Nested Monte-Carlo Search for Scheduling an Automated Guided Vehicle in a Blocking Job-Shop

Eduard von Bothmer, Matúš Mihalák[1111−2222−3333−4444], and Mark H. M. Winands[0000−0002−0125−0824]

Department of Advanced Computing Sciences,
Maastricht University, Maastricht, The Netherlands

Abstract. We consider a job-shop scheduling problem where a single automated guided vehicle (AGV) transports jobs between workstations. There are multiple but few job types, each with a specific path through the workstations and a specific processing time per workstation. The AGV can transport one job at a time, and there are no buffers, meaning each workstation must be empty before the AGV can deliver a new job. The goal is to schedule the AGV to minimize the makespan, which is the time when the last job is processed. We provide an Integer Linear Programming (ILP) formulation to find an optimal solution, and observe that within few minutes it can only solve small instances. As a remedy, we design a heuristic algorithm using the Nested Monte-Carlo Search (NMCS) paradigm. We compare its performance with two greedy algorithms and a local search approach. In the experiments, the NMCS-heuristic significantly outperforms traditional ILP methods and greedy algorithms under limited time resources.

Keywords: Scheduling · AGV · Makespan · Nested Monte-Carlo Search

1 Introduction

Automated guided vehicles (AGVs) are increasingly being used in modern manufacturing plants to enable high-mix, low-volume production, where diverse product types are manufactured in small quantities [17]. Their task is to transport products from one workstation to another in the production process. AGVs and the related Autonomous Mobile Robots (AMRs) and Autonomous Intelligent Vehicles (AIVs), offer flexibility in navigating between workstations, unlike traditional conveyor belts. This flexibility makes them ideal for transporting jobs along different production paths within assembly lines.

The flexibility of AGVs introduces unique scheduling challenges that are absent in conveyor-based systems: AGVs need to decide which product to transport next taking into account the distances between the workstations and times needed to process products at workstations. These decisions influence the performance of the system, which is typically measured in *makespan* – the total time needed to produce all products [18].

Many of the underlying scheduling problems are NP-hard in general [5, 13], even with a single AGV, if every product has individual processing times at workstations. For the specific case of a single type of product (the processing time at a station is the same for every product), the NP-hardness is an open problem, but the problem remains difficult to solve (to optimality) for relatively small instances, even when the scheduling concerns only one AGV [4]. In this paper, motivated by a real-world case of a small production line with one AGV, we build upon the previous work of Boom et al. [4] and develop quick heuristics that compute good schedules for a setting with relatively small number of different types of jobs. Concretely, we study the following scheduling problem: there is a single AGV in a job-shop environment where multiple jobs (products) of few different types are to be processed on several workstations. Each job type has a predetermined path through the workstations, and every job of that type requires transportation between the corresponding workstations by the AGV, which can only carry one job at a time. The stations have no buffers: a job blocks a station from processing next job until the AGV moves the processed job to the next station. The goal is to minimize the overall time needed to process all jobs. While being a fundamental problem to be solved, it also describes an industrial scenario of a demo-line of a car-manufacturing company VDL Nedcar, which actually triggered our interest in this setting.

In this paper, we adapt the Nested Monte-Carlo Search technique [7] to our optimization problem. We guide the search with the help of greedy roll-outs. To evaluate the performance of the developed heuristic approach, we compare it with the performance of commercial solvers on integer-linear-programming (ILP) formulations of the problem, and with the performance of natural greedy algorithms. For this, we adapt the ILP-formulation of Boom et al. [4] of the single-job-type single-AGV setting to our setting with several job types. We also adapt the greedy algorithms of Boom et al. [4] to our setting; this is a non-trivial adaptation, since, as we observe, for multiple job-types, standard greedy approaches may lead to deadlocks. Finally, we also design a local-search heuristic and evaluate its performance as a post-processing heuristic.

The results demonstrate effectiveness of our heuristic algorithm compared to traditional ILP methods and greedy algorithms. The results prove the potential of our heuristic for real-world applications, particularly in settings where quick and efficient scheduling decisions are critical. Additionally, besides the setting with multiple job-types, we tested our heuristic in the single job-type setting, and observe that our heuristic performs better that the heuristics of [4].

Paper organization. In Section 2, we formally define the problem. We then review the related literature in Section 3. We describe the ILP formulation and the developed algorithms in Section 4. The experiments are described and presented in Section 5 and 6, respectively. We conclude the paper in Section 7.

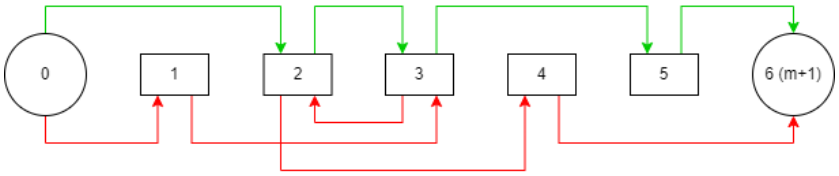

Fig. 1. In this instance of the problem, there is two types of jobs: the one following the green path and the one on the red path.

2 Problem Definition

We consider the makespan minimization problem in the following job-shop setting. There are m workstations and n jobs. Each job is one of t types. A *workstation path* is a sequence of a subset of workstations. Every job of type $k = 1, ..., t$ needs to be processed on the workstations given by a given workstation path P_k, in the order as they appear in P_k. In the beginning, all jobs are at a starting station, which will be referred to as the *source station*, or station 0. Before a job of type k can be processed at station $i \in P_k$, it needs to be transported from the previous station on the path P_k, which we denote as $\text{prev}(i, k)$, to station i.

A single AGV performs every transportation task. The AGV can transport only one job at a time. When the AGV arrives with a job at station i, $1 \leq i \leq m$, the job can be moved from the AGV to the station only if the station has no job on it. That is, we assume that a job from a station cannot be swapped with a job on the AGV, we assume that there is no (storage) buffer at stations 1,2,. . . ,m, where jobs can be temporarily stored, and we assume that every workstation can process at most one job at a time. Upon moving a job of type k to station i, the processing of the jobs starts immediately, and takes time $pr(i, k)$. We assume, without loss of generality, that the time it takes to load/unload the job to/from the AGV at any station is zero. The AGV can move, with or without a job on it, from any station to any other station. It takes time $d(i, i')$ to move from station i to station i'.

The processing is non-preemptive, i.e., it cannot be interrupted. After a job has been processed by the last station on its workstation path, it needs to be transported to the so called *sink station*, which we also call the station $m + 1$. At this moment, the job is fully processed and considered leaving the system. Equivalently, we can consider station $m + 1$ to be able to store an arbitrary number of jobs. Station $m+1$ does not appear in the path of any job type. However, every workstation path starts with station 0 (the source station). Figure 1 illustrates, schematically, the setting with two types of jobs.

We seek a schedule for the AGV to transport all the jobs along their workstation paths such that the time when the last job is processed and transported to the sink station $m + 1$ is minimized. The schedule is not bound to transport a job along its entire workstation path before doing anything else: the schedule can interleave transportation tasks of various jobs and of various types. Since

the AGV can carry at most one job at a time, a schedule can be expressed as a sequence of transportation tasks, which are executed by the AGV in the given order. Here, a transportation task consists in transporting a job j of type k from station i to the next station on j-th workstation path P_k, denoted by $\text{next}(i, k)$. We denote by a tuple (i, k) the transportation task of transporting the job of type k from station i to the next station along P_k. Hence, a schedule is a sequence $S = ((s_0, k_0), (s_1, k_1), \ldots, (s_p, k_p), \ldots)$ of transportation tasks, such that for every job of type k and for every $i \in P_k$ there is in S one transportation task (i, k) (recall that station 0 is always in P_k). Thus, the size of S is $\sum_k n_k \cdot |P_k|$, where n_k is the number of jobs of type k. Naturally, not every permutation of the mentioned transportation tasks is a valid schedule; transportation task (s, k) at position p of S is valid if the station s contains a job of type k, and the station $\text{next}(s, k)$ has no job on it.

The time needed to execute a transportation task (i, k) is a sum of the following times:

- time $d(L, i)$ from the AGV's current location (station) L to station i,
- waiting time W of the AGV at station i for the the job at station i to be processed. This value can also be 0 in case processing finished before the AGV arrived at station i,
- transportation time $d(i, \text{next}(i, k))$.

Notice that both values L and W depend on the previous transportation tasks performed by the AGV. The completion time of the transportation task (s_p, k_p) at position p in schedule S, for $p = 0, \ldots, |S|$, is denoted by C_p. The objective is to minimize the makespan, i.e., the completion time $C_{|S|-1}$.

3 Related Work

Several papers deal with the topic of scheduling AGVs in a manufacturing environment [15, 20, 25]. Typically, the literature deals with different concrete manufacturing settings where transportation happens via AGVs or the like. Our problem with one type of jobs is a special instance of the *robotic-cell problem* [5] (RCP). In RCP, all jobs need to be processed by a set of machines in the same order, with different processing times per machine and per job, with no *buffers* in between stations and with one AGV transporting the jobs from one machine to another. RCP where every transportation is instant (takes zero time) becomes a well-studied flow-shop problem, which is known to be NP-hard [13]. Our problem with one type of jobs is a special case of RCP, since in RCP, every job can be of a unique type. It has been introduced and studied by Boom et al. [4], who provided a characterization of valid schedules, developed an ILP formulation of the problem, observed its computational infeasibility, and developed and compared greedy algorithms and a heuristic based on iterative solving of smaller subproblems via ILP. Our problem with several job types allows for different workstation paths, which takes the setting away from RCP (and the related flow-shop).

Our setting does not contain buffers at the station. There are relatively many papers that considered buffers. For example, Caumond et al. [6] develops a general mathematical framework and an ILP formulation for a general scheduling setting with buffers of certain capacities at stations. Similar problems have been studied with the assumption of infinite input/output buffer space at each machine [3, 12, 10, 27, 6] or transfer stations with unlimited buffer space [19].

Since we allow arbitrary workstation paths, our problem is a job-shop scheduling problem with job-transportation via single AGV, without buffers at stations, and a limited number of job types.

A problem similar to ours is described in [14], where a flexible re-entrant job-shop with blocking and two AGVs is addressed and an MILP is developed, and solutions for small instances are presented. In our problem, we consider one AGV, and we do not consider re-entering jobs (i.e., jobs that go through the same machine more than once) and we aim to find a feasible and close-to optimum schedules solution for larger instances in a limited amount of time (than what an MILP can provide).

Monte-Carlo search algorithms have been used quite successfully in optimization and scheduling domains [1, 11]. Especially nested Monte-Carlo [7, 9] search has shown promising results in applications ranging from bus network regulation [8], the traveling-salesperson problem with time windows [21], the snake-in-the-box problem [16] and even finding the ground state energy of proteins [23]. A subset of the authors used a hierarchical Monte-Carlo tree-search to find good orderings of jobs for a black-box scheduler of a manufacturing line [26].

4 Methods

In the following sections we describe the algorithms that we designed and implemented for our scheduling problem.

4.1 ILP Formulation

ILP formulation expresses our scheduling problem as a minimization problem of a linear function over integer variables, subject to linear constraints (inequalities) over integer variables. First, we describe the objective function, the main variables and how they encode a schedule. Afterwards, we describe linear constraints that ensure the variables encode a valid schedule.

Decision Variables and Objective Function. We view the scheduling problem as the problem of deciding, for every position (index) $p = 1, 2, \ldots, |S|$ in the schedule S, which transportation task to execute at position p. We model such a decision by a binary decision variable

$$X_{i,k,p} \in \{0, 1\}, \tag{1}$$

which has value of 1 if transportation task (i, k) is assigned to position p, and it has value of 0 otherwise. We define such a decision variable for all combinations of values of i, k, and p.

Moreover, we define the variable C_p, $p = 1, \ldots, |S|$, representing the completion time of the task assigned to position p in schedule S.

The objective function that we want to minimize is $C_{|S|}$.

Validity Constraints. We first assure that exactly one transportation task is assigned to each position p in S. We can assure this with the following equalities:

$$\forall p : \sum_{k} \sum_{i \in P_k} X_{i,k,p} = 1 \tag{2}$$

Moreover, for every type k and every station $i \in P_k$, we need to schedule n_k many tasks (i, k) in S:

$$\forall k, i \in P_k : \sum_{p} X_{i,k,p} = n_k \tag{3}$$

The above constraints ensure that the variables $X_{i,k,p}$ model a permutation of all the required transportation tasks. We further need to ensure that the permutation modeled by variables $X_{i,k,p}$ is a valid schedule. Specifically, transportation task (i, k) can be positioned at p in schedule S if:

- There is a job of type k at station i. This means that, up until and not including position p, the number of occurrences of task $(prev(i, k), k)$ (how many times did a job of type k arrive at station i) is one larger than the number of occurrences of task (i, k) (how many times did a job of type k leave station i). We get the following *conditional* constraint:[1]

$$\forall i, k, p : X_{i,k,p} = 1 \implies \sum_{q<p} X_{prev(i,k),k,q} - \sum_{r<p} X_{i,k,r} = 1 \tag{4}$$

- There is no job at the station $next(i, k)$. That is, the number of times a job is moved to station $next(i, k)$ needs to be equal to the number of times a job is move from station $next(i, k)$. Let T be the set of tuples (i', k') such that $next(i', k') = next(i, k)$. Then, we get the following constraints

$$\forall i, k, p : X_{i,k,p} = 1 \implies \sum_{q=0}^{p-1} \sum_{(i',k') \in T} X_{i',k',q} - \sum_{r=0}^{p-1} \sum_{k'=1}^{t} X_{\mathrm{next}(i,k),k,r} = 0. \tag{5}$$

[1] Formally, conditional constraints are not part of an ILP formulation; however, standard modeling techniques can transform the conditional constraints to non-conditional, and the ILP solver that we used does this automatically.

Completion-Time Constraints. We now present constraints that assure that the value of variable C_p is equal to the completion time of the transportation task that is schedule at position p in schedule S. We relate the value of C_p to the value of the previous variable C_{p-1}. We start with the constraint for C_1 (recall that the AGV starts at station 0):

$$C_1 = \sum_k X_{0,k,1} \cdot d(0, next(0,k)) \tag{6}$$

Assume that the transportation task at position p is (i,k). Then, the value of C_p for $p > 1$ needs to satisfy the following two constraints. First, C_p is at least time C_{p-1} (time when AGV finished the previous task $p-1$) plus the time to travel from the drop-off station of transportation task $p-1$ to the pick-up station of task p plus the time to further travel to the drop-off station of task p. We get the following constraint

$$C_p \geq C_{p-1} + d(\text{next}(i',k'), i) + d(i, next(i,k)), \tag{7}$$

where (i',k') is the transportation task at position $p-1$. Second, if the AGV needs to wait for a job to finish at station i, then C_p will be the time when the job has been delivered to station i, plus its processing time, plus the time needed to deliver the job to its next station. We get the following constraint

$$C_p \geq PC_p + pr(i,k) + d(i, \text{next}(i,k)), \tag{8}$$

where PC_p is the time when the job at station i has been brought to station i, which is equal to some completion time $C_{p'}$ of a transportation task $p' < p$. Naturally, PC_p is a variable, and can be computed by the following constraints:

$$\forall i,k,p,q < p : PC_p \geq C_q - (1 - Y_{i,k,p,q}) \cdot M, \tag{9}$$

where M is a large constant and $Y_{i,k,p,q} \in \{0,1\}$ is a binary variable, which takes value 1 if task (i,k) is assigned to position p and the latest instance of task $(prev(i,k),k)$ is assigned to position q. The constraints for $Y_{i,k,p,q}$ are

$$\forall i,k,p,q : Y_{i,k,p,q} \geq X_{prev(i,k),k,q} + X_{i,k,p} - 1 - \sum_{q+1 \leq r < p} X_{prev(i,k),k,r}. \tag{10}$$

4.2 Greedy Algorithms

Two greedy algorithms were implemented as a benchmark for the heuristic solution. Both greedy algorithms create the schedule S iteratively, choosing the next transportation task among all valid choices according to a greedy criterion. The greedy criteria of our greedy algorithms follow the same principle as the ones described by [4] for the setting with one job type. However, with several job types, greedy algorithms can lead to a *deadlock*, i.e., a situation when schedule is not finished, but there is no valid choice for a transportation task to be made.

To illustrate a deadlock, consider the setting in Figure 1 as a graph, with stations as vertices and paths as edges. A deadlock happens when there is a cycle in the graph, and all the stations in the cycle have a job on them. In our example, if station 2 has a job following the green path and station 3 has a job following the red path in Figure 1, neither of the two jobs can be transported to the next station. This means that, if an algorithm is generating a schedule step-by-step and a deadlock is created after performing action at position p, the algorithm will not find any feasible tasks that can be performed at position $p+1$.

For this reason, we enhance the greedy algorithms to backtrack if they get stuck in a deadlock. We use the following two greedy criteria for choosing the next transportation task:

- Pick a valid transportation task that the AGV can start earliest. This is the travel time from the current location of the AGV to the pick-up location for the task and the eventual waiting time for the job to finish processing.
- Pick a valid transportation task (i, k) that the AGV can complete the earliest. This the time needed to start the task (the greedy criterion above) plus the travel time $d(i, \text{next}(i, k))$.

We refer to the greedy algorithms as GreedyStart and GreedyFinish, respectively. In our experimental evaluation, GreedyStart consistently outperforms GreedyFinish (for around 25 different instances, GreedyStart performed the same or better than GreedyFinish in 23 instances, and in the other 2 cases, it was at most 0.4 percent worse; for the lack of space we do not display the exact comparison). In the remainder of the paper, we chose to only use GreedyStart when using a greedy algorithm.

4.3 Iterative Nested Monte-Carlo Search

Iterative Nested Monte-Carlo Search (INMCS) alorithm iteratively performs multiple runs of Nested Monte Carlo Search (NMCS) [7, 1], which is an algorithm designed to improve efficiency of search processes when no effective heuristic is available to guide such search. Our problem involves searching for a close-to-optimal schedule, which NMCS achieves by iteratively generating the schedule task-by-task. This is accomplished by generating several schedules randomly or following a heuristic and by keeping track of the tasks that were performed by the best performing generated schedule. This schedule generation is performed on multiple levels, where the higher level stores the best solution found on the lower level. Specifically, Algorithm 1, adapted from [7], describes the basic implementation of the Nested Monte-Carlo Search (NMCS), where

- *task* is the transportation task that is currently executed,
- *play(p, t)* executes a given task t at a given position p in the schedule,
- *sample(play(p, t))* after a transportation task has been executed at position p, sample() generates the full schedule from $p + 1$ and returns its makespan.

INMCS iteratively performs several runs of the NMCS algorithm until the result converges. Moreover, in our implementation, the best found solution in

Algorithm 1 Nested Monte Carlo Search with input (position, level)

```
1: bestScore ← −1
2: while time is available and schedule is not fully generated do
3:     if level = 1 then
4:         task ← argmax_tasks(sample(play(position, task)))
5:     else
6:         task ← argmax_tasks(nested(play(position, task), level − 1))
7:     end if
8:     if score of task > bestScore then
9:         bestScore ← score of task
10:        bestSequence ← sequence after task
11:    end if
12:    bestTask ← task of bestSequence
13:    position ← play(position, bestTask)
14: end while
15: return bestScore
```

the previous run of the NMCS algorithm is assigned as best sequence in the next run.

In our implementation, the *sample*() method is implemented by employing an ϵ-greedy strategy, which works as the GreedyStart algorithm described in Section 4.2 with the exception that it performs a random valid task with probability ϵ. We chose the value of ϵ to be 0.1 experimentally, among candidate values 0.1, 0.3, 0.5, and 0.7. We set the value of the highest level to 2.

4.4 Local Search

Local search (LS) receives as input a valid schedule, called *starting schedule* and iteratively tries to improve it by changing the position of few transportation tasks in the schedule. Concretely, given a valid schedule S, our local search approach creates for every position p and every position $p' > p$ a schedule $S_{p,p'}$ that moves task from position p' to position p, and moves task from position p to the first position after p that makes the new schedule valid. Naturally, local search keeps the best schedule among S and all the generated schedules, and iterates.

5 Experiments

In all our instances, workstations are positioned on a straight line, and the travel times between two consecutive workstations determine the travel time between any two workstations (simply as the sum of all travel times in between).

We consider two types of travel times – *fixed* travel times and *random* travel times. The so-called *fixed* travel times refer to the scenario where every travel time between consecutive workstations is five minutes. These travel times are a good estimate of a real-world scenario from company VDL Nedcar. For random

travel times, we generate travel times uniformly at random, constrained such that the total travel time from station 0 to station $m+1$ is at most 30 minutes.

We experiment with single job-type instances and multiple job-type instances. All the experiments were run on a laptop with 8GB of RAM and with an 11th generation quad-core Intel i5 CPU.

Single Type of Jobs. We consider several settings, varying in the number of jobs, the number of stations, and the distances between the stations; see Section 6 for details. For every considered setting, we generate 5 random instances by selecting the processing time at every workstation uniformly at random from the interval $[1, 15]$, and for every tested algorithm, we average the results of the algorithm on the five instances.

We measure the performance of the ILP formulation (best objective value), GreedyStart, INMCS, and a time-window heuristic (TW) from [4]. Finally, we also tested Local Search. Each algorithm was given a maximum time of five minutes to return a solution. The TW heuristic iteratively solves ILP formulations of subproblems induced by the iterations. The total limit of five minutes was equally distributed among the subproblems; any remaining time from current iteration was further redistributed equally among subsequent iterations.

Multiple Types of Jobs. For the setting with multiple types of jobs, we have different workstation paths P_k, $k = 1, \ldots, t$, per type k, and also, for every station i, different processing times $proc(i, k)$ per job type k.

We consider settings with various total number of jobs: smaller instances, up to 20 total jobs, had two types of jobs. The proportions of the number of jobs per each job type were 25% and 75%, respectively; larger instances, with 33 and 99 total jobs, respectively, had three types of jobs. We generated instances where jobs were split equally per job type, and we also generated instances where jobs were split uniformly at random per job type, with the condition of having at least one job per type.

We further made experiments with workstation paths that do not form cycles (and thus a deadlock cannot appear), and workstation paths that do form cycles (and thus a deadlock can appear). For each such setting (with or without deadlock), we generated paths uniformly at random twice, and averaged the results over the generated paths. (For each such generated paths, we generated five random instances – the processing times – and averaged the results.)

For multiple job-types setting, we tested all algorithms but TW, as TW is a heuristic for a single job-type setting. We tested the algorithms with five and fifteen minutes time-limit.

6 Results

The results of the experiments for the setting with a single-job type are displayed in Table 1. The results of the experiments for the setting with multiple-job

types are presented in Tables 2, 3, and 4. In the column with the name of the algorithm (INMCS, ILP, etc.), we report the average difference (in percentage) of the makespan found by the respective algorithm compared to the average makespan found by GreedyStart. So, a negative difference means, on average, a better (smaller) makespan. We do not report the results for the local search algorithm, as it did not improve upon the greedy heuristic in any of the instances of the single-type setting, and the average improvement for the multiple-types setting varied between 0.1% and 2%, while often spending over two minutes of runtime, compared to the below-second runtime of the greedy algorithm.

Every row reports the average performance of the algorithms in a concrete setting (defined by the number of jobs, the types of distances, number of types, etc.). The first column describes whether the inter-station distance are uniform (label "fix 2,5") or random (label "rand"). The label "fix 2,5" further tells that we ran experiments where the uniform distance is 2, and we ran experiments where the uniform distance is 5. The label "D" describes that the generated paths form a cycle (and thus a potential deadlock"), and the label "Ø" describes that the generated paths cannot form a deadlock. The column labeled $|P_k|$ describes the length of the generated workstation paths used in the respective experiment. For example, in Table 2 the label $[3, 5][3, 3]$ means that we ran experiments with path lengths 3 and 5, and also experiments with path lengths 3 and 3, and report one average of the obtained results. In Tables 3 and 4 for settings with three types of jobs only one length is reported as both sets of paths had the same length. The column n in the same tables reports the total number of jobs (of all types) in the setting.

An asterisk in a cell means that in at least one of the instances for that row, the respective algorithm did not find a valid schedule. An asterisk without a number next to it means that the algorithm did not find a schedule for all instances.

6.1 Results for single type setting

From Table 1, we observe that GreedyStart and INMCS are able to find an optimal solution for small instances (it matches the solution found by the ILP formulation). The TW-heuristic struggles to find the optimal solution but its difference in performance is negligible for the first small instances. With larger number of stations and jobs INMCS shows its power by constantly outperforming GreedyStart. Its performance is particularly notable in settings with a high number of workstations but small number of jobs.

INMCS and also GreedyStart strongly outperforms the TW-heuristic on larger instances; TW manages to get a better performance than the greedy on instances with 6 or 10 stations and small amounts of jobs (even better than INMCS), however it performs poorly, compared to the greedy, on larger instances or it even does not manage to find a solution within the time limit.

Algorithms were quick for small instances ($<$ 10s for INMCS, $<$ 1s for TW and ILP) but ILP gets soon to 5 minutes once the amount of machines or jobs increase, similarly to TW. Runtime of INMCS also grows quickly with the size

Table 1. Comparison to the GreedyStart (difference to makespan) in a single-type setting. There are either fixed distances in between machines (fix + distance amount) or random distances (rand).

Setting	$\|P_k\|$	**n**	**INMCS%**	**TW%**	**ILP%**
fix 2,5	2	5	0.0	0.8	0.0
		20	0.0	0.0	0.0
		35	0.0	0.0	0.0
	6	5	-11.2	-9.2	-13.0
		20	-3.6	17.2*	24.0*
		35	-6.0	*	9.1*
	10	5	-15.3	3.1	-7.8
		20	-10.3	*	*
		35	-9.7	*	*
rand	2	5	0.0	0.0	0.0
		20	0.0	0.0	0.0
		35	0.0	0.0	0.0
	6	5	-8.2	-7.5	-14.5
		20	-10.8	0.5*	4.0*
		35	-8.3	-8.1*	*
	10	5	-25.6	2.5	-10.0
		20	-16.5	*	*
		35	-11.9	*	*

of the instances, but on instances with a lot of jobs and few machines it manages to get a reduced runtime (40 secs with 10 machines and 5 jobs) compared to the other algorithms. It seems from the experiments that the runtime of the ILP is mainly affected by the increase in the number of workstations, as it reaches the time limit from the first instance with a higher number of them, even with a small number of jobs. A similar phenomenon happens to the runtime of INMCS, but here the increase in computational time is much larger for higher numbers of jobs than higher number of machines. The runtime of TW is shorter compared to the ILP, however, INMCS manages to converge fast and find a better solution.

6.2 Results for multiple types setting

From Tables 2, 3, and 4 we observe that in almost every setting INMCS significantly outperformed GreedyStart. For relatively small instances the ILP found optimum solutions (while INMCS did not), but on larger instances, ILP often struggled to find a feasible solution. This shows one advantage of INMCS, it always finds a feasible solution, and often better ones than a greedy approach. Nevertheless, the runtime remains a challenge as GreedyStart always generates a solution within one second, even in settings where deadlocks can happen, while INMCS requires more than 30-40 minutes to converge on larger instances. This runtime increases even more if we used more levels.

From the results, we can see that the possibility of deadlocks did not impact the performance of INMCS, as it effectively returned a solution without them,

Table 2. Comparison to GreedyStart (difference in makespan) with two types of jobs. Maximum available computation time was 5 minutes.

Setting	$\lvert P_k \rvert$	**n**	**INMCS%**	**ILP%**
fix 2,5 $\not D$	[3,5][3,3]	5	-8.3	-18.2
		20	-6.8	0.7
fix 2,5 D	[6,5][4,3]	5	-7.7	-13.5
		20	-6.5	4.1*
rand $\not D$	[3,5][3,3]	5	-9.8	-20.0
		20	-6.5	1.8
rand D	[6,5][4,3]	5	-10.2	-14.1
		20	-6.2	3.3*

although with slight performance differences. To the contrary, the ILP often failed to yield a solution when deadlocks can appear. This difference is also reflected in the makespan of its solutions. One recurring phenomenon is that INMCS does not manage to outperform the greedy for instances with large amounts of jobs, random inter-station distances, and possibility of deadlocks. This requires more investigation in future research.

While not in any of the tables, we note that local search performed poorly, achieving marginal improvements between 0.1% and 2% over the greedy.

With respect to the runtime, ILP struggles even more than for the single-type setting, while NMCS (and also LS) on small instances takes seconds to converge. Deadlocks pose another challenge for all three algorithms, but especially for ILP which runs from 15 seconds in settings without deadlocks (on smaller instances) to 130/150 seconds in settings with deadlocks (on smaller instances). It always takes the full available 5 minutes to find a solution for 20 jobs. Local search again takes more time than NMCS (4 vs. 1 secs) for small instances but then much less with 20 jobs (17 vs. 100/200 secs). Also NMCS struggles more in settings with deadlocks (computation time almost doubles). On even larger instances (with 5 minutes of allowed runtime and 3 types of jobs) INMCS takes the full 5 minutes while LS takes 100/200 seconds with 33 total jobs and 5 minutes with 99 jobs. With 15 minutes of allowed runtime INMCS takes full 15 minutes but LS takes 400/500 seconds for 99 jobs. In both cases ILP was not able to find any solution for 99 jobs (just generating the model took too much time) and took full 5 and 15 minutes with 33 total jobs.

7 Conclusions

In this paper we studied a job-shop setting with few types of jobs, where the jobs need to be transported between the workstations by a single AGV of capacity one, and studied the optimization problem of scheduling the AGV to minimize the makespan.

We observed that an ILP formulation, while guaranteeing an optimum schedule, does not find a solution within few minutes even for relatively small instances, even when using commercial state-of-the-art ILP solvers. To solve the

Table 3. Comparison to GreedyStart (difference in makespan). Maximum allowed computation time is 5 min.

Setting	$\lvert P_k \rvert$	**n**	**INMCS%**	**ILP%**
fix 2,5 Ø	[5,5,5]	33	-7.9	310.5*
		99	-3.3	NA
fix 2,5 D	[6,6,6]	33	-5.2	*
		99	-0.2	NA
rand Ø	[5,5,5]	33	-12.2	*
		99	-6.0	NA
rand D	[6,6,6]	33	-7.2	*
		99	0.9	NA

Table 4. Comparison to GreedyStart (difference in makespan). Maximum allowed computation time is 15 min.

Setting	$\lvert P_k \rvert$	**n**	**INMCS%**	**ILP%**
fix 2,5 Ø	[5,5,5]	33	-8.7	36.5*
		99	-3.6	NA
fix 2,5 D	[6,6,6]	33	-5.5	298.9*
		99	-1.1	NA
rand Ø	[5,5,5]	33	-13.2	3.5*
		99	-6.5	NA
rand D	[6,6,6]	33	-8.2	*
		99	0.3	NA

problem in practice, heuristics are used. We designed a heuristic based on the Iterative Nested Monte-Carlo Search technique to address the practical problem. INMCS iteratively performs multiple runs of Nested Monte-Carlo Search to find solutions. We compared INMCS with greedy algorithms, and also local search applied as a post-processing to solutions found be the greedy algorithms.

Our experimental evaluation show that INMCS consistently outperforms all considered algorithms within the given run-time bounds of 5 or 15 minutes, especially when the input instances get larger. Noteworthy, for settings with a single job-type, INMCS also outperforms the time-window ILP-heuristic of [4].

For future research we would like to compare our INMCS implementation with other single-agent Monte-Carlo Search approaches such as SP-MCTS [24], NRPA [22] or NMCTS [2].

Acknowledgments. We thank VDL Nedcar for discussing real-world scenarios. This work 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.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Baier, H.: Monte-Carlo tree search enhancements for one-player and two-player domains. Ph.D. thesis, Maastricht University (2015). https://doi.org/10.26481/dis.20151124hb
2. Baier, H., Winands, M.H.M.: Nested Monte-Carlo Tree Search for online planning in large MDPs. In: Proceedings of the 20th European Conference on Artificial Intelligence. pp. 109–114. IOS Press, The Netherlands (2012). https://doi.org/10.3233/978-1-61499-098-7-109
3. Bilge, Ü., Ulusoy, G.: A time window approach to simultaneous scheduling of machines and material handling system in an FMS. Operations Research **43**(6), 1058–1070 (1995)
4. Boom, E., Mihalák, M., Thuijsman, F., Winands, M.H.M.: Scheduling single AGV in blocking flow-shop with identical jobs. In: ICORES. pp. 325–332 (2024)
5. Carlier, J., Haouari, M., Kharbeche, M., Moukrim, A.: An optimization-based heuristic for the robotic cell problem. European Journal of Operational Research **202**(3), 636–645 (2010)
6. Caumond, A., Lacomme, P., Moukrim, A., Tchernev, N.: An MILP for scheduling problems in an FMS with one vehicle. European Journal of Operational Research **199**(3), 706–722 (2009)
7. Cazenave, T.: Nested monte-carlo search. In: Twenty-First International Joint Conference on Artificial Intelligence (2009)
8. Cazenave, T., Balbo, F., Pinson, S.: Using a monte-carlo approach for bus regulation. In: 2009 12th International IEEE Conference on Intelligent Transportation Systems. pp. 1–6 (2009). https://doi.org/10.1109/ITSC.2009.5309838
9. Cazenave, T., Jouandeau, N.: Parallel nested monte-carlo search. In: Proceedings of IEEE International Symposium on Parallel Distributed Processing (IPDPS). pp. 1–6 (2009). https://doi.org/10.1109/IPDPS.2009.5161122
10. Deroussi, L., Gourgand, M., Tchernev, N.: A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles. International Journal of Production Research **46**(8), 2143–2164 (2008)
11. Edelkamp, S., Gath, M., Greulich, C., Humann, M., Herzog, O., Lawo, M.: Monte-carlo tree search for logistics. In: Clausen, U., Friedrich, H., Thaller, C., Geiger, C. (eds.) Commercial Transport. pp. 427–440. Springer International Publishing, Cham (2016)
12. Erol, R., Sahin, C., Baykasoglu, A., Kaplanoglu, V.: A multi-agent based approach to dynamic scheduling of machines and automated guided vehicles in manufacturing systems. Applied soft computing **12**(6), 1720–1732 (2012)
13. Hall, N.G., Sriskandarajah, C.: A survey of machine scheduling problems with blocking and no-wait in process. Operations Research **44**(3), 510–525 (1996). https://doi.org/10.1287/opre.44.3.510
14. Heger, J., Voss, T.: Optimal scheduling for automated guided vehicles (AGV) in blocking job-shops. In: Advances in Production Management Systems. The Path to Intelligent, Collaborative and Sustainable Manufacturing: IFIP WG 5.7 International Conference, APMS 2017, Hamburg, Germany, September 3-7, 2017, Proceedings, Part I. pp. 151–158. Springer (2017)
15. Hosseini, A., Otto, A., Pesch, E.: Scheduling in manufacturing with transportation: Classification and solution techniques. European Journal of Operational Research (2023)

16. Kinny, D.: A new approach to the snake-in-the-box problem. In: Raedt, L.D., Bessiere, C., Dubois, D., Doherty, P., Frasconi, P., Heintz, F., Lucas, P.J.F. (eds.) ECAI 2012 - 20th European Conference on Artificial Intelligence. Frontiers in Artificial Intelligence and Applications, vol. 242, pp. 462–467. IOS Press (2012). https://doi.org/10.3233/978-1-61499-098-7-462
17. Mayer, S., Höhme, N., Gankin, D., Endisch, C.: Adaptive production control in a modular assembly system — towards an agent-based approach. In: Proceedings of the 17th IEEE International Conference on Industrial Informatics (INDIN). vol. 1, pp. 45–52 (2019). https://doi.org/10.1109/INDIN41052.2019.8972152
18. Pinedo, M.L.: Planning and Scheduling in Manufacturing and Services. Springer New York, 2nd edn. (2009). https://doi.org/10.1007/978-1-4419-0910-7
19. Poppenborg, J., Knust, S., Hertzberg, J.: Online scheduling of flexible job-shops with blocking and transportation. European Journal of Industrial Engineering **6**(4), 497–518 (2012)
20. Qiu, L., Hsu, W.J., Huang, S.Y., Wang, H.: Scheduling and routing algorithms for AGVs: a survey. International Journal of Production Research **40**(3), 745–760 (2002)
21. Rimmel, A., Teytaud, F., Cazenave, T.: Optimization of the nested monte-carlo algorithm on the traveling salesman problem with time windows. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A.G.B., Urquhart, N., Uyar, A.Ş. (eds.) Applications of Evolutionary Computation. pp. 501–510. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
22. Rosin, C.D.: Nested rollout policy adaptation for monte carlo tree search. In: Walsh, T. (ed.) IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence. pp. 649–654. IJCAI/AAAI (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-115
23. Roucairol, M., Cazenave, T.: Solving the hydrophobic-polar model with nested monte carlo search. In: Nguyen, N.T., Botzheim, J., Gulyás, L., Núñez, M., Treur, J., Vossen, G., Kozierkiewicz, A. (eds.) Advances in Computational Collective Intelligence - 15th International Conference, ICCCI 2023, Budapest, Hungary, September 27-29, 2023, Proceedings. Communications in Computer and Information Science, vol. 1864, pp. 619–631. Springer (2023). https://doi.org/10.1007/978-3-031-41774-0_49
24. Schadd, M.P.D., Winands, M.H.M., Tak, M.J.W., Uiterwijk, J.W.H.M.: Single-Player Monte-Carlo Tree Search for SameGame. Knowledge-Based Systems **34**, 3–11 (2012). https://doi.org/10.1016/j.knosys.2011.08.008
25. Vis, I.F.: Survey of research in the design and control of automated guided vehicle systems. European journal of operational research **170**(3), 677–709 (2006)
26. Wimmenauer, F., Mihalák, M., Winands, M.H.M.: Monte-carlo tree-search for leveraging performance of blackbox job-shop scheduling heuristics (2022), https://arxiv.org/abs/2212.07543
27. Zheng, Y., Xiao, Y., Seo, Y.: A tabu search algorithm for simultaneous machine/AGV scheduling problem. International Journal of Production Research **52**(19), 5748–5763 (2014)