Production Line Optimization with Model Based Methods

T. Hajba, Z. Horváth, C. Kiss-Tóth, and J. Jósvai

Abstract In this paper we deal with different models of production lines of factories and consider the makespan optimization problem based on these models. We consider state-of-the-art and novel mathematical optimizers including exact and heuristic methods. We apply these optimizers to both standard academic and industrial data sets. We see that in a large rate of the considered cases the novel exact optimizers converged to the optimum fast which is surprising being the problems NP-hard and the problem sizes big. This shows the importance of exploiting the structure present in the industrial data using standardized industrial data sets for testing mathematical algorithms devoted to solve industrial problems and that some provided exact mathematical optimizers are fast and perform accurately on the considered industrial problems.

1 Introduction

One of the most important tasks at modern factories is the optimal scheduling of the order of jobs on the production lines of the factory since this affects highly the makespan of the set of jobs to be processed per day and thus determines the total number of jobs that can be processed per day. Though experienced managers can provide satisfactory schedules, with the components of the realization of the Digital Factory concept schedules of higher quality can be achieved. Namely, the Digital Factory concept (see [21]) can be regarded as mapping of the real physical processes of the factory to tools of the information technology. The Digital Factory methodology includes production simulation tools using real manufacturing data (bill of materials, production plan, operation sequence, makespan, capacity usage, lateness, etc.). In everydays environment where the production scheduling tasks are situation driven - because of unreliable information, production line fall out, material delivery failures, etc. - simulation based scheduling is asked. A production system has many influential parameters, to optimize a set of jobs in a system where there are conflicting goals, so the mathematical solving method can be very

T. Hajba • Z. Horváth (✉) • C. Kiss-Tóth • J. Jósvai
Széchenyi István University, Győr, Hungary
e-mail: hajbat@sze.hu; horvathz@sze.hu; ktchris@sze.hu; josvai@sze.hu

© Springer International Publishing AG 2017

L. Ghezzi et al. (eds.), *Math for the Digital Factory*, Mathematics in Industry 27,
DOI 10.1007/978-3-319-63957-4_8

complex. Therefore a combination of simulation tools and mathematical methods can be an effective answer for this problem. Especially when we consider the Industry4.0 vertical and horizontal integration process within the production system. This integration and distributed decision making methodology need sophisticated modeling, simulation and mathematical optimization methods and tools. These aspects let us think about our research work, which combines modeling, simulation and mathematical optimization of production lines, gives important results for the next steps in the field of smart and networked digital factory. This allows automatic construction of models and/or simulations for the examined production flow, in our case the work on the production line, and poses the corresponding optimization problems at model/simulation level. Then the inverse mapping of the optima of the model or simulation based optimization problems gives a suggestion to the production line managers for the actual scheduling.

We note that often a combination of models and simulations is advantageous. Indeed, simulations validated at high accuracy are typically time consuming and thus simulation based optimization takes often too long for using it in a daily routine of a company. On the other hand, one objective evaluation at model based optimization is typically much faster than that with simulation but the accuracy of fast models is lower than that with an enhanced simulation. Thus a fast and reliable optimization can consist of a model based hierarchic optimization that includes the evaluation of the design elements with accurate simulation to check whether the actually used model is accurate enough.

In this paper we deal with optimal scheduling of the production lines using mathematical models and their optimization methods. Namely, we provide a review of the models based on mathematical optimization methods and test results on academic and industrial data. In Sect. 2 we define the considered models to the production line: first the basic model, the Permutation Flow Shop Problem (PFSP), which is studied thoroughly in the literature and then the recently introduced and studied models that handle more features of real industrial production lines than PFSP, the Permutation with Repetition Flow Shop Problem (R-PFSP) and the Permutation with Repetition Flow Shop Problem with Buffers and Palettes (PB-R-PFSP) (see [3]). Here repetition refers to the fact that in industrial situations there exist several types of jobs and each schedule contains many jobs of the same type. Then in Sect. 3 we define and discuss some frequently used heuristic optimization methods for the PFSP models. In Sect. 4 first we present mixed integer linear programming (MILP) formulations for the PFSP, which has been studied in several papers, see [8, 9, 12, 16–19, 22, 23] and three new MILP models of [2]. According to numerical experiments in papers [18, 19] the MILP formulations of the PFSP models combined with exact solvers (e.g. those based on branch and bound) are capable to solve only small sized PFSPs. However, applying our new MILP models to R-PFSP and PB-R-PFSP we see in Sect. 5 that large scale industrial problems arising from the automotive industry (see [5]) and their analogues become exactly solvable. We close the chapter with drawing conclusions.

2 Production Line Models: The PFSP and the PB-R-PFSP

In the regular permutation flow shop problem we are given a production system of M machines and a set of N jobs. Every job has to be processed on every machine in the same order, i.e. every job has to be processed first on the first machine then on the second machine and so on. The processing times of the jobs at the machines are known in advance and deterministic. The regular PFSP entails the following assumptions which form the constraints of the optimization problem:

- Machines are continuously available from time 0.
- Every job is available for processing at time 0.
- Each job can be processed only on one machine at a time.
- Each machine can process at most one job at a time.
- The jobs must be processed on the machines without preemption.
- Setup times are included in the processing times, or ignored.
- Any number of jobs can wait between consecutive machines.

In this paper the objective of the optimization problem is the minimization of the makespan, i.e. the minimization of the completion time of the last job of the order on the last machine.

Many manufacturing problems have special features which are not included in the regular PFSP. One such property is that in real-life problems we often have a lot of jobs that have the same processing times on every machines (see [2]). It is said in this case that these jobs have the same type. Taking this property into consideration can drastically reduce the number of different permutations, i.e. the design space for the PFSP. Namely, if there are T types and n_t jobs of type t then the number of permutation reduces form $N!$ to $\frac{N!}{(n_1)!\cdot(n_2)!\ldots(n_t)!}$.

Another property of industrial situations is the presence of palette usage at lines. Namely, jobs are often carried on palettes on the line and the number of palettes is bounded from above, typically less than the number of jobs. This means that only a limited number of jobs can be on the line at a time.

Moreover, since the palette size and the space between consecutive machines are given, only a limited number of palettes (hence limited number of jobs) can wait between consecutive machines which means that there are limited buffer sizes between the consecutive machines.

These properties are summarized as follows:

- the number of different types of the jobs is less than the total number of the jobs;
- only a given number of palettes can be used to carry the jobs;
- the buffer sizes between consecutive machines are finite and given in advance;

We call a PFSP which contains repeated jobs, limited buffer sizes between consecutive machines and bounded number of palettes *Permutation with Repetition Flow Shop Problem with Palettes and Buffer* (PB-R-PFSP).

3 Heuristic Optimization Methods for the PFSP Model

During the last three decades many kind of heuristic approaches were published to solve the PFSP problem. In this section we are going to give a review about some of these algorithms we tested on our problem sets.

3.1 NEH Heuristic

One of the most famous constructive solution for the FPSP was proposed by Nawaz, Enscore and Ham in 1983 [10]. Their algorithm is based on the assumption that jobs with higher total processing time make bigger effect on the objective function, this means we should give higher priority to them. NEH algorithm constructs a solution by inserting the jobs into an empty permutation one by one. We are going to describe the details below.

First of all we calculate the total processing times for all jobs, and after that we choose two jobs with the highest value. We consider the two possible partial schedules, and choose the better one. During the rest of the algorithm the relative positions of these two jobs are fixed.

In the next step we pick up the job with the third highest total processing time, and put it into the permutation to the proper place found by an exhaustive search procedure. This means that we place the job to the first, the second and to the third (last) place of the partial sequence, and keep the best solution we get. This process is repeated until all the jobs are placed into the permutation. The pseudo-code of the algorithm can be presented as follows (Fig. 1).

The major advantage of the NEH algorithm is that we are able to get a good solution in a short time, since the total number of iterations (makespan evaluations of partial solutions) is $\frac{N(N+1)}{2}$. Although many algorithms can be found in the literature, NEH is still a state of the art algorithm for the PFSP optimization and also used for creating initial solutions for many other heuristic methods with higher computational complexity such as in the following two heuristics described in the next sections.

```
1: For each job $i$ calculate $T_i = \sum_{r=1}^{M} P_{ri}$ where $P_{ri}$ is the processing time of job $i$ on machine $r$.
2: Arrange the job indices $i$ into a list $L$ in descending order of $T_i$.
3: Pick the first two jobs of the list $L$, and find the best permutation $\pi_2$ of these two jobs by
   calculating the makespan for the two possible partial solutions. Fix the relative positions of
   these two jobs for the remaining steps. Set $i = 3$.
4: Pick the job from the $i$th position of the list $L$ and find the best sequence by inserting it at all
   possible positions of $\pi_{i-1}$ (without changing the relative positions of the previous jobs).
5: If $N = i$ then STOP, otherwise set $i = i + 1$, and go to Step 4.
```

Fig. 1 Pseudo-code for the NEH algorithm

3.2 Ant-Colony Algorithms

The name ant-colony algorithms (ACO) group a family of techniques to solve combinatorial optimization problems. In the framework of these techniques the motivation is to imitate the pheromone trail in the nature used by real ants for communication and feedback. Basically, these techniques are population-based, cooperative search procedures. During the optimization ant-colony algorithms use simple agents (called ants) that iteratively construct permutations, and this solution construction is guided by these artificial pheromone trails. The details how we calculate these artificial trails is a problem-specific heuristic information. This information has to be tuned for the problem we would like to solve.

To understand this optimization method we have to define the meaning of solution components. Ants iteratively construct possible permutations from these components, and leave pheromone. In this context pheromones indicate the intensity of ant-trails with respect to solution components. These values are determined from the influence of each solution component to the objective function. The trails also form a kind of adaptive memory in this search procedure: we update the intensities at the end of each iteration, the effect of the permutation created for the last is the biggest. In this framework π_{ij} denotes the trail intensity of setting job i in position j of a sequence. Since every job can be placed at every place we have to store N^2 intensity values.

In every iteration one single ant constructs a complete solution starting with an empty permutation and iteratively adding components until a complete solution is constructed. After the construction each ant gives feedback on the solution by leaving pheromone (updating trail intensity) on each solution component.

After updating the intensities we apply a local search scheme for the created permutation to find possibly the best solution in the neighborhood. Summarizingly the general structure of ACO algorithms can be described in Fig. 2.

Since many kind of ACO algorithms were published for the PFSP problem, we chose one, named PACO published by Rajendran and Ziegler in 2004 [13], supposed to perform the best on the Talliard problem set. In the rest of this section we will present the details of the PACO algorithm.

Fig. 2 General pseudo-code for ACO algorithms

```
1: Initialize the pheromone trails and parameters.
2: While termination condition is not met do the following:
     ▷ construct a solution;
     ▷ improve the solution by local search;
     ▷ update the pheromone trail intensities.
3: Return the best solution found.
```

3.2.1 Initialization

For initializing the trail intensities we need an initial solution. Usually we can use the NEH heuristic to find an initial solution, in PACO we improve this by applying the job-index-based local search three times. We denote the makespan of this solution with Z_{best}. After this the initialization of the pheromone trails looks as follows:

$$\pi_{ik} = \begin{cases} \frac{1}{Z_{best}} & \text{if } |\text{position of job } i \text{ in the seed sequence} - k| + 1 \leq \frac{n}{4} \\ \frac{1}{2Z_{best}} & \text{if } \frac{n}{4} < |\text{position of job } i \text{ in the seed sequence} - k| + 1 \leq \frac{n}{2} \\ \frac{1}{4Z_{best}} & \text{otherwise.} \end{cases}$$

The idea behind this initialization is that we think the initial solution is good enough, so putting a job on a place which is closer to his place in the initial solution probably gives better result.

3.2.2 Construction of a New Solution

Every iteration of the PACO algorithm creates a new permutation iteratively. We place an unscheduled job i for place k using the following scheme:

- $T_{ik} = \sum_{q=1}^{k} \pi_{ik}$ and sample a uniform random number u in range [0, 1].
- If $u \in [0, 0.4]$: choose the best unscheduled job in the best sequence obtained so far.
- If $u \in (0.4, 0.8]$: among the first five unscheduled jobs in the best sequence choose the job with the maximum T_{ik} value.
- If $u \in (0.8, 1]$: among the set of the first five unscheduled jobs in the best sequence select job i with the probability of $\frac{T_{ik}}{\sum_l T_{lk}}$.

If the number of the unscheduled jobs is less then five, then consider all of them. After this procedure apply the job-index-based local search procedure three times, and denote the makespan of this solution with $Z_{current}$.

3.2.3 Update of the Pheromone Intensities

Let h denote the position of job i in the resultant sequence. If the number of jobs is less or equal to 40:

$$\pi_{ik}^{new} = \begin{cases} \varrho \cdot \pi_{ik}^{old} + \frac{1}{diff \cdot Z_{current}} & \text{if } |h - k| \leq 1 \\ \varrho \cdot \pi_{ik}^{old} & \text{otherwise.} \end{cases}$$

If the number of jobs is greater than 40:

$$\pi_{ik}^{new} = \begin{cases} \varrho \cdot \pi_{ik}^{old} + \frac{1}{diff \cdot Z_{current}} & \text{if } |h-k| \leq 2 \\ \varrho \cdot \pi_{ik}^{old} & \text{otherwise.} \end{cases}$$

where $diff = (|\text{position of job } i \text{ in the best sequence obtained so far } - k| + 1)^{\frac{1}{2}}$ and ϱ is a constant, fixed as 0.75 during the algorithm.

We ran the PACO algorithm for 300 iterations.

3.3 Tabu Search Approaches

Tabu search (TS) is a general framework which can be used to find near-optimal solutions of hard combinatorial optimization problems. Using a tabu search algorithm first the neighborhood of a solution has to be defined. Then starting from an initial solution, at each iteration the algorithm examines the neighborhood of the actual solution and one of the neighbors (usually the best) is chosen to be the actual solution in the next iteration. To avoid returning back to a previously used solution tabu search algorithms use a so-called tabu list containing elements of forbidden moves. The algorithm stops if a stopping criterion (for example the number of total iterations) is reached.

Next we describe in more detail the TS heuristic for the PFSP of Nowicki and Smutnicki [11].

3.3.1 Moves and Neighborhood

In the PFSP a solution is represented as a permutation. To be able to define the neighborhood of a permutation we first introduce the concept of moves. Let π be a permutation and a and b two positions. Removing the job from position a of the permutation and putting it into position b of the permutation is called a *move* and denoted with (a,b). The *neighborhood* of π is the set of permutations that can be reached from π with one move. Each job we remove can be placed to $N-1$ places, since the move (a,a) does not change the permutation. Furthermore the moves $(a-1,a)$ and $(a,a-1)$ yield the same permutation hence every permutation has $(N-1)^2$ neighbors. Since searching in such a large neighborhood would be time-consuming we will reduce the number of potential neighbors. In order to be able to do this we will introduce the definition of blocks and critical path.

It is known, that the makespan of a permutation π can be written in the following form:

$$C_{\max}(\pi) = \max_{1=j_0 \leq j_1 \leq \cdots \leq j_{M-1} \leq j_M = N} \sum_{i=1}^{M} \sum_{j=j_{i-1}}^{j_i} p_{i\pi(j)}. \quad (1)$$

Let us associate with each permutation π a directed grid graph $G(\pi) = (V, E)$ with node weights:

$$V = \{1, \dots, M\} \times \{1, \dots N\},$$

$$E = \bigcup_{i=1}^{M} \bigcup_{j=1}^{N-1} \{((i,j), (i,j+1))\} \cup \bigcup_{i=1}^{M-1} \bigcup_{j=1}^{N} \{((i,j), (i+1, j))\}.$$

where the node (i,j) represents the ith machine and the jth job of the permutation and the weight of node (i,j) is $P_{i\pi(j)}$. Then formula (1) means that the makespan of π is the weight of the critical (longest) path in this grid graph, from node $(1, 1)$ to (M, N).

Let us suppose that the critical path in $G(\pi)$ is $(1, s_0), \dots, (1, s_1), (2, s_1), \dots, (2, s_2), \dots, (M, s_{M-1}), \dots, (M, S_M)$, where $1 = s_0 \leq s_1 \leq \dots \leq s_M = n$. Then this path consists of the vertical edges $((i, s_i), (i + 1, s_i)$ for $i = 1, 2, \dots, M - 1$ and horizontal subpaths $(i, s_{i-1}), \dots, (i, s_i)$ if $s_{i-1} < s_i$. If for machine i condition $s_{i-1} < s_i$ holds then the sequence of positions $s_{i-1}, s_{i-1} + 1, \dots, s_i$ is called a *block*. Each position can be contained in one or two blocks.

It can be proven that performing a move $v = (a, b)$ to permutation π for which a and b are inside the same block gives us a solution π' for which $C_{\max}(\pi') \geq C_{\max}(\pi)$, which means we do not have to analyze such neighbors. Based on their experiments Nowitzki and Smutnicki reduced further the neighborhood of a permutation.

A move (a, b) is called a right move, if $a < b$ else the move is called a left move. Tests showed that for position a if a is a beginning of the lth block or lies inside the lth block then it is enough to examine right moves (a, b) in which b lies in the first few positions of the $(l + 1)$th block (the next block). Similarly if a is an end of the lth block or lies inside the lth block then it is enough to examine left moves (a, b) in which b lies in the last few positions of the $(l - 1)$th block (the previous block). The algorithm uses the parameter ε which controls that for a exactly how many positions b from the next and previous blocks are considered for the right and left moves. The value of ε depends on the M and N, namely

$$\varepsilon = \begin{cases} 0 & \text{if } \frac{N}{M} > 3 \\ 0.5 & \text{if } 2 < \frac{N}{M} \leq 3 \\ 1 & \text{if } \frac{N}{M} \leq 2 \end{cases}$$

Denoting by $ZR_j(\pi, \varepsilon)$ the set of the above defined right moves of position j and by $ZL_j(\pi, \varepsilon)$ the set of the above defined left moves for position j the set of investigated moves:

$$Z(\pi, \varepsilon) = \bigcup_{j=1}^{N-1} ZR_j(\pi, \varepsilon) \cup \bigcup_{j=2}^{N} ZL_j(\pi, \varepsilon)$$

and the solutions we get after performing these moves are the neighbors of the solution π. It is worth mentioning that if $\varepsilon = 0$ then for every position j at most one left and one right move is examined hence in this case the neighborhood of a permutation contains at most $2n - 2$ elements. As we mentioned earlier for every permutation there are overall $(n - 1)^2$ different moves which means that the above procedure can drastically reduce the size of the neighborhood of a permutation making the algorithm to be faster. After defining the moves we describe the tabu list, the main idea of this heuristic approach.

3.3.2 Tabu List

Tabu list is a technique to prevent cycling during the search procedure. The TS algorithm of Nowicki and Smutnicki uses a tabu list with fixed length (*maxt*) (i.e. the list can contain *maxt* elements) which contains pair of jobs, initialized with elements $(0, 0)$ at the beginning. If during the algorithm a move $v = (a, b)$ is performed, then the first element of the list is deleted and the pair $(\pi(a), \pi(a + 1))$ if $a < b$ and $(\pi(a - 1), \pi(a))$ otherwise is added to the end of the list.

In the search procedure a move (a, b) from permutation β is „tabu" if at least one pair $(\beta(j), \beta(a)), j = a + 1, \dots, b$ is in the tabu list if $a < b$, and at least one pair $(\beta(a), \beta(j)), j = b, \dots, a - 1$ is in the tabu list otherwise.

After we defined all components of the algorithm, we describe the searching strategy.

3.3.3 Neighborhood Searching Strategy

At the first stage of the searching procedure from every set ZL_j and ZR_j we choose a representative with the smallest makespan. In this way we get a new set of neighbors containing $2(n - 1)$ elements independently from the value of ε denoted with $\hat{Z}$.

At the second stage we classify these solutions into three categories: unforbidden (*UF*), forbidden but profitable (*FP*) and forbidden and non-profitable (*FN*). A forbidden move from π is profitable, if it leads to a solution whose makespan is less then $F(C_{\max}(\pi))$ where F is an aspiration function.

Finally we decide to perform the best move from the set of the *UF* and the *FP* moves. If all moves are *FN* then we add $(0, 0)$ to the tabu list; this process is repeated until an *UF*-move can be chosen (Fig. 3).

```
1: Find sets $UF$-moves $X = \{v \in \hat{Z}(\pi, \varepsilon) : v \text{ is not tabu}\}$ and $FP$-moves $Y = \{v \in \hat{Z}(\pi, \varepsilon) :$
   $v \text{ is tabu}, C_{\max}(\pi_v) < F(C_{\max}(\pi))\}$.
2: If $X \cup Y \neq \emptyset$ then select $v' \in X \cup Y$ with the smallest makespan. Update the tabu list and Exit.
3: Add a zero element to the tabu list and go to Step 2.
```

Fig. 3 Pseudo-code for the neighborhood searching procedure

```
1: Set $\pi := \pi^*$, $C^* := C_{\max}(\pi)$, $iter := 0$, $ret := 0$, $T := \emptyset$ and $save = true$. Set $F(x) := \infty$
   for all $x$ non-negative integer.
2: Set $iter := iter + 1$, $ret := ret + 1$. Find $Z(\pi, \varepsilon)$ and $\hat{Z}(\pi, \varepsilon)$.
3: Find a move $v' \in \hat{Z}(\pi, \varepsilon)$, a neighbor $\pi' = \pi_{v'}$, a makespan $C' = C_{\max}(\pi')$ and modified
   $T'$ using the NSP. Modify the aspiration function $F(C) := \min\{F(C), C'\}$, $F(C') :=$
   $\min\{F(C'), C\}$. If $save = true$ then $Z^* := \hat{Z}(\pi, \varepsilon) \setminus \{v'\}$, $T^* = T$ and $save = false$.
   Set $\pi := \pi'$, $T := T'$ and $C := C'$.
4: If $C < C^*$ then set $\pi^* := \pi$, $C^* := C$, $ret := 0$, $save := true$ and go to **Step 2**.
5: If $ret < maxret$ go to **Step 2**.
6: If $Z^* \neq \emptyset$ and $iter < maxiter$ then set $\pi := \pi^*$, $\hat{Z}(\pi, \varepsilon) := Z^*$, $C := C^*$, $T := T^*$,
   $save =: true$, $ret := 1$ and go to **Step 3**; otherwise *STOP*.
```

Fig. 4 Pseudo-code for the tabu search algorithm

After this we will show the tabu search algorithm proposed by Nowicki and Smutnicki.

The algorithm can be launched from an arbitrary initial solution, but we start our search procedure from the permutation π given by the NEH heuristics. We set $\pi^* = \pi$, and start with an empty tabu list. In each iteration we find the set of moves Z and the representatives $\hat{Z}$. Applying the neighborhood searching procedure we select a move $v' \in \hat{Z}$, create a new tabu list and modify the aspiration function. If the new solution is better then the best one found so far, we update C^*.

If C^* does not decrease during *maxret* iterations then we jump back to π^* and continue searching with the stored representatives and tabu list Z^* and T^*. We have two stopping criteria: the maximum number of iterations (*maxiter*) performed, or Z^* is empty. We can summarize this algorithm into a pseudo-code in Fig. 4.

During the tests we used $maxiter = 30{,}000$, $maxt = 8$, $maxret = 500$.

4 Exact Optimization for the PFSP and PB-R-PFSP Models

One possible method to get the optimal solution of a PFSP is to formulate the PFSP as a mixed integer linear programming (MILP) problem and solve it by an appropriate software (such as CPLEX, GUROBI, and so on). The advantage of this approach compared to heuristic methods is that even if the software can not give optimal solution during the prescribed time limit for the running time it always gives a lower bound for the optimal value. This means that using MILP models we always know how far the given solution is from the optimal one. Since the PFSP is NP-hard for $M \geq 3$, the drawback of this approach is that only small or medium-sized problems can be solved optimally this way. However, taking into account the rapid growth of the performance of computers and softwares that can solve MILP models, one can expect that larger and larger problems become solvable this way.

4.1 MILP Models of the PFSP and R-PFSP

The MILP models of the PFSP [1, 8, 9, 12, 14–19, 22, 23] can be divided into two parts. The models of the Wagner family describe a permutation by giving for each position the job that is placed to this position. The models of the Manne family describe a permutation by giving for each par a jobs i and j whether i precedes j in the permutation or not. Empirical studies showed [18, 19] that the models of the Wagner family are superior to the models of the Manne family with regard to the required solution times. Therefore in [2, 3] based on MILP models of the Wagner family for the PFSP 3 new MILP models for the R-PFSP and PB-R-PFSP were introduced. In this section we first describe the R-TS2 model for the PFSP and then the R-TS2 model for the R-PFSP and the PB-R-TS2 model for PB-R-PFSP are introduced.

4.1.1 The TS2 Model for the PFSP

The TS2 model was presented in [19]. Let us denote by $C_{r,j}$ the completion time of the jth job of the order on machine r (so $C_{r,j}$ is a continuous variable for all (r,j) pairs $(1 \le r \le M,\ 1 \le j \le N)$. Let Z_{ij} be a binary variable for all (i,j) indeces $(1 \le i \le N,\ 1 \le j \le N)$. The value of Z_{ij} is equal to 1 if job i is placed to the jth place of the order, otherwise Z_{ij} is equal to 0. The constraints of the TS2-model imply that the following conditions are satisfied.

- Each job is assigned to exactly one place in the sequence.

$$\sum_{j=1}^{N} Z_{ij} = 1 \qquad 1 \le i \le N \tag{2}$$

- Each position in the sequence is filled with exactly one job

$$\sum_{i=1}^{N} Z_{ij} = 1 \qquad 1 \le j \le N \tag{3}$$

- The job in the $(j+1)$th position of the sequence can not finish on any machine until the job in the jth position of the sequence is finished on that machine and job in the $(j+1)$th position of the sequence is processed on that machine.

$$C_{rj} + \sum_{i=1}^{N} P_{ri} Z_{i,j+1} \le C_{r,j+1}, \qquad 1 \le r \le M,\ 1 \le j \le N-1 \tag{4}$$

- A job can not be finished on machine $r + 1$ until its finished on machine r and processed on machine $r + 1$.

$$C_{rj} + \sum_{i=1}^{N} P_{r+1,i} Z_{ij} \le C_{r+1,j}, \qquad 1 \le r \le M-1,\ 1 \le j \le N \tag{5}$$

- The first job of the order can not finish earlier on machine 1 than its duration time on machine 1.

$$\sum_{i=1}^{N} P_{1i} Z_{i1} \le C_{11} \tag{6}$$

- The makespan is the completion time of the last job of the sequence on the last machine.

$$C_{max} = C_{MN} \tag{7}$$

Hence the TS2 model can be summarized as

Minimize (7) Subject to: (2)–(6).

4.1.2 The R-TS2 Model for the R-PFSP

The Permutation with Repetition Flow Shop Problem (R-PFSP) is a special PFSP which contains jobs that have equal processing times on every machines. We say that jobs i and j have the same *type* if they have equal processing times on each machine. To describe a permutation in an R-PFSP it is enough to give for each position j of the sequence the type of the job that is placed to that position. This means that the TS2 model for the PFSP can be simplified to model R-PFSP-s. The following R-version of the TS2model, named R-TS2 were introduced by Hajba and Horváth in [2].

We will denote by T the number of different types and by n_t the number of jobs of type t $(1 \le t \le T)$. Let us denote the processing time of a job of type i on machine r by P'_{ri} and let Z_{ij} be a binary variable for all (i, j) indeces $(1 \le i \le T,\ 1 \le j \le N)$. The value of Z_{ij} is equal to 1 if a job of type i is placed to the jth place of the order, otherwise Z_{ij} is equal to 0. The constraints of the R-TS2-model imply that the following conditions are satisfied.

- There are n_i jobs in the sequence that are of type i.

$$\sum_{j=1}^{N} Z'_{ij} = 1 \qquad\qquad 1 \le i \le T \tag{8}$$

- Each position in the sequence is filled with exactly one type of job.

$$\sum_{i=1}^{T} Z'_{ij} = 1 \qquad 1 \le j \le N \tag{9}$$

- The job in the $(j+1)$th position of the sequence can not finish on any machine until the job in the jth position of the sequence is finished on that machine and job in the $(j+1)$th position of the sequence is processed on that machine.

$$C_{rj} + \sum_{i=1}^{T} P'_{ri} Z'_{i,j+1} \le C_{r,j+1}, \qquad 1 \le r \le M,\ 1 \le j \le N-1 \tag{10}$$

- A job can not be finished on machine $r+1$ until its finished on machine r and processed on machine $r+1$.

$$C_{rj} + \sum_{i=1}^{T} P'_{r+1,i} Z'_{ij} \le C_{r+1,j}, \qquad 1 \le r \le M-1,\ 1 \le j \le N \tag{11}$$

- The first job of the order can not finish earlier on machine 1 than its duration time on machine 1.

$$\sum_{i=1}^{T} P'_{1i} Z'_{i1} \le C_{11} \tag{12}$$

- The makespan is the completion time of the last job of the sequence on the last machine.

$$C_{max} = C_{MN} \tag{13}$$

The R-TS2 can be formalized as follows below.

Minimize (13) subject to (8)–(12).

Size Complexity of the TS2 and R-TS2 Models

The size complexity of the TS2 and R-TS2 models are presented in Table 1. The main difference is that the R-TS2 model contains much fewer binary variable than the TS2 model. The reason for it is that in the R-TS2 model we only have to give for each position the type of the job that is placed to that position (instead of giving for each position the job that is placed to that position). Finally the R-TS2 model contains less constraints than the TS2 model.

Table 1 Size complexity of the models

Model	Binary variable	Real variable	Constraints
TS2	N^2	$MN + 1$	$2MN - M + N + 1$
R-TS2	NT	$MN + 1$	$2MN - M + T + 1$

Note that N = number of jobs, T = number of types, M = number of machines

4.1.3 The PB-R-TS2 Model of the PB-R-PFSP

A *Permutation with Repetition Flow Shop Problem with Palettes and Buffer* (PB-R-PFSP) is a special R-PFSP in which only a limited number of jobs can wait between consecutive machines and the jobs are carried on palettes through the line and the number of palettes is bounded from above. We will denote by K the number palettes and by b_i the buffer size between machines i and $i + 1$.

The following PB-R-TS2 model was introduced in [3]. The model is the extension of the R-TS2 model. The constraints of the PB-R-TS2 model of the PB-R-PFSP ensure that the following two extra conditions hold.

- The number of the palettes is equal to K which implies that the jth job of the sequence cannot start its processing on the first machine until the $(j - K)$th job of the order is finished on the last machine.

$$C_{M,j-K} + \sum_{i=1}^{T} P'_{1,i} Z'_{ij} \leq C_{1j} \qquad K + 1 \leq j \leq N \tag{14}$$

- At most b_r jobs can wait in the buffer between machines r and $r + 1$. This condition is formulated as

$$C_{r+1,j-b_r-1} - \sum_{i=1}^{T} P'_{r+1,i} Z'_{i,j-b_r-1} + \sum_{i=1}^{T} P'_{r,i} Z'_{ij} \leq C_{rj} \tag{15}$$
$$1 \leq r \leq M - 1,\ 2 + b_r \leq j \leq N$$

The PB-R-TS2 model can be formalized as follows below.

Minimize (13) subject to (8)–(12), (14) and (15).

4.2 *Lower Bounds for the PFSP Model with Heuristic Methods*

For the production line managers it is very helpful to get a guaranteed lower bound to the optimum since it can be used to estimate how far the solution is from the optimal solution. The MILP solvers always provide the user with a guaranteed lower bound. However, for a high quality bound we need an excellent MILP solver which could not be available for the users for several reasons.

In this subsection we provide a pretty simple heuristic method to get a lower bound. To this aim we relax some of the assumptions of the problem such that the relaxed problem should be easily solvable. Then the optimal makespan of the relaxed problem is a valid lower bound for the original problem.

To get a lower bound we relax the constraints that every machine can process at most one job at a time. This idea was introduced by Lageweg in [7]. Suppose that M_k and M_l are different machines and relax the assumption that every other machine can process at most one job at a time (i.e. the other machine can process any number of jobs at a time). In the relaxed problem each job starts its processing on the first machine at time 0 and job i is ready for processing on machine M_k at time $\sum_{r=1}^{k-1} P_{ri}$. Similarly, when job i is finished on machine M_k then it takes $\sum_{r=k+1}^{l-1} P_{ri}$ time for it to be ready for processing on machine M_l and moreover after job i is finished on machine M_l then it takes $\sum_{r=l+1}^{m} P_{ri}$ time for it to process on machines $M_{l+1}, M_{l+2}, \cdots M_m$. This means that the resulting relaxed problem is a two-machine permutation flow shop problem with release dates, time lags and delivery times with objective function minimizing the makespan. This problem is denoted by $F2|r_j, l_j, q_j, prmu|C_{\max}$ where for all job j the release date r_j is given by

$$r_j = \begin{cases} \sum_{i=1}^{k-1} P_{ij} & \text{if } k > 1 \\ 0 & \text{if } k = 1, \end{cases}$$

the time lag l_j is given by

$$l_j = \begin{cases} \sum_{i=k+1}^{l-1} P_{ij} & \text{if } k < l-1 \\ 0 & \text{if } k = l-1, \end{cases}$$

and the delivery time q_j is given by

$$q_j = \begin{cases} \sum_{i=l+1}^{m} P_{ij} & \text{if } l < m \\ 0 & \text{if } l = m. \end{cases}$$

Here the release date r_j of job j means that we can not start processing job j on the first machine earlier than r_j. The time lag l_j of job j means that after job j finishes on the first machine it has to wait at least l_j time before we can start its processing on the second machine. The delivery time q_j of job j can be thought as the time of post processing j after it finishes on the second machine. Finally *prmu* states that only permutation schedules are allowed.

Since this is still a hard problem to solve we have to relax this problem too. Fixing the release dates and delivery times for every job j to $\min_{j\in J} r_j$ and $\min_{j\in J} q_j$ we get a two-machine PFSP with time lags $F2|l_j, prmu|C_{\max}$. It was shown by Rinnoy Kan

[6] that this problem can be solved in polynomial time by applying the Johnson's rule [4]. Hence for machine pairs (M_k, M_l) a valid lower bound for the original PFSP is

$$LB_{k,l} = \min_{j \in J} r_{kj} + C^{kl}_{\max} + \min_{j \in J} q_{kj}$$

where $C^{kl}_{\max}$ denotes the optimal makespan of the problem $F2|l_j, prmu|C_{\max}$. By running through all machine pairs a valid lower bound for the original PFSP is

$$LB = \max_{1 \le k < l} LB_{kl} \ .$$

5 Numerical Experiments

5.1 Test Problems

To analyze the performance of the heuristics and the MILP models on academic and industrial problems we generated two sets of test problems. The problems in both sets can be described with three parameters: the number of jobs N, the number of types T and the number of machines M. The number of machines was fixed $M = 50$, and we generated problems for $N = 100$ and $N = 200$. The values of T were $5, 10, 20, 50, 100$ for $N = 100$, and $5, 10, 20, 50, 100, 200$ for $N = 200$. For every triple we generated 5 problems. This procedure gave us $(5 + 6) \times 5 = 55$ processing time matrices for both sets. The two sets differed in the distribution of the values of the processing times.

In the first set we tried to create industrial problems. Hence we took a real problem from the industry, containing 11 types and 50 machines, and for each machine r we calculated the mean m_r and the standard deviation σ_r of the processing times on machine r. After that for each machine r the processing times on machine r were random numbers drawn from the normal distribution with parameters m_r and σ_r.

In the second set we created academic type problems with the procedure introduced by Taillard [20]. This means that the processing times were random integers coming from a uniform distribution with range $[0, 99]$.

5.2 Results

Three heuristics (NEH, TABU SEARCH and PACO) and one MILP model (R-TS2 model) were applied to solve the test problems. The formulations of the TS2 model were written in GAMS modeling language and solved by using CPLEX 12.3 on an Intel Xeon E31225 3.1 GHz personal computer equipped with 4 GB RAM. The

CPLEX options employed were mixed integer programming, parallel mode with four threads and a time limit of 600 s.

The relative gap of a solution is calculated by the formula

$$relative\ gap = 100 \cdot \frac{C_{best} - LB}{C_{best}} \tag{16}$$

where C_{best} is the makespan of the solution, and if CPLEX optimally solved the TS2 model in 10 min then LB is equal to the optimal makespan of the problem else LB is the lower bound of the optimum calculated the way it was described in Sect. 4.2. The average relative gaps are presented in Tables 2 and 3.

Table 2 Average relative gaps (in percentage) in the industrial problems

N	T	NEH	Tabu	Paco	R-TS2
100	5	1.14	0.52	0.18	0
100	10	0.52	0.1	0	0
100	20	1.37	0.74	0.47	0.35
100	50	1.49	0.24	0.25	1.54
100	100	1.16	0.21	0.26	*
200	5	0.86	0.34	0.06	0.05
200	10	0.24	0.01	0.16	0.03
200	20	0.49	0.45	0.11	1
200	50	0.35	0.13	0.06	*
200	100	0.49	0.09	0.11	*
200	200	0.52	0.38	0.25	*

N = number of jobs, T = number of types, number of machines $M = 50$

$*$At least in one instance no solution were found in 10 min

Table 3 Average relative gaps (in percentage) in the Talliard problems

N	T	NEH	Tabu	Paco	R-TS2
100	5	7.59	5.88	3.55	8.83
100	10	13.80	8.74	6.81	*
100	20	12.79	8.68	8.78	*
100	50	15.88	12.26	12.78	*
100	100	18.05	14.94	15.30	*
200	5	3.62	2.68	0.92	6.07
200	10	7.62	3.95	1.68	*
200	20	8.73	4.74	4.00	*
200	50	11.64	8.55	8.07	*
200	100	12.90	10.21	9.85	*
200	200	13.96	11.86	11.85	*

N = number of jobs, T = number of types, number of machines $M = 50$

$*$At least in one instance no solution were found in 10 min

In the industrial problems for problem sizes ($N = 100$, $T = 5$) and ($N = 100$, $T = 10$) the MILP model R-TS2 found the optimal solution in all five instances while for problem sizes ($N = 200$, $T = 5$) and ($N = 200$, $T = 10$) the MILP model R-TS2 found the optimal solution in four of the five instances. For larger problems (($N = 100$, $T = 100$), (N=200, T=50,100,200)) the R-TS2 model did not even find an initial solution in the 10-min time limit. Overall the Tabu search found the optimal solution in 18 of the 55 industrial problems while PACO found the optimal solution in 23 of the 55 instances.

In contrast to the industrial problems in the Talliard-like problems none of the four methods found the proven optimal solution in any of the 55 instances. It can be seen in Table 3 that for problem sizes with more than five types the R-TS2 MILP model did not even find an initial solution in any of the five instances. Furthermore the NEH, Tabu search and Paco heuristics have much larger relative gaps in the Taillard like problems than in the industrial like problems. The explanation of this phenomenon is that the algorithm described in Sect. 4.2 gives much stronger lower bounds in the industrial like problems than in the Taillard-like problems. Table 4 contains the lower bounds and optimal value of the 10 instances of industrial problems with problem size ($N = 100$, $T = 5$) and ($N = 100$, $T = 10$). It can be seen that in 4 of the 10 instances the lower bound calculated by the algorithm of Sect. 4.2 equals to the optimum of the problem.

5.2.1 Problems with Finite Buffer Sizes and Palettes

For the industrial problems with sizes ($N = 100$; $T = 5$) and ($N = 100$; $T = 10$) we solved the overall 10 instances with finite buffer sizes between the machines and finite number of palettes using the PB-R-TS2 model. The buffer sizes were chosen from real-world problems and the number of palettes was $K = 55$. We compared the optimal values of these PB-R-PFSPs with the makespan given by the Tabu search applied to the PFSP with infinite buffer sizes and infinite number of palettes (which can be considered as the relaxation of the PB-R-PFSP). It turned

Table 4 Comparing the lower bounds and the optimal values in the industrial problems

N	T	Lower bound	Optimum
100	5	48,721	48,721
		50,351	50,351
		52,527	52,814
		49,160	49,707
		50,621	50,789
100	10	49,484	49,607
		49,314	49,521
		47,632	47,632
		48,840	48,840
		49,556	49,627

out that in 9 of the 10 instances the solution given by Tabu search applied to the PFSP (with infinite buffer sizes and infinite number of palettes) was optimal for the PB-R-PFSP problem, too. In the remaining one case the Tabu search applied to the PFSP gave a solution to the PB-R-PFSP with relative gap 0.51.

6 Conclusions

We can summaries the lessons learnt with the observations as follows.

- Large-scale R-PFSPs in which the number of types is small and the machines are unbalanced can be solved efficiently by using MILP models and exact solvers.
- Both the Tabu Search and the PACO heuristics give good solutions (close to the optimum) for large-scale PFSPs containing unbalanced machines.
- For PFSPs containing unbalanced machines the two-machine relaxation of the problem gives lower bound close to the optimum.
- For industrial like PB-R-PFSPs (R-PFSPs with palettes and limited buffers) one may compute a good (initial) solution of the problem by omitting the palettes and the buffer sizes between the machines (i.e. setting the number of palettes and buffer sizes between consecutive machines to infinite) and solving the relaxed R-PFSP.

These scientific results can effectively be used in the digital factory environment. We consider a production line with planning and shop floor software tools of digital factory to collect, analyse and process real life manufacturing data. With integration of the solving methods investigated by our work into the planning and scheduling process of the digital factory, we can produce effective production schedule supporting the end-to-end digital integration goal of the digital factory. This integrated production schedule can be the one of the basics for the smart and networked Industry4.0 production environment.

References

1. Baker, K.R.: Introduction to Sequencing and Scheduling. Wiley, New York (1974)
2. Hajba, T., Horváth, Z.: New effective MILP models for PFSPs arising from real applications. Cent. J. Oper. Res. **21**, 729–744 (2012)
3. Hajba, T., Horváth, Z.: MILP models for the optimization of real production lines. Cent. J. Oper. Res. **23**, 899–912 (2015)
4. Johnson, S.M., Optimal two -and three stage production schedules with setup times included. Nav. Res. Logist. Q. **1**(1), 61–68 (1954)
5. Jósvai, J.: Production process modeling and planning with simulation method, mounting process optimisation. In: The International Conference on Modeling and Applied Simulation. Universidad de La Laguna, 23–25 September 2009, pp. 240–245 (2009)
6. Kan, A.H.G.R.: Machine Scheduling Problems: Classifications, Complexity and Computation. Nijhoff, The Hague (1976)

7. Lageweg, B.J., Lenstra, J.K., Kan, A.H.G.R.: A general bounding scheme for the permutation flow-shop problem. Oper. Res. **26**, 53–67 (1978)
8. Liao, C.L., You, C.T.: An improved formulation for the job-shop scheduling problem. J. Oper. Res. Soc. **43**, 1047–1054 (1992)
9. Manne, A.S.: On the job-shop scheduling problem. Oper. Res. **8**, 219–223 (1960)
10. Nawaz, M., Enscore, E.E., Ham, I.: A heuristic algorithm for the m-machine n-job flow-shop sequencing problem. Omega **11**, 91–95 (1983)
11. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the permutation flow-shop problem. Eur. J. Oper. Res. **91**, 160–175 (1996)
12. Pan, C.H.: A study of integer programming formulations for scheduling problems. Int. J. Sys. Sci. **28**, 33–41 (1997)
13. Rajendran, C., Ziegler, H.: Ant-colony algorithms for permutation flowshop scheduling to minimize total makespan/total flowtime of jobs. Eur. J. Oper. Res. **155**, 426–438 (2004)
14. Stafford, E.F.: On the development of a mixed-integer linear programming model for the flowshop sequencing problem. J. Oper. Res. Soc. **39**, 1163–1174 (1988)
15. Stafford, E.F., Tseng, F.T.: On the Strikar-Gosh MILP model for the $N \times M$ SDST flowshop problem. Int. J. Prod. Res. **28**, 1817–1830 (1990)
16. Stafford, E.F., Tseng, F.T.: Two models for a family of flowshop sequencing problems. Eur. J. Opr. Res. **142**, 282–293 (2002)
17. Stafford, E.F., Tseng, F.T.: New MILP models for the permutation flowshop problem. J. Oper. Res. Soc. **59**, 1373–1386 (2008)
18. Stafford, E.F., Tseng, F.T., Gupta, N.D.: An empirical anlysis of integer programming formulations for the permutation flowshop. Omega **32**, 285–293 (2004)
19. Stafford, E.F., Tseng, F.T., Gupta, N.D.: Comparative evaluation of the MILP Flowshop models. J. Opr. Res. Soc. **56**, 88–101 (2005)
20. Taillard, E.: Benchmarks for basic scheduling problems. Eur. J. Oper. Res. **64**, 278–285 (1993)
21. VDI: Digital Factory Fundamentals. VDI 4499 Guideline, Düsseldorf (2008)
22. Wagner, H.M.: An integer linear-programming model for machine scheduling. Nav. Res. Log. Q. **6**, 131–140 (1959)
23. Wilson, J.M.: Alternative formulations of a flow-shop scheduling problem. J. Oper. Res. Soc. **40**, 395–399 (1989)