since the AGV is required to finish the preceding task (which occurs at time c_{p-1}), and then travel to station s_p. Since c_p is the maximum of linear functions, the above constraints can be expressed by linear inequalities in a standard way, if we can express

$c_{pred(p)}$, $d(s_p, s_p + 1)$, and $d(l(p), s_p)$, respectively, by linear constraints. To do so, we introduce new variables c_p^{pred}, d_p and d_p^{ℓ} that get the values, respectively, of the just mentioned expressions. We can set these variables to the right values by non-trivial yet standard modeling techniques. Due to space constraints, we defer the details to the full version of the paper.

These constraints now fully describe the ILP-formulation of the scheduling problem.

4.2 Time-Window Heuristic

It turns out that state-of-the-art ILP-solvers such as Gurobi (Gurobi Optimization, LLC, 2023) do not solve the ILP for instances of 25 jobs and 6 stations within few hours. Since ILP can be solved relatively quickly for smaller instances, we use this to design a heuristic that decomposes the problem of computing schedule $S = (s_1, \dots, s_{n(m+1)})$ at once into several subproblems of computing a contiguous part of S of size w; here, we call w the time-window of the heuristic.

Formally, the *time-window heuristic*, parameterized by a positive integer $w < n(m+1)$, is an iterative algorithm that in iteration $i = 0, 1, 2, \dots$ computes the sub-schedule $(s_{1+w\cdot i}, \dots, s_{w\cdot(i+1)})$ of schedule S using the ILP formulation that only takes into account the decisions (variables) relevant to the considered sub-schedule. Naturally, we modify the ILP so that the state from previous decisions $(s_1, \dots, s_{i\cdot w})$ are taken into account (basically, we turn the respective variables from previous iterations into constants in the current iteration). We set the objective of the ILP for the subproblem to be the time it takes to execute the transportation tasks of the considered time-window, i.e., the makespan of that sub-problem.

The size of the time-window influences the runtime and the quality of the computed schedule. We will experimentally evaluate this later in the paper.

4.3 Greedy Algorithms

To get an insight into the performance of the time-window heuristic for larger instances, we compare it to two straightforward greedy algorithms. Both greedy algorithms compute the schedule $s = (s_1, \dots, s_{n(m+1)})$ iteratively task-by-task, starting with task s_1. Both greedy algorithms choose the task s_i in iteration i to be the task among all tasks that keeps the schedule feasible (according to Theorem 1) that minimizes one of the following greedy scores:

- Time it takes for the AGV to start task s_i. It is the travel time of the AGV from current location to station s_i plus the waiting time for the job at s_i to be finished.
- The completion time of task s_i. It is the time to start task s_i plus the travel time $d(s_i, s_i + 1)$.

We denote by Greedy-start and Greedy-finish the greedy algorithm that uses the first and the second greedy score, respectively.

Table 1: Runtime of the ILP on the randomized instances. The mean of the runtime T is measured in seconds.

n	m + 1	r	T[mean]	T[var]
6	5	0.1	81	1381
6	5	0.4	107	4570
6	5	0.8	205	48786
6	5	1.1	173	29053
7	4	0.1	7.92	11.00
7	4	0.4	4.15	1.70
7	4	0.8	4.97	7.03
7	4	1.1	5.96	8.42
7	5	0.1	884	313628
7	5	0.4	319	59471
7	5	0.8	366	62302
7	5	1.1	348	34389

5 EXPERIMENTS

Set-up. We experimentally evaluate the exact ILP-based algorithm, the ILP-based heuristic, and the two greedy algorithms. First, we measure the run-time of the ILP for small instances of varying values of n and m. To investigate the influence of processing times and travel times on the runtime, we create, for each combination of n and m, random instances where the average of the generated processing times is a r-fraction of the average of the generated travel times; here, r is a parameter. Concretely, we draw travel times from a uniform distribution taking integer values between 15 and 25 such that the mean is 20. For given parameter r, we draw processing times from a uniform distribution taking integer values between 0 and $r \cdot 40$ such that the mean is $r \cdot 20$. For each combination of n, m, and r we draw ten instances, and we report the average of the observed values. We note that the generated travel times are truly random, and in particular may not satisfy the triangle inequality. Second, we run the time-window heuristic and the greedy algorithms on the generated instances. Third, we also generate random instances as described above for larger values of n and m, and run the heuristic and the greedy-algorithms on these instances.

We run the experiments on a laptop with 8GB of RAM, and with a quad-core Intel i5 CPU of the 11th generation. We use the Gurobi solver and its Python interface for ILP-related experiments.

Results. The results of the experiments are presented in Tables 1, 2, and 3. Table 1 shows that increasing m has larger effect on the run-time than increasing n.

Table 2: Experiments for smaller instances. ρ is the ratio between the makespan of the algorithm and the optimum makespan.

Algorithm	n	$m+1$	r	w	ρ[mean]	ρ[variance]	T[mean]	T[variance]
TW	6	5	0.1	1k	0.99	0.000	0.18	0.001
TW	6	5	0.1	3k	1.00	0.000	1.96	0.235
Greedy-start	6	5	0.1		0.99	0.000	0.00	0.000
Greedy-finish	6	5	0.1		0.99	0.000	0.000	0.000
TW	6	5	0.4	1k	0.94	0.003	0.17	0.002
TW	6	5	0.4	3k	0.98	0.000	2.19	0.186
Greedy-start	6	5	0.4		0.92	0.002	0.00	0.000
Greedy-finish	6	5	0.4		0.87	0.003	0.00	0.000
TW	6	5	0.8	1k	0.98	0.002	0.138	0.001
TW	6	5	0.8	3k	0.99	0.000	3.82	1.071
Greedy-start	6	5	0.8		0.90	0.006	0.00	0.000
Greedy-finish	6	5	0.8		0.87	0.008	0.00	0.000
TW	6	5	1.1	1k	0.95	0.004	0.12	0.001
TW	6	5	1.1	3k	0.99	0.000	2.70	1.010
Greedy-start	6	5	1.1		0.90	0.002	0.00	0.000
Greedy-finish	6	5	1.1		0.87	0.007	0.00	0.000
TW	7	5	0.1	1k	0.99	0.001	0.23	0.002
TW	7	5	0.1	3k	0.99	0.000	8.48	9.184
Greedy-start	7	5	0.1		0.99	0.000	0.00	0.000
Greedy-finish	7	5	0.1		0.99	0.000	0.00	0.000
TW	7	5	0.4	1k	0.93	0.004	0.23	0.004
TW	7	5	0.4	3k	0.97	0.001	5.12	7.173
Greedy-start	7	5	0.4		0.91	0.003	0.00	0.000
Greedy-finish	7	5	0.4		0.87	0.004	0.00	0.000
TW	7	5	0.8	1k	0.98	0.002	0.18	0.001
TW	7	5	0.8	3k	0.99	0.000	6.64	12.161
Greedy-start	7	5	0.8		0.90	0.007	0.00	0.000
Greedy-finish	7	5	0.8		0.86	0.009	0.00	0.000
TW	7	5	1.1	1k	0.95	0.004	0.16	0.001
TW	7	5	1.1	3k	0.98	0.001	5.40	4.238
Greedy-start	7	5	1.1		0.90	0.002	0.00	0.000
Greedy-finish	7	5	1.1		0.87	0.007	0.00	0.000

Table 3: Experiments for larger instances ($n = 25$, $m+1 = 6$). M is the makespan of the computed schedule.

Algorithm	r	w	T[mean]	T[variance]	M[mean]	M[variance]
TW	0.1	1k	11.72	1.567	903.11	11221.111
TW	0.1	2k	45.23	1099.14	885.44	10802.28
TW	0.1	3k	1075.13	908646.03	885.33	10853.25
Greedy-start	0.1		0.00	0.00	896.44	11133.03
Greedy-finish	0.1		0.00	0.00	913.44	13209.03
TW	0.4	1k	11.75	0.82	2461.70	433514.90
TW	0.4	2k	35.85	283.70	2375.90	267671.21
TW	0.4	3k	819.37	146320.90	2347.90	247904.32
Greedy-start	0.4		0.00	0.00	2579.30	305117.57
Greedy-finish	0.4		0.00	0.00	2708.70	199664.90
TW	0.8	1k	11.33	0.66	4176.30	345108.01
TW	0.8	2k	39.67	487.16	3912.30	291342.90
TW	0.8	3k	662.17	238831.36	3846.50	276170.28
Greedy-start	0.8		0.00	0.00	4288.00	433658.22
Greedy-finish	0.8		0.00	0.00	4596.70	725442.23
TW	1.1	1k	10.84	0.46	5646.30	1468051.12
TW	1.1	2k	28.19	131.33	5359.20	1314365.96
TW	1.1	3k	378.68	11430.71	5280.70	1198168.01
Greedy-start	1.1		0.00	0.00	5872.50	858943.17
Greedy-finish	1.1		0.00	0.00	6499.00	1427400.67

In any case, experiments (not reported in the table) show that scheduling around 25 jobs requires, typically, several hours. The experiments do not reveal any monotonous effect of r on runtime.

Tables 2 and 3 show the performance of the time-window heuristic and the two greedy algorithms.

Overall, for small instances, the time-window heuristic gets close to the optimal makespan, not dropping below 93% with a window size of $1k$, and not dropping below 97% with a window size of $3k$. It is not surprising to see that increasing the time-window w leads to better schedules. The running times on the larger instances are reasonable, even when taking a larger window size.

The greedy algorithms do not perform well, specifically for larger ratios r of travel/processing time. It is interesting to note that for larger instances, the algorithm taking the earliest time to start a task (Greedy-start) is consistently giving a better solution on average than the other algorithm (Greedy-finish). It would be interesting to investigate whether it holds up for other cases as well.

6 CONCLUSION

In this paper, we considered a flow-shop setting with identical jobs and no buffers operated by a single AGV, and studied the makespan minimization problem of scheduling the transportation tasks of the single AGV. Surprisingly, this problem has not been studied in the literature before. We provided a characterization of feasible schedules, and used it to design an ILP formulation of the problem. We observed that even state-of-the-art ILP solvers cannot solve more than few stations and jobs within hours of computation time, and resorted to heuristics. We used the ILP-formulation as a subroutine and developed a time-window based heuristic that decomposes the scheduling problem into several subproblems defined by time-windows of fixed size, and solve the subproblems using our ILP formulation. We experimentally evaluated this approach with two simple greedy algorithms and, for small instances, also with optimum algorithms (based on the ILP formulation). We observed that the time-window heuristic performs well both in the run-time as well as in the quality of computed schedules.

We have done few more experiments that due to space constraints were not presented in this paper. We plan to do more experiments and report on these in a full version of the paper. Furthermore, we note that all our algorithms can be applied to the setting where the travel time depends on whether the AGV carries a job or not.

For future research, we would like to settle the complexity question of deciding whether the scheduling problem is NP-hard. Furthermore, we aim to test the developed heuristic in real-world scenario in one of the demo manufacturing lines of VDL Nedcar.

ACKNOWLEDGEMENTS

This research has received financial support from the Ministry of Economic Affairs and Climate, under the grant "R&D Mobility Sectors" carried out by the Netherlands Enterprise Agency. We also want to thank VDL Nedcar for fruitful discussions and for the research question based on their real-world use-case.

REFERENCES

Carlier, J., Haouari, M., Kharbeche, M., and Moukrim, A. (2010). An optimization-based heuristic for the robotic cell problem. *European Journal of Operational Research*, 202(3):636–645.

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

Gurobi Optimization, LLC (2023). Gurobi Optimizer Reference Manual.

Hall, N. G., Kamoun, H., and Sriskandarajah, C. (1998). Scheduling in robotic cells: Complexity and steady state analysis. *European Journal of Operational Research*, 109(1):43–65.

Hall, N. G. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525.

Hosseini, A., Otto, A., and Pesch, E. (2023). Scheduling in manufacturing with transportation: Classification and solution techniques. *European Journal of Operational Research*.

Hurink, J. and Knust, S. (2001). Makespan minimization for flow-shop problems with transportation times and a single robot. *Discrete Applied Mathematics*, 112(1):199–216. Combinatorial Optimization Symposium, Selected Papers.

Kharbeche, M., Carlier, J., Haouari, M., and Moukrim, A. (2011). Exact methods for the robotic cell problem. *Flexible Services and Manufacturing Journal*, 23:242–261.

Kise, H. (1991). On an automated two-machine flowshop scheduling problem with infinite buffer. *Journal of the Operations Research Society of Japan*, 34(3):354–361.

Vis, I. F. (2006). Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709.