Scheduling of Mobile Robots using Constraint Programming*

#53

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 [17] 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.* [20] 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 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 [20].

* Supported by organization x.

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]. 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 [16,2]. Our problem is a combination of scheduling [6] and vehicle routing [15]. 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 [21,22], 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.* [18] 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 [7,18,9] 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 [13,5,14]. Recently, the propagation procedure including transition times into the classic filtering algorithm for unary resources [12] was proposed to solve this problem. This is also the approach taken by CP Optimizer where non-overlapping constraint with transition times is available [16]. 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 which was not considered before to our best knowledge.

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 the 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 best 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. 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 the vehicle routing which is called the job shop scheduling with transportation resources [20]. 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). There is also a special loading/unloading (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 machine 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 perfom 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.

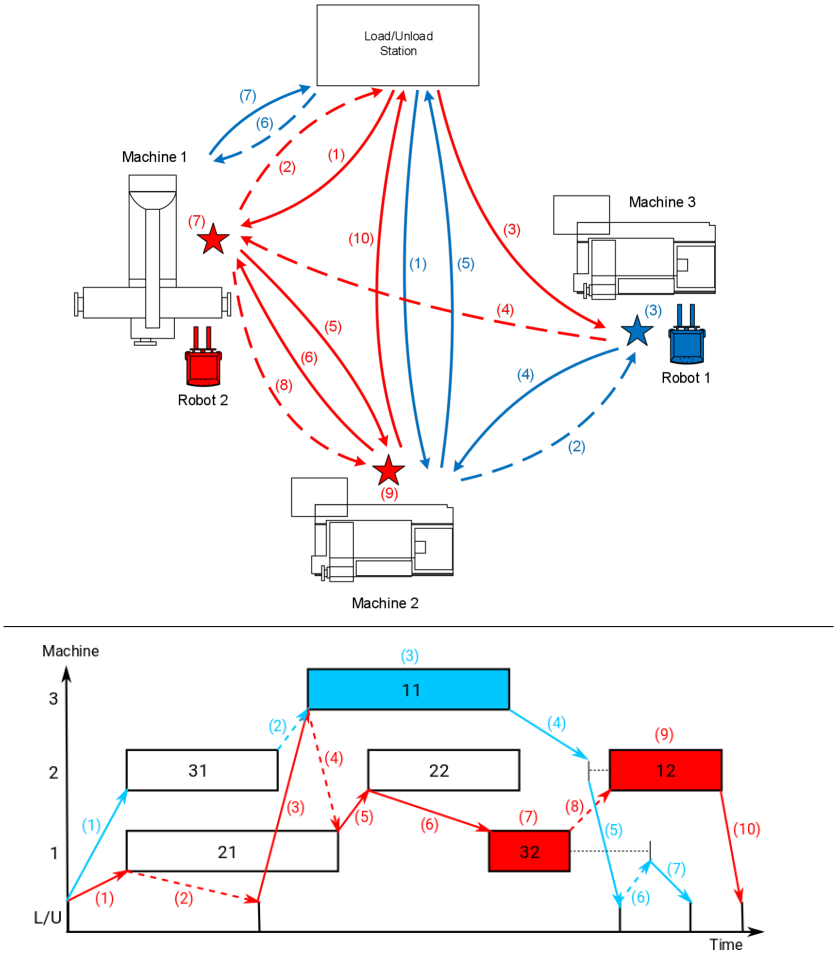

Fig. 1. Layout of FMS with mobile robots, and the schedule (based on [11]).

For instance, we can see that the third job is started by transportation (1) using the blue robot, consequently, the 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 and the transportation of the components (loaded trips). 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 Activities

As discussed before, there are `n` jobs and `m` machines. Interval variables are used to encapsulate both processing of job operations as well as transportations of job components between machines. For each job `j`, the interval variable `activity[j][o][s]` is available for each its processing (`s=0`) by the operation `o`, and each its transportation (`s=1`) of the job components from the machine where the operation `o-1` is processed to the machine where the operation `o` is processed (we will write shortly: from the operation `o-1` to the operation `o`). The transportation to the first operation starts at the L/U machine (which is furter index by `0`).

```
dvar interval activity[j in 1..n][o in 1..m][s in 0..1]
    optional (o > nbOperations[j])
    size activitySize[j][o][s];
```

The operations `o` of one job are listed in the array based on their order of processing on machines. The job `j` may not be processed on all machines, the number of jobs where it is processed equals to `nbOperations[j]`. This corresponds to the fact that the activities with `o > nbOperations[j]` are optional, and their presence is set to zero using the `presenceOf` constraint[1].

The size of the activity `activitySize[j][o][s]` corresponds either to the processing time of the operation (for `s=0`) or to the travel time (for `s=1`) between the machines where the operations `o-1` and `o` are processed.

Example 2. The operation 21 in Fig. 1 is represented by the processing activity `activity[2][1][0]`. The transportation (5) from the operation 21 to the operation 22 is represented by the transportation activity `activity[2][2][1]`.

Note that both processing and transportation is encapsulated in one array of interval variables to be later available in the `noOverlap` constraint for non-overlapping of selected activities.

3.2 Temporal Constraints

The activities for processing and transportation of one job are related by the precedence constraints. The transportation activity `activity[j][o][1]` from the operation `o-1` to the operation `o` must precede the processing activity `activity[j][o][0]` for each job `j`. Recall that the third index in the `activity` array corresponds to `0` and `1` for processing and transportation activities, respectively.

[1] Note that arrays of variable size are not available in CP Optimizer.

```
  forall (j in 1..n, o in 1..nbOperations[j])
    endBeforeStart(activity[j][o][1], activity[j][o][0]);
```

At the same time, the procesing activity `activity[j][o-1][0]` of the job `j` for the operation `o-1` must be followed by the transportation of the job components to the operation `o`, *i.e.*, by the transportation activity `activity[j][o][1]`.

```
  forall (j in 1..n, o in 2..nbOperations[j])
    endBeforeStart(activity[j][o-1][0], activity[j][o][1]);
```

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 (j in 1..n, o in 1..nbOperations[j]:
        operation[j][o].machine == i) activity[j][o][0];
```

such that the structure `operation[j][o].machine` stores the machine for each job `j` and its operation `o`. Consequently, the `noOverlap` constraint is posted for all machines.

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

3.4 Alternative Interval Variables

There is the interval variable `rbtActivity[r][j][o][s]` which ensures selection of one proper robot `r` for all transportation activities and for the processing activities which requires a robot for processing.

```
dvar interval rbtActivity
    [r in 1..rb][j in 1..n][o in 1..m][s in 0..1] optional
    size activitySize[j][o][s];
```

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. When some job `j` has less operations than machines (`nbOperations[j]<m` holds), then the excessive robot activities are set not to be present by the `presenceOf` constraint.

The following `alternative` constraint means that one robot transportation activity `rbtActivity[r][j][o][1]` is selected (is present) for each transportation activity `activity[j][o][1]`.

```
forall (j in 1..n, o in 1..nbOperations[j])
  alternative(activity[j][o][1],
              all (r in 1..rb) rbtActivity[r][j][o][1]);
```

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[j][o].processRequired` stores information about needed robots). Again one robot processing activity `rbtActivity[r][j][o][0]` will be present for each processing activity `activity[j][o][0]` requiring a robot.

```
forall (j in 1..n, o in 1..nbOperations[j]:
       operation[j][o].processRequired == 1)
       alternative(activity[j][o][0],
                   all (r in 1..rb) rbtActivity[r][j][o][0]);
```

In addition, all robot processing activities `rbtActivity[r][j][o][0]` 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-overlaping 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 `rbtActivity[r][j][o][0]` for all jobs `j` and operations `o` where a robot is required. In addition, there are the robot transportation activities `rbtActivity[r][j][o][1]`. Of course, the present activities are involved in the final sequence only.

```
dvar sequence rbtRoute[r in 1..rb]
    in all (j in 1..n, o in 1..m, s in 0..1)
    rbtActivity[r][j][o][s]
    types all (t in 1..rbtTotalActivities) activityType[t];
```

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 machine to the operation 11 (to the machine 3) by the robot transportation activity `rbtActivity[red][1][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 `rbtActivity[red][2][2][1]`. 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 (`rbtActivity[red][3][2][0]` at machine 1) to the destination activity (`rbtActivity[red][1][2][0]` 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 the activities (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 [16,15], 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 Section 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 `rbtActivity[red][1][1][1]` (3) has the type `03` because it corresponds to the transportation from the L/U machine to the machine 3. The activity `rbtActivity[red][2][2][1]` (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 `rbtActivity[red][3][2][0]` for the operation 32 by (7) has the type `11` as it is processed on the machine 1.

The final comment is related to the one dimensional array of types in the sequence variables `rbtRoute` presented above in their definiton. This is used to assign types to all activities in the sequence. Unfortunately, we learnt that types cannot have more than one dimension. So, we enconded the types for all robot activities `rbtActivity[r][j][o][s]` into one dimension represented by the array `activityType[t]` having the size `rbtTotalActivities` equal to `m*m*2`, *i.e.*, for each robot, we count an activity per job and operation doubled due to processing and transportation.

3.6 Different Approach with Pickup and Delivery Activities

The first approach solving our problem was based on interval variables both for pickup and delivery, *i.e.*, the definition of the interval variable `activity` as well

as `rbtActivity` extends the dimension `s in 0..2` such that `1`, `2` corresponds to pickup and delivery, respectively. Both pickup and delivery variables are time points, *i.e.*, their size equals to zero. As before, these intervals are set not to be present when `o > nbOperations[j]` holds.

Example 5. The transportation (5) in Fig. 1 from the operation 21 to the operation 22 is represented by the pickup activity `activity[2][2][1]` and the delivery activity `activity[2][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, j in 1..n, o in 1..nbOperations[j]) {
   presenceOf(rbtActivity[r][j][o][1]) ==
       presenceOf(rbtActivity[r][j][o][2]);
   prev(rbtRoute[r],rbtActivity[r][j][o][1],rbtActivity[r][j][o][2]); }
```

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.

```
forall (j in 1..n, o in 1..nbOperations[j])
   endAtStart( activity[j][o][1], activity[j][o][2],
      travelTimes[operation[j][o-1].machine][operation[j][o].machine]);
```

Finally, there is a slightly different implementation of the precedence constraints from Section 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 Redundant Constraints

Earliest Starting Time We have tried to implement various additional constraints. The only important constraint is related to the earliest starting time of the `activity`. For each operation, we sum up processing times of all operations of the same job which precede it, and travel times of all transportations between machines of the preceding operations of this job. This sum is used as the earliest starting time of each processing activity. The same sum can be computed for each transportation activity, again based on the preceding processing and transportation activities of the same job.

Precedences between Operations The common constraints in job shop problems express precedence relations between two consecutive operations (the processing activities) of one job.

```
forall (j in 1..n, o in 2..nbOperations[j])
   endBeforeStart( activity[j][o-1][0], activity[j][o][0]);
```

In our case, this is a redundant constraint, because we already have temporal constraints (see Section 3.2) between the transportation activities and the processing activities of one job.

Cumulative constraints Often it is helpful to accompany several `noOverlap` constraints related to the same set of activities by the cumulative function constraining the total number of resources. We can have cumulative functions for all machines

```
cumulFunction machineUsage =
   sum (j in 1..n, o in nbOperations[j]) pulse(activity[j][o][0],1);
machineUsage <= m;
```

and for all robots as well.

```
cumulFunction robotUsage =
   sum (r in 1..rb, j in 1..n, o in nbOperations[j], s in 0..1)
       pulse(rbtActivity[r][j][o][s],1);
robotUsage <= rb;
```

3.8 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. 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(j in 1..n) endOf(activity[j][nbOperations[j]][0]);
```

3.9 Search

CP Optimizer [16] employs a combination of Large Neighbourhood Seach and Failure-directed Search [23] by default. Other options are introduced by multi-point search and depth-first search [1,16]. In the experimental part, we will explore differences among these search algorithms. Based on our analysis, the default search is significantly better than other options (see Section 4.2).

Other 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[2]. We asked authors of the paper [11] for the MIP implementation in OPL to compare it 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 [16,3]. 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

To provide a reasonable comparison, we use data instances from [11] available from the website[3]. 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 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 [19].

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 settings of the model are presented in the first six lines. All of them were completed using the default search. The base model includes transportation activities (denoted *Basic* in Table 1), and other models extend it by redundant constraints (see Section 3.7). The version *Precedence*, *Cumulative*, and *Default* includes precedences between operations, cumulative constraints, and the earliest starting time, respectively. Finally, the experiment *All redund.* includes all redundant constraints.

The model denoted *Default* including transportation activities with the enhanced earliest starting times performs the best. Other redundant constraints

[2] In case of publication, we will make the source code publicly available.
[3] 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.
Basic	0.011	0.004	0.246	0.189	277.189	13.758	25,855.813	1,172.359	0.566	0.057
Precedence	0.010	0.005	0.285	0.189	287.964	13.062	26,212.665	1,171.287	0.607	0.051
Cumulative	0.015	0.006	0.283	0.142	–	–	–	–	0.852	0.097
All redund.	0.011	0.005	0.089	0.099	1.272	0.218	1.168	0.942	0.841	0.076
Default	0.010	0.006	0.048	0.020	0.799	0.187	1.018	0.936	0.583	0.072
DFS	0.005	0.005	56.061	1.607	0.405	0.040	8.382	6.915	16.049	0.841
MultiPoint	0.020	0.008	0.097	0.043	–	–	–	–	–	–

have not improved the performance of the model at all. We can see the critical importance of constraining the earliest starting time in models *Default* and *All redund.*, since other models have poor performance even on some rather easy problems. See EX11 and EX22 where the model with cumulative constraints was even unable to solve problems within 12 hours. Versions *Basic* and *Precedence* were not good either with the worst results on EX22.

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 Section 3.6. Performance of this model was very weak resulting in 96.794±5.086 seconds 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, search algorithms are compared in the last three lines where the results for the default search (*Default*), the depth-first search (*DFS*), and the multi-point search (*Multi-point*) are given. Note that all experiments were completed for the best setting of the model as discussed above.

While the best performance of the default search is not clear based on the instance EX110 and EX11 where the *DFS* is about 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 1,167.94, 8.23, and 27.53 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 hours for three problems, even though there are still rather easy data instances.

Given all the experiments, we can conclude that the default setting of CP Optimizer with transportation activities enhanced by the computation of the earliest starting time is the best setting which is further used to perform experiments on all data instances.

4.3 Results and Comparison with MIP

We compare results of our best CP solver with transportation activities and the earliest starting time 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 $t/p < 0.25$.

The results for the first set of 40 data instances were mostly computed within 10 seconds (15 instances within a second, other 16 instances in less than 10 seconds). The three most demanding instances EX71, EX74, and EX104 needed almost an hour, half an hour, and 5 minutes, respectively. The remaining 6 instances required the computational time between 10−100 seconds (23.70 seconds on average). For the MIP solver, the instances EX71, EX74, and EX104 are very hard, the results were computed within 12 hours 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 seconds. 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, 17, and 17 problems, respectively. The smallest speed-up was 9.4, and it was 34.9 on average for 17 problems with the speed-up smaller than a hundred.

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. All problems from the 9 and 10 job sets need computational time higher than 0.5 second. The job sets 4 and 6 with the exclusion of EX420 require between 0.5 and 0.1 second. The job set 2 with the exclusion of EX241 (the only problem with tripled processing time) together with EX420 has computational times of 0.1 – 0.02 second. 23 remaining instances suffice with solution time smaller than 0.02 second on average. The MIP solver needs more than 9 seconds for the job sets 8–10 (together with EX410). Less than 1 second is needed for the job sets 1 and 5. The remaining 21 instances require 1–9 seconds. 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 and some problems from 2–4 job sets. Finally, 27 other instances were running 47.1 faster on average, and the smallest speed-up was 14.0. It is good to see very good results across all the instances for the CP solver because such running times allows 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

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	0.80	0.19	21.09	14.00	26.4	EX110	0.010	0.006	0.418	0.097	41.8
EX21	7.95	1.91	813.09	307.91	102.2	EX210	0.048	0.020	5.052	1.084	106.0
EX31	2.41	0.44	151.71	54.37	62.9	EX310	0.017	0.005	2.401	1.293	138.5
EX41	32.23	6.92	11,062.71	4,671.52	343.2	EX410	0.479	0.121	10.043	2.625	21.0
EX51	2.68	0.50	103.41	28.65	38.5	EX510	0.008	0.007	0.451	0.113	54.1
EX61	8.23	1.33	2,144.12	819.86	260.7	EX610	0.272	0.106	9.954	12.807	36.6
EX71	3,126.82	3,323.14	–	–	0/30	EX710	0.011	0.004	2.318	1.471	217.3
EX81	0.02	0.01	39.11	28.99	2,444.4	EX810	0.011	0.005	43.181	30.684	3,925.6
EX91	5.28	0.91	1,471.48	1,134.73	278.7	EX910	0.590	0.360	10.945	2.517	18.5
EX101	44.24	7.14	11,034.87	3,868.56	249.5	EX1010	0.634	0.296	52.802	69.282	83.3
EX12	0.22	0.08	2.15	0.76	9.9	EX120	0.008	0.004	0.333	0.071	43.4
EX22	1.02	0.94	12.47	5.74	12.2	EX220	0.045	0.031	3.975	0.905	89.0
EX32	0.59	0.44	5.52	0.97	9.4	EX320	0.014	0.005	1.217	0.592	89.0
EX42	5.97	0.80	1,439.60	614.47	241.3	EX420	0.059	0.019	7.103	1.787	119.7
EX52	0.29	0.07	3.57	1.11	12.2	EX520	0.010	0.004	0.449	0.147	46.4
EX62	0.41	0.06	38.15	28.02	92.8	EX620	0.217	0.085	7.555	2.816	34.8
EX72	1.80	0.31	269.45	145.27	149.4	EX720	0.011	0.006	4.875	1.840	430.2
EX82	0.02	0.00	72.78	63.20	4,645.8	EX820	0.008	0.004	27.567	21.253	3,595.7
EX92	0.89	0.19	166.36	79.32	187.0	EX920	0.633	0.431	9.306	3.044	14.7
EX102	2.61	0.70	555.15	184.64	212.4	EX1020	0.539	0.302	42.312	29.385	78.5
EX13	0.36	0.16	3.98	1.11	11.1	EX130	0.009	0.004	0.361	0.058	38.6
EX23	1.47	0.68	65.59	37.82	44.7	EX230	0.048	0.065	4.091	0.805	85.8
EX33	0.30	0.04	5.17	1.11	17.0	EX330	0.015	0.005	1.171	0.475	76.4
EX43	6.97	0.69	1,481.41	673.00	212.7	EX430	0.223	0.052	4.488	1.168	20.2
EX53	0.64	0.09	14.61	7.14	22.7	EX530	0.009	0.005	0.367	0.098	40.8
EX63	0.58	0.07	86.11	52.28	147.6	EX630	0.287	0.146	6.646	1.705	23.1
EX73	5.42	1.75	661.64	341.92	122.0	EX730	0.011	0.006	4.340	1.602	383.0
EX83	0.01	0.00	76.68	83.95	7,932.7	EX830	0.007	0.005	26.695	18.926	3,813.5
EX93	1.64	0.23	356.05	144.02	217.5	EX930	0.695	0.522	9.748	3.511	14.0
EX103	3.06	0.69	948.83	366.85	310.2	EX1030	0.723	0.469	41.732	22.047	57.7
EX14	1.00	0.22	22.03	15.27	21.9	EX140	0.011	0.004	0.440	0.146	41.3
EX24	10.75	1.85	610.69	241.69	56.8	EX241	0.013	0.006	3.407	0.974	255.6
EX34	4.36	0.89	205.19	87.97	47.0	EX340	0.014	0.005	2.282	1.213	159.2
						EX341	0.015	0.005	1.637	0.637	111.6
EX44	20.86	3.88	2,316.20	994.28	111.1	EX441	0.283	0.047	7.826	1.841	27.7
EX54	2.42	0.38	84.69	29.51	34.9	EX541	0.007	0.004	0.464	0.100	63.2
EX64	22.49	4.47	1,651.60	772.60	73.4	EX640	0.444	0.335	8.311	2.380	18.7
EX74	1,880.81	4,796.44	32,366.03	–	1/30	EX740	0.013	0.007	4.541	1.930	358.5
						EX741	0.012	0.005	4.690	1.645	380.3
EX84	0.44	0.28	67.76	55.66	152.5	EX840	0.010	0.005	25.581	20.276	2,646.3
EX94	11.65	1.69	2,745.50	1,645.57	235.7	EX940	0.528	0.213	13.448	3.775	25.5
EX104	298.28	68.60	34,239.82	–	2/30	EX1040	0.653	0.476	57.863	45.987	88.7

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, a quarter of them has higher computational demands. When we compare these results with the earlier MIP approach, there is a significant speed-up. For 20, 15 instances out of 40, 42 instances, the speed-up was in more than two orders of magnitudes, respectively. 3 data instances have been mostly unsolved by MIP within 12 hours. For the remaining instances, there is an average speed-up being 34.9 and 47.1 for 17 out of 40 instances and 27 out of 42 instances, respectively. There were only 2 out of 40 instances solved less than ten times faster (9.4 and 9.9), the smallest speed-up for another set of 42 instances was 14.0. To conclude, this makes up a nice example of the constraint programming application.

In the future, we would like to further extend our problems based on real life. Interesting characteristics can be studied by consideration of uncertain and dynamic environment where changes are happening due to uncertain behaviour of mobile robots or 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.

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. European Journal of Operational Research **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.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 37–49. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

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 (2006)
7. Berbeglia, G., Pesant, G., Rousseau, L.M.: Checking the feasibility of dial-a-ride instances using constraint programming. Transportation Science **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. Operations Research **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.) Principles and Practice of Constraint Programming. pp. 490–506. Springer International Publishing (2018)
10. Cordeau, J.F., Laporte, G.: The dial-a-ride problem: models and algorithms. Annals of Operations Research **153**(1), 29–46 (2007)
11. Dang, Q.V., Nguyen, C.T., Rudová, H.: Scheduling of mobile robots for transportation and manufacturing tasks. Journal of Heuristics **25**(2), 175–213 (2019)
12. Dejemeppe, C., Van Cauwelaert, S., Schaus, P.: The unary resource with transition times. In: Pesant, G. (ed.) Principles and Practice of Constraint Programming. pp. 89–104. Springer International Publishing (2015)
13. 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)
14. Grimes, D., Hebrard, E.: Job shop scheduling with setup times and maximal timelags: A simple constraint programming approach. In: Lodi, A., Milano, M., Toth, P. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 147–161. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
15. 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 (2006)
16. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP optimizer for scheduling. Constraints **23**(2), 210–250 (2018)
17. Lasi, H., Fettke, P., Kemper, H.G., Feld, T., Hoffmann, M.: Industry 4.0. Business & Information Systems Engineering **6**(4), 239–242 (2014)
18. Liu, C., Aleman, D.M., Beck, J.C.: Modelling and solving the senior transportation problem. In: van Hoeve, W.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 412–428. Springer International Publishing, Cham (2018)
19. Madsen, O., Bøgh, S., Schou, C., Andersen, R.S., Damgaard, J.S., Pedersen, M.R., Krüger, V.: Integration of mobile manipulators in an industrial production. Industrial Robot: The International Journal of Robotics Research and Application **42**(1), 11–18 (2015)
20. 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. pp. 1–11. Springer International Publishing (2016)
21. 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)

22. 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)
23. Vilím, P., Laborie, P., Shaw, P.: Failure-directed search for constraint-based scheduling. In: Michel, L. (ed.) Integration of AI and OR Techniques in Constraint Programming. pp. 437–453. Springer International Publishing (2015)