Scheduling of Mobile Robots Using Constraint Programming

Stanislav Murín(✉) and Hana Rudová

Faculty of Informatics, Masaryk University, Brno, Czech Republic
murin.stanislav@mail.muni.cz, hanka@fi.muni.cz

Abstract. Mobile robots in flexible manufacturing systems can transport components for jobs between machines as well as process jobs on selected machines. While the job shop problem with transportation resources allows encapsulating of transportation, this work concentrates on the extended version of the problem, including the processing by mobile robots. We propose a novel constraint programming model for this problem where the crucial part of the model lies in a proper inclusion of the transportation. We have implemented it in the Optimization Programming Language using the CP Optimizer, and compare it with the existing mixed integer programming solver. While both approaches are capable of solving the problem optimally, a new constraint programming approach works more efficiently, and it can compute solutions in more than an order of magnitude faster. Given that, the results of more realistic data instances are delivered in real-time, which is very important in a smart factory.

Keywords: Scheduling · Constraint programming · Mobile robot · Flexible manufacturing system · Transportation · IBM ILOG CPLEX Optimization Studio

1 Introduction

The concept of the smart factory was defined by the Industry 4.0 [18] project recently. There is an emphasis on automation, data exchange, and flexible manufacturing technologies. A flexible manufacturing system consists of the work machines such as automated CNC machines connected by a material handling system and the central control computer. The material handling system can be realized by conveyors or automatic guided vehicles (AGV). AGVs are used to transport materials between machines [8].

Traditional job shop scheduling problems have been extended to work with the AGVs for transportation of material whenever the job changes from one machine to another. This class of problems is called the job shop scheduling with transportation (JSPT) as discussed by Nouri *et al.* [21] who reviewed various approaches applied to this problem. The classical work of Bilge and Ulusoy [8] formulated a nonlinear mixed integer programming model which was intractable

T. Schiex and S. de Givry (Eds.): CP 2019, LNCS 11802, pp. 456–471, 2019.
https://doi.org/10.1007/978-3-030-30048-7_27

due to the size and nonlinearity. To handle that, they applied an iterative heuristic approach to solve the problem. Later on, many metaheuristic and heuristic approaches have been studied to solve this problem [21].

Recent approach [11] extended the JSPT to allow for the inclusion of mobile robots who can perform various value-added tasks on machines as well as transportation of material between machines. They proposed an exact approach using linear mixed integer programming and solved the corresponding extension of benchmark problems from [8] optimally. These problems have varying difficulty, some of them can be solved by the mixed integer programming (MIP) approach within few seconds while computations of others took more than ten hours. To deliver solutions in a short time (within several seconds), the hybrid heuristic based on genetic algorithms was also proposed and implemented in [11]. Very latest work studied solution to this problem using adaptive large neighborhood search [12], and it is aimed to obtain solutions in real-time. Our paper also concentrates on this problem while proposing a different exact approach represented by constraint programming (CP).

In our approach, we use IBM ILOG CP Optimizer and its Optimization Programming Language [2,17]. Our problem is a combination of scheduling [6,24] and vehicle routing [16,25]. To solve the problem, we need to assign mobile robots to each transportation and processing where the robot is needed, as well as assign starting time to each transportation and processing. The transportation includes pickup of the job components and its delivery to the consequent machine where the job is processed.

Since we are not aware of any CP approach to our problem, we explored similarities with other close problems. There are relations to pickup and delivery problems [22,23], as well as dial-a-ride problems [10] where their vehicles correspond to mobile robots and pickup and delivery requests between origins and destinations can be seen as our transportations between the origin and destination machine. Berbeglia *et al.* proposed the first exact algorithm to check the feasibility of dial-a-ride problems using CP [7]. Liu *et al.* [19] approached the senior transportation problem using CP, MIP, logic-based Benders decompositions as well as constructive heuristic and found CP being the best approach. A similar problem of the patient transportation [9] was recently solved by CP efficiently. All these CP approaches [7,9,19] consider activities for pickup and delivery separately, corresponding to the fact that they may be interleaved among each other. However, this is in contrast to our approach, where each pickup is followed by the corresponding delivery. We show that the model with separate variables for pickup and delivery is not effective enough, and it must be replaced by a model where pickup and delivery activities are replaced by one transportation activity.

Other related problems are represented by scheduling with setup times or costs [4] where sequence-dependent setup times may correspond to our transportations between machines processed for consequent operations. Taking into account CP, various approaches solved the job shop scheduling problem with sequence-dependent setup times integrating setups with the objective function and search [5,14,15]. Recently, the propagation procedure, including transition

times into the classic filtering algorithm for unary resources [13] was proposed to solve this problem. This is also the approach taken by CP Optimizer, where non-overlapping constraint with transition times is available [17]. While this is an interesting concept, it cannot be directly applied in our case since our transportation activities must be related to two different locations. To handle that, we propose a more complex modeling approach with both non-overlapping constraints as well as transportation activities.

Let us summarize the contributions of our work.

1. We introduce the first CP model for scheduling with mobile robots.
2. A novel approach to handle complex transportations using non-overlapping constraints is proposed.
3. A new CP approach is in more than an order of magnitude faster in computing of optimal solution than the earlier MIP approach [11].
4. For smart factories, it is important that real-time computation (within a second) is achieved for more realistic data instances.

The next section describes our scheduling problem with mobile robots. Section 3 presents the detail CP model with transportation activities and the alternative approach with the pickup and delivery activities. The section concludes by the discussion of the search options. Section 4 introduces data instances, explores different versions of our model and search, and compares our approach with the MIP solver taken from [11]. The final section summarizes the results and presents some ideas for future work.

2 Problem Description

We have `m` machines where `n` jobs are processed. Each job is composed of the set of operations each processed on a different machine. Typically, jobs are not processed on all machines, so the number of operations per job differs. For each job, there is a specific order of the operations, *i.e.*, we know in advance the order of processing of operations on machines for each job. Operations of one job cannot overlap, and only one operation (and job) can be processed on one machine at any time.

Our problem is related to the combination of the job shop scheduling problem and vehicle routing, which is called the job shop scheduling with transportation resources [21]. It means that there are vehicles, *e.g.*, automated guided vehicles (AGV), which are used for transportation of the components between every two consequent operations of one job [8]. The AGVs are identical, and the travel times between machines are specified (they include loading and unloading times) by the distance matrix. There are sufficient input and output buffer space at each machine where components of the job or robots can wait. There is also a special loading/unloading "dummy" machine (L/U station) where all AGVs start, and all materials for jobs are available. So, the first transportation for each job starts at the L/U station and the last one ends there.

In this paper, we study a recently proposed problem where some operations need processing by mobile robots [11]. Mobile robots are identical and perform both transportations as well as the processing of selected operations. Each robot can perform at most one activity (processing or transportation) at a time.

Example 1. A sample problem with the schedule is shown in Fig. 1. There are two robots, three jobs, three machines and the L/U station where the processing of all jobs starts. Routes for both robots are also depicted with the numbered transportations and processings.

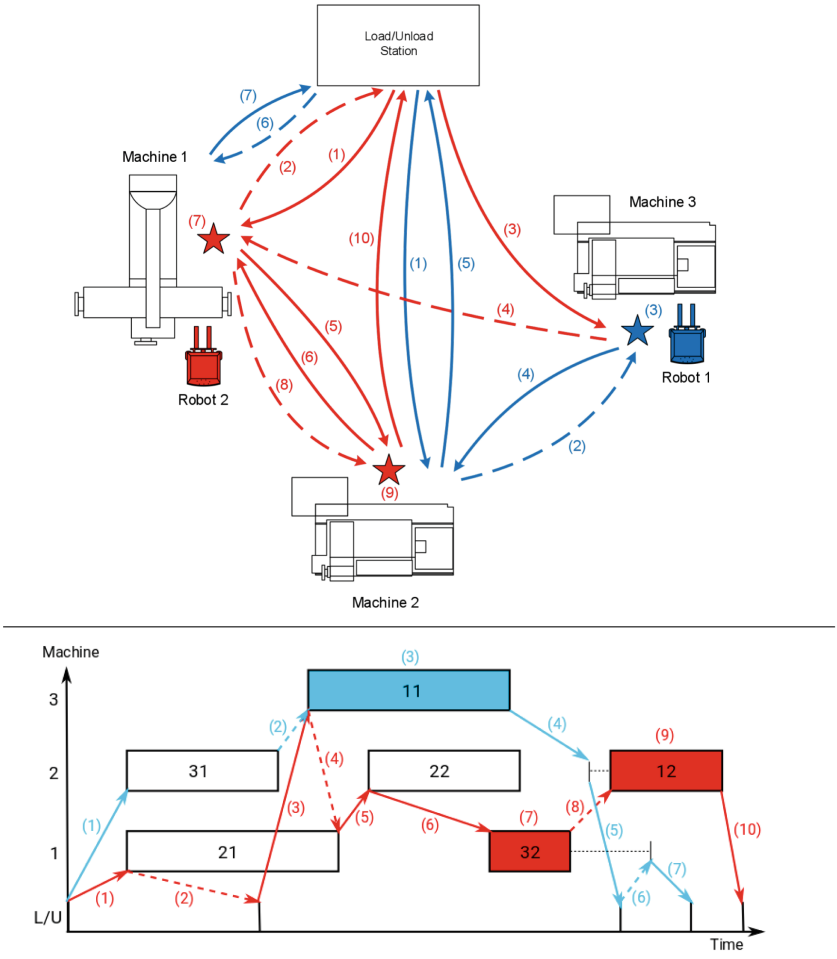

Fig. 1. Layout of FMS with mobile robots, and the schedule (based on [11]).

For instance, we can see that the third job is started by transportation (1) using the blue robot. Consequently, operation 31 is processed on the machine 2.

After some delay, the red robot transports components of the third job to the machine 1 by (6) which is followed by the processing of the operation 32 using the same red robot (7). Finally, job components are transported by the blue robot (7) to the L/U station, again after some delay.

In the robot routes, we can see the processing by robots (colored boxes) and the transportation of the components (so-called loaded trips; solid lines). There are also dashed lines showing transportations of robots without any materials (so-called empty trips) which are necessary to move the robot to the machine where it is needed, *e.g.*, the transportation (4) between the machines 3 and 1.

3 Model and Search

We will describe particular components of the model, starting from the base variables and constraints followed by more sophisticated concepts. Also, we would like to relate similar ideas together to make an understanding of the model easier. The final part of this section will discuss explored possibilities of the search method.

3.1 Interval Variables for Processing and Transportation

Interval variables are used to encapsulate both processing of operations as well as transportation of job components between machines. These variables represent processing and transportation activities.

A pair `<j,o>` identifies an operation processed as an `o`-th operation of the job `j` based on predefined ordering. We have a `Set` of such tuples to represent all operations. Note that operations of each job are organized in `Set` such that ordering of operations represents the processing order of operations on machines.

```
tuple pair {int j; int o;}
{pair} Set;
```

Using the set of tuples allows to handle variable number of operations per job easily.

For each operation `jo=<j,o>`, the interval variable `processing[jo]` is available for its processing. The interval variable `transport[jo]` represents transportation of the job components from the machine where the previous operation is processed to the machine where the operation `jo` is processed (we will write: transportation from previous operation to the current operation `jo`). The transportation to the first operation starts at the L/U station (which is further index by `0`).

```
dvar interval processing[jo in Set] size processingSize[jo];
dvar interval transport[jo in Set] size transportSize[jo];
```

The size of the activity corresponds either to the processing time of the operation `jo` given by `processingSize[jo]` or to the travel time `transportSize[jo]` between the machines where the operation `jo` and its previous operation are processed.

Example 2. The operation 21 in Fig. 1 is represented by the processing activity `processing[<2,1>]`. The transportation (5) from the operation 21 to the operation 22 is represented by the transportation activity `transportation[<2,2>]`.

3.2 Temporal Constraints

The activities for processing and transportation of one job are related by the precedence constraints. The transportation activity `transport[jo]` from the previous operation to the operation `jo` must precede the processing activity `processing[jo]`.

```
forall (jo in Set)
   endBeforeStart(transport[jo], processing[jo]);
```

At the same time, the transportation activity `transport[jo]` must be preceeded by the processing activity from the previous operation. The previous operation of `jo` in the `Set` can be obtained using `prev(Set,jo)` function.

```
forall (jo in Set: jo.o != 1)
   endBeforeStart(processing[prev(Set,jo)], transport[jo]);
```

Note that the first transportation from L/U station to the next machine (`jo.o=1`) is not proceeded by any processing.

3.3 Non-overlapping of Operations on Machines

The operations processed in the same machine cannot overlap. This can be achieved by the inclusion of the sequence variable

```
dvar sequence machinePlan[i in 1..m] in
     all (jo in Set: operation[jo].machine == i) processing[jo];
```

for each of the `m` machines. The structure `operation[jo].machine` stores the machine for each operation `jo`. Consequently, the `noOverlap` constraint is posted for all machines.

```
forall(i in 1..m) noOverlap(machinePlan[i]);
```

3.4 Alternative Interval Variables

There are interval variables `rbtTransport[r][jo]` and `rbtProcessing[r][jo]` which ensure selection of one proper robot `r` for all transportation activities and for the processing activities which requires a robot for processing.

```
dvar interval rbtTransport [r in 1..rb][jo in Set] optional
     size transportSize[jo];
dvar interval rbtProcessing [r in 1..rb][jo in Set] optional
     size processingSize[jo];
```

The keyword `optional` corresponds to the fact that (at most) one robot activity will be selected by the `alternative` constraints to choose among the `rb` robots.

The following `alternative` constraint means that one robot transportation activity `rbtTransport[r][jo]` is selected (is present) for each transportation activity `transport[jo]`.

```
forall (jo in Set)
  alternative(transport[jo], all (r in 1..rb) rbtTransport[r][jo]);
```

The other `alternative` constraint is posted for all processing activities where the robot is working on the operation together with the machine (the data structure `operation[jo].robotRequired` stores information about needed robots). Again one robot processing activity `rbtProcessing[r][jo]` will be present for each processing activity `processing[jo]` requiring a robot.

```
forall (jo in Set: operation[jo].robotRequired == 1)
  alternative(processing[jo], all (r in 1..rb) rbtProcessing[r][jo]);
```

In addition, all robot processing activities `rbtProcessing[r][jo]` are set not be present by the `presenceOf` constraint when the robot is not needed.

3.5 Non-overlapping of Activities for Robots

Finally, we have all important elements of the model to propose how to handle non-overlapping of activities for each robot. Firstly, we define the sequence variable `rbtRoute[r]` for each robot `r` which includes all activities for the robot `r`. There are robot processing activities `rbtProcessing[r][jo]` for all operations `jo` where a robot is required. In addition, there are the robot transportation activities `rbtTransport[r][jo]`. Of course, the present activities are involved in the final sequence only.

```
dvar sequence rbtRoute[r in 1..rb]
    in append(all (jo in Set) rbtProcessing[r][jo],
              all (jo in Set) rbtTransport[r][jo])
    types append(all (jo in Set) processingType[jo],
                 all (jo in Set) transportType[jo]);
```

As you can see, the sequence variables have defined their type which allows to handle the *empty* transportations of the robots between machines when their next processing or the origin of the next transportation is scheduled on a different machine than the robot is placed.

Example 3. The red robot in Fig. 1 performs the transportation from the L/U station to the operation 11 (to the machine 3) by the robot transportation activity `rbtTransport[red][<1,1>]` denoted by (3). Consequently, the red robot needs to go the machine 1 by the transportation (4). At the machine 1, the red robot needs to pick up the job components for the job 2 and transport them to the machine 2 by (5) and `rbtTransport[red][<2,2>]`. The transportation (4) represents the empty trip (and it is not represented by any activity).

For each empty trip, *e.g.* (8), we need to make sure that there is enough time to perform transportation from the origin activity (`rbtProcessing[red][<3,2>]` at machine 1) to the destination activity (`rbtProcessing[red][<1,2>]` at machine 2). This is completed by the `noOverlap` constraint with the `distanceMatrix`.

```
forall(r in 1..rb) noOverlap(rbtRoute[r], distanceMatrix);
```

This constraint ensures non-overlapping of all activities for each robot `r` as well as handling of travel times for their empty trips. The distance matrix is a set of tuples encoding the distances (travel times) between every two types.

```
tuple Triplet { int type1; int type2; int distance; }
{Triplet} distanceMatrix;
```

In vehicle routing problems [16,17], each activity represents processing at some location. Locations have defined their distances by the distance matrix. When we assign the corresponding location as a type to each activity, the `noOverlap` constraint enforces the minimal travel time between every two consequent activities from the different locations.

As mentioned in Sect. 1, the concept in vehicle routing problems cannot be directly implemented in our case, because our transportation activities are related to two different machines/locations. The transportation activity starting at the machine `i1` and ending at the machine `i2` cannot simply be related to one of the machines. Instead, we propose to have types associated with the two machines `i1,i2`. Next, the distance matrix needs to define distances for each quadruple `i1,i2` and `i3,i4` corresponding to the travel time between `i2` and `i3`. Note that the distance between the type `i1,i2` and `i2,i3` is zero. Of course, the processing activities still resides on one machine which results in the type `i,i` for the machine `i`. It means that the space complexity for the distance matrix corresponds to $\mathcal{O}(\mathtt{m}^4)$.

Example 4. The robot transportation activity `rbtTransport[red][<1,1>]` (3) has the type `03` because it corresponds to the transportation from the L/U station to the machine 3. The activity `rbtTransport[red][<2,2>]` (5) has the type `12` corresponding to the transportation from the machine 1 to the machine 2. The empty trip (4) has secured the minimal travel time given by the distance between the types `03` and `12` which is given by the travel time between the machines 3 and 1.

The robot processing activity `rbtProcessing[red][<3,2>]` for the operation 32 by (7) has the type `11` as it is processed on the machine 1.

3.6 Different Approach with Pickup and Delivery Activities

The initial solution approach for our problem was based on interval variables for both pickup and delivery, *i.e.*, there are interval variables `pickup` and `delivery` as well as `rbtPickup` and `rbtDelivery`. Both pickup and delivery variables are time points, *i.e.*, their size equals to zero.

Example 5. The transportation (5) in Fig. 1 from the operation 21 to the operation 22 is represented by the pickup activity `pickup[<2,2>]` and the delivery activity `delivery[<2,2>]`.

There are new constraints related to the fact that the same robot must complete pickup and delivery for one transportation. It means that their presence variables must have the same value. Also, the pickup robot activity directly precedes the delivery robot activity for the corresponding transportation. This is achieved by the `prev` constraint.

```
forall (r in 1..rb, jo in Set) {
   presenceOf(rbtPickup[r][jo]) == presenceOf(rbtDelivery[r][jo]);
   prev(rbtRoute[r],rbtPickup[r][jo],rbtDelivery[r][jo]); }
```

For one transportation, the ending time of the pickup activity must be separated from the starting time of the delivery activity just by the travel time between machines. Travel times are stored in the structure `travelTimes[i1][i2]` for each two machines `i1`, `i2` including the L/U station. Note that this data structure is also used to construct the distance matrix.

```
forall (jo in Set: jo.o != 1)
   endAtStart(pickup[jo], delivery[jo],
     travelTimes[operation[prev(Set,jo)].machine][operation[jo].machine]);
```

Since we need to include traveling from the L/U station to the first machine for each job, we also have the following constraints.

```
forall (jo in Set: jo.o == 1)
   endAtStart(pickup[jo], delivery[jo],
     travelTimes[0][operation[jo].machine]);
```

Finally, there is a slightly different implementation of the precedence constraints from Sect. 3.2. In the first constraint, we consider precedence between delivery (instead of transportation) and processing activities. The second constraint implements precedence between processing and pickup (again replacing transportation) activities.

3.7 Objective Function

The objective of the problem is to minimize the completion time of all activities, *i.e.*, the makespan. In our case, we minimize the maximal completion time of the last processing activity of each job (the data structure `nbOperations` stores the number of operations per each job). It means that we do not consider the last transportation to the L/U station which is in correspondence with earlier works [8,11].

```
minimize max(jo in Set: jo.o == nbOperations[jo.j])
              endOf(processing[jo]);
```

3.8 Search

CP Optimizer [17] employs a combination of Large Neighbourhood Search and Failure-directed Search [26] by default. Other options are introduced by multi-point search and depth-first search [1,17]. In the experimental part, we will explore the differences among these search algorithms. Based on our analysis, the default search is significantly better than other options (see Sect. 4.2).

In all cases, the search process is performed to find an optimal solution as well as prove the optimality of the found solution. Certainly, proving the optimality takes much longer time than the computation of the first optimal solution.

Another possibility of how to tune the search in CP Optimizer can be introduced by consideration of search phases together with grouping variables and their specific ordering. However, this does not make much sense for our problem. We have also explored various built-in variable and value ordering, but none of them appeared to have a positive effect.

To conclude, we use the default search setting of CP Optimizer.

4 Experiments

In this section, we describe the experimental evaluation. Our approach is implemented using the academic distribution of IBM ILOG CPLEX Optimization Studio 12.8 in OPL[1]. We compare it with the MIP implementation in OPL [11] using the same version and setting. We use random seeds to diversify the solution approach, which plays a significant role in statistical comparison in both CP and CPLEX engines [3,17]. To allow for that, 30 runs are completed for each experiment. The experiments are run on a computer with Intel Xeon Gold 6130, 16 GB RAM using a single thread.

4.1 Data Instances

We use data instances from [11] available from the website[2]. These data instances extend JSPT data from [8] by introducing the processing by robots on some operations.

There are 82 data instances with 4 machine layouts and different t/p ratios (travel time/processing time). There are 4 machines (plus the L/U station) in each layout, 5–8 jobs, 13–21 operations, and 2 robots. The first group of 40 data instances has relatively high t/p ratio ($t/p > 0.25$), while the other 42 data instances have it relatively low ($t/p < 0.25$). The names given to the instances start with prefix "EX" followed by the number of the job set and the layout. The names of the instances in the second group include the additional 0, 1 digit implying that the processing time is doubled or tripled, respectively. In the second group, travel times are halves (*e.g.*, "EX541" corresponds to the job

[1] The source code, including data instances, is available from https://github.com/StanislavMurin/Scheduling-of-Mobile-Robots-using-Constraint-Programming.

[2] https://sites.google.com/site/schedulingmobilerobots/.

Table 1. Performance of various setting in the CP approach (in seconds).

	EX110		EX210		EX11		EX22		EX63	
	Time	S.D.	Time	S.D.	Time	S.D.	Time	S.D.	Time	S.D.
Default	0.010	0.003	0.028	0.015	0.745	0.142	0.447	0.133	0.564	0.077
DFS	0.006	0.005	2.443	0.939	0.473	0.018	2.567	1.235	16.763	0.844
MultiPoint	0.018	0.007	0.136	0.083	–	–	–	–	–	–

set 5 with tripled processing time and the layout 4 with half travel time). For the second group, it results in a more realistic data set, since the robot takes a long time for processing in comparison with its travel time [20].

4.2 Setting

In this section, we compare different versions of our model as well as built-in search methods.

Model Setting. In Table 1, the experiments exploring efficiency of our model using the default CP Optimizer setting are presented in the first line denoted *Default*. We have also tried different levels of propagation as available within CP Optimizer, but none of them has a positive effect.

The alternative model replaces transportation activities by the pickup and delivery activities as described in Sect. 3.6. Performance of this model was very weak, resulting in 96.794 ± 5.086 s even for the easiest EX110 problem. This confirmed our expectation after some trial experiments that this model cannot be used at all.

Search Setting. In Table 1, the results for the default search (*Default*), the depth-first search (*DFS*), and the multi-point search (*Multi-point*) are given.

While the best performance of the default search is not clear based on the instance EX110 and EX11 where the *DFS* is almost two times faster, it becomes decided on other instances. The depth-first search was significantly worse for other problems. The speed-up of the default search was 84.25, 5.74, and 29.72 for EX210, EX22, and EX63, respectively. It confirmed our preliminary experiments where the depth-first search was very slow on many problems. The multi-point search was not even able to find a solution within 12 h for three problems, even though there are still rather easy data instances.

Given the initial experiments, we can conclude that the default setting of CP Optimizer is the best, and it is further used to perform experiments on all data instances.

4.3 Results and Comparison with MIP

We compare the results of our CP solver with the results of the MIP solver from [11]. Table 2 includes the computational times (*Time*) with the standard deviations (*S.D.*) for both approaches together with the speed-up (*Speed.*) of the CP approach. The left columns show results for 40 data instances with $t/p > 0.25$, and the right columns for 42 instances with $t/p < 0.25$. Different levels of the blue color are used to emphasize differences in computational times. Darker colors show shorter computational times while lighter colors demonstrate higher times. Similarly, red colors show differences in speed-up. Here darker colors demonstrate better results of our CP solver.

The results for the first set of 40 data instances were mostly computed within 10 s (15 instances within a second, other 17 instances in less than 10 s). The three most demanding instances EX71, EX74, and EX104 needed 23.5, 14.3, and 4.8 min, respectively. The remaining 5 instances required the computational time between 10−100 s (23.85 s on average). For the MIP solver, the instances EX71, EX74, and EX104 are very hard, and the results were computed within 12 h in 3 out of 90 cases (values in the column for speed-up specify the number of solved instances). Other instances were also much harder for MIP than for CP. Only 5 instances suffice with the computational time less than ten seconds. For 12 other instances, solutions were computed within a hundred seconds, and 11 more instances needed 100−1,000 s. The last 9 instances required more than a thousand seconds. We cannot compare speed-up for EX71, EX74, and EX104 instances which were hard for both solvers, because the MIP solver has not mostly computed solutions. The CP solver was 1,000, 100−1,000, and less than 100 faster for 3, 18, and 16 problems, respectively. The smallest speed-up was 8.8, and it was 38.1 on average for 16 problems with the speed-up smaller than a hundred.

We can observe the correlation between the computational time and the job set or the layout complexity. The layouts 1 and 4 appear to be generally more difficult for both solvers, having average computational times of 6.5 and 1,975.8 s for the layout 1, and 8.1 and 963.0 s for the layout 4 for CP and MIP, respectively. Layout 2 and 3 are generally easier with the average computational times of 1.0 and 217.6 s for the layout 2, and 1.4 and 261.2 s for the layout 3 for CP and MIP, respectively. Please note, that job sets 7 and 10 were excluded from average computation over layouts, because of unfinished MIP tests in layout 1 and 4.

The results for the second set of instances is in the right part of Table 2. We can see that computational times of the CP approach are always smaller than one second, which is very important because these problems represent more realistic data sets as discussed before. Computational times rather rely on the given job set which is given by smaller importance of travel times in this set of problems, since the travel times were halved and processing times were doubled or tripled. The problems from the job sets 9, and 10 need the highest computational time, mostly higher than 0.5 s. The job sets 4 and 6 with the exclusion of EX420 require between 0.1 and 0.6 s. 27 remaining instances suffice with solution time smaller than 0.05 s on average. The MIP solver needs more than 9 s for the job sets 8–10

Table 2. Performance of CP and MIP approach (computational times in seconds).

	$t/p > 0.25$						$t/p < 0.25$					
No.	CP		MIP			No.	CP		MIP			
	Time	S.D.	Time	S.D.	Speed.		Time	S.D.	Time	S.D.	Speed.	
EX11	0.74	0.14	21.09	14.00	28.3	EX110	0.010	0.003	0.418	0.097	40.4	
EX21	6.40	0.98	813.09	307.91	127.0	EX210	0.028	0.015	5.052	1.084	178.3	
EX31	1.45	0.35	151.71	54.37	104.8	EX310	0.011	0.004	2.401	1.293	218.3	
EX41	29.64	6.11	11,062.71	4,671.52	373.3	EX410	0.381	0.093	10.043	2.625	26.4	
EX51	2.09	0.35	103.41	28.65	49.4	EX510	0.009	0.003	0.451	0.113	52.0	
EX61	7.81	1.92	2,144.12	819.86	274.5	EX610	0.295	0.092	9.954	12.807	33.7	
EX71	1,410.05	1,098.22	–	–	0/30	EX710	0.014	0.005	2.318	1.471	161.7	
EX81	0.03	0.05	39.11	28.99	1,448.5	EX810	0.011	0.003	43.181	30.684	3,810.1	
EX91	4.51	0.70	1,471.48	1,134.73	326.0	EX910	0.496	0.131	10.945	2.517	22.1	
EX101	40.30	4.86	11,034.87	3,868.56	273.8	EX1010	0.484	0.072	52.802	69.282	109.1	
EX12	0.20	0.07	2.15	0.76	10.5	EX120	0.008	0.005	0.333	0.071	41.6	
EX22	0.45	0.13	12.47	5.74	27.9	EX220	0.045	0.026	3.975	0.905	87.7	
EX32	0.35	0.04	5.52	0.97	15.6	EX320	0.009	0.005	1.217	0.592	130.4	
EX42	4.93	0.64	1,439.60	614.47	291.8	EX420	0.023	0.004	7.103	1.787	313.4	
EX52	0.38	0.09	3.57	1.11	9.4	EX520	0.010	0.005	0.449	0.147	44.9	
EX62	0.39	0.06	38.15	28.02	98.7	EX620	0.213	0.112	7.555	2.816	35.5	
EX72	1.65	0.43	269.45	145.27	162.9	EX720	0.012	0.005	4.875	1.840	417.9	
EX82	0.01	0.01	72.78	63.20	7,529.4	EX820	0.012	0.006	27.567	21.253	2,362.9	
EX92	0.95	0.22	166.36	79.32	174.7	EX920	0.575	0.157	9.306	3.044	16.2	
EX102	2.11	0.87	555.15	184.64	262.9	EX1020	0.443	0.261	42.312	29.385	95.5	
EX13	0.45	0.14	3.98	1.11	8.8	EX130	0.008	0.004	0.361	0.058	43.3	
EX23	1.17	0.50	65.59	37.82	56.0	EX230	0.013	0.006	4.091	0.805	322.9	
EX33	0.34	0.08	5.17	1.11	15.3	EX330	0.010	0.003	1.171	0.475	113.4	
EX43	6.32	0.85	1,481.41	673.00	234.5	EX430	0.157	0.075	4.488	1.168	28.5	
EX53	0.63	0.11	14.61	7.14	23.3	EX530	0.009	0.006	0.367	0.098	42.4	
EX63	0.56	0.08	86.11	52.28	152.6	EX630	0.338	0.171	6.646	1.705	19.7	
EX73	5.18	1.25	661.64	341.92	127.8	EX730	0.009	0.005	4.340	1.602	500.8	
EX83	0.01	0.00	76.68	83.95	8,216.0	EX830	0.012	0.005	26.695	18.926	2,288.1	
EX93	1.69	0.29	356.05	144.02	210.2	EX930	0.482	0.145	9.748	3.511	20.2	
EX103	3.53	0.45	948.83	366.85	269.1	EX1030	0.389	0.130	41.732	22.047	107.4	
EX14	1.23	0.19	22.03	15.27	17.9	EX140	0.013	0.005	0.440	0.146	33.0	
EX24	8.86	1.54	610.69	241.69	68.9	EX241	0.013	0.005	3.407	0.974	269.0	
EX34	3.80	0.64	205.19	87.97	54.0	EX340	0.012	0.004	2.282	1.213	185.0	
						EX341	0.015	0.005	1.637	0.637	109.1	
EX44	18.11	3.03	2,316.20	994.28	127.9	EX441	0.274	0.045	7.826	1.841	28.6	
EX54	1.77	0.29	84.69	29.51	47.9	EX541	0.008	0.004	0.464	0.100	55.6	
EX64	21.06	3.46	1,651.60	772.60	78.4	EX640	0.508	0.502	8.311	2.380	16.4	
EX74	858.75	1,761.51	–	–	1/30	EX740	0.011	0.005	4.541	1.930	412.8	
						EX741	0.011	0.004	4.690	1.645	413.8	
EX84	0.11	0.06	67.76	55.66	637.3	EX840	0.009	0.005	25.581	20.276	2,951.7	
EX94	10.17	1.50	2,745.50	1,645.57	269.9	EX940	0.547	0.211	13.448	3.775	24.6	
EX104	286.20	43.52	–	–	2/30	EX1040	0.509	0.048	57.863	45.987	113.8	

(together with EX410 and EX610). Less than 1 s is needed for the job sets 1 and 5. The remaining 20 instances require 1–9 s. This results in a tremendous speed-up for the job set 8 being more than 1,000. The speed-up in two orders of magnitude was achieved for the job set 7, 10 and some problems from 2–4 job sets. Finally, 21 other instances were running 35.6 faster on average, and the smallest speed-up was 16.2. It is good to see very good results across all the instances for the CP solver because such running times allow applying the solver even with the real-time demands.

As we can see, the performance of the CP engine is much better in comparison to the CPLEX engine using their standard setting. It has been shown that the CP approach is better than the MIP approach in more than an order of magnitude, being even in three orders of magnitude faster for some of the problems.

5 Conclusion

We have proposed a novel CP approach for scheduling with mobile robots. It is an important recent problem which needs to be handled in smart factories nowadays. The approach is very effective given our new proposal with non-overlapping constraints, including transportation activities. For more realistic data instances (42 problems), all solutions can be computed within one second, which allows real-time computation needed by a smart factory. In this case, our exact solver can replace even a hybrid heuristic solver proposed in [11] to compute solutions fast. Data instances from the second harder data set (40 problems) can be mostly solved within ten seconds, less than a quarter of them has higher computational demands. When we compare these results with the earlier MIP approach, there is a significant speed-up in one, sometimes two, or even three orders of magnitude. To conclude, this makes up a nice example of the constraint programming application.

In the future, we would like to extend further our problems based on real life. Interesting characteristics can be studied by consideration of an uncertain and dynamic environment where changes are happening due to the uncertain behavior of mobile robots or the existence of new jobs. Certainly, our interests lie in further improvements to the current model and search. Last but not least, we would like to study other combinations of scheduling and vehicle routing problems where non-overlapping constraints with transportation activities can be helpful.

Acknowledgements. We would like to thank the anonymous reviewers for their careful reading of our paper and their insightful comments and suggestions.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the program "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

References

1. IBM ILOG CPLEX Optimization studio CP Optimizer user's manual, version 12 release 8. IBM Corporation (2017)
2. IBM ILOG CPLEX Optimization studio OPL language user's manual version 12 release 8. IBM Corporation (2017)
3. Achterberg, T.: Random seeds. IBM Community CPLEX Optimizer Forum, IBM Corporation (2013)
4. Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. Eur. J. Oper. Res. **187**(3), 985–1032 (2008)
5. Artigues, C., Belmokhtar, S., Feillet, D.: A new exact solution algorithm for the job shop problem with sequence-dependent setup times. In: Régin, J.-C., Rueher, M. (eds.) CPAIOR 2004. LNCS, vol. 3011, pp. 37–49. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24664-0_3
6. Baptiste, P., Laborie, P., Le Pape, C., Nuijten, W.: Chapter 22 constraint-based scheduling and planning. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, pp. 761–799. Elsevier, Amsterdam (2006)
7. Berbeglia, G., Pesant, G., Rousseau, L.M.: Checking the feasibility ofdial-a-ride instances using constraint programming. Transp. Sci. **45**, 399–412 (2011)
8. Bilge, U., Ulusoy, G.: A time window approach to simultaneous scheduling of machines and material handling system in an FMS. Oper. Res. **43**(6), 1058–1070 (1995)
9. Cappart, Q., Thomas, C., Schaus, P., Rousseau, L.-M.: A constraint programming approach for solving patient transportation problems. In: Hooker, J. (ed.) CP 2018. LNCS, vol. 11008, pp. 490–506. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98334-9_32
10. Cordeau, J.F., Laporte, G.: The dial-a-ride problem: models and algorithms. Ann. Oper. Res. **153**(1), 29–46 (2007)
11. Dang, Q.V., Nguyen, C.T., Rudová, H.: Scheduling of mobile robots for transportation and manufacturing tasks. J. Heuristics **25**(2), 175–213 (2019)
12. Dang, Q.V., Rudová, H., Nguyen, C.T.: Adaptive large neighborhood search for scheduling of mobile robots. In: Proceedings of ACM GECCO Conference, pp. 224–232. ACM (2019)
13. Dejemeppe, C., Van Cauwelaert, S., Schaus, P.: The unary resource with transition times. In: Pesant, G. (ed.) CP 2015. LNCS, vol. 9255, pp. 89–104. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23219-5_7
14. Focacci, F., Laborie, P., Nuijten, W.: Solving scheduling problems with setup times and alternative resources. In: Artificial Intelligence for Planning and Scheduling (AIPS), pp. 92–101 (2000)
15. Grimes, D., Hebrard, E.: Job shop scheduling with setup times and maximal time-lags: a simple constraint programming approach. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 147–161. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13520-0_19
16. Kilby, P., Shaw, P.: Chapter 23 vehicle routing. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, pp. 801–836. Elsevier, New York (2006)
17. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP optimizer for scheduling. Constraints **23**(2), 210–250 (2018)

18. Lasi, H., Fettke, P., Kemper, H.G., Feld, T., Hoffmann, M.: Industry 4.0. Bus. Inf. Syst. Eng. **6**(4), 239–242 (2014)
19. Liu, C., Aleman, D.M., Beck, J.C.: Modelling and solving the senior transportation problem. In: van Hoeve, W.-J. (ed.) CPAIOR 2018. LNCS, vol. 10848, pp. 412–428. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93031-2_30
20. Madsen, O., et al.: Integration of mobile manipulators in an industrial production. Ind. Robot Int. J. Robot. Res. Appl. **42**(1), 11–18 (2015)
21. Nouri, H.E., Driss, O.B., Ghédira, K.: A classification schema for the job shop scheduling problem with transportation resources: state-of-the-art review. In: Silhavy, R., Senkerik, R., Oplatkova, Z.K., Silhavy, P., Prokopova, Z. (eds.) Artificial Intelligence Perspectives in Intelligent Systems. AISC, vol. 464, pp. 1–11. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33625-1_1
22. Parragh, S.N., Doerner, K.F., Hartl, R.F.: A survey on pickup and delivery problems, Part I: transportation between customers and depot. Journal für Betriebswirtschaft **58**(1), 21–51 (2008)
23. Parragh, S.N., Doerner, K.F., Hartl, R.F.: A survey on pickup and delivery problems, Part II: transportation between pickup and delivery locations. Journal für Betriebswirtschaft **58**(2), 81–117 (2008)
24. Pinedo, M.L.: Planning and Scheduling in Manufacturing and Services, 2nd edn. Springer, New York (2009)
25. Toth, P., Vigo, D.: Vehicle Routing: Problems, Methods, and Applications. Society for Industrial and Applied Mathematics, 2 edn. (2014)
26. Vilím, P., Laborie, P., Shaw, P.: Failure-directed search for constraint-based scheduling. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 437–453. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18008-3_30