CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	The hybrid flexible flowshop with transportation times
Authors	Armstrong, Eddie;Garraffa, Michele;O'Sullivan, Barry;Simonis, Helmut
Publication date	2021-10-15
Original Citation	Armstrong, E., Garraffa, M., O'Sullivan, B. and Simonis, H. (2021) 'The hybrid flexible flowshop with transportation times', 27th International Conference on Principles and Practice of Constraint Programming (CP 2021), Montpellier, France, 25 - 29 October, 16 (18pp). doi: 10.4230/LIPIcs.CP.2021.16
Type of publication	Conference item
Link to publisher's version	https://zenodo.org/record/5168966 - 10.4230/LIPIcs.CP.2021.16
Rights	© 2021, Eddie Armstrong, Michele Garraffa, Barry O'Sullivan, and Helmut Simonis. Licensed under Creative Commons License CC-BY 4.0 - https://creativecommons.org/licenses/by/4.0/
Download date	2026-11-16 07:34:22
Item downloaded from	https://hdl.handle.net/10468/12116

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

The Hybrid Flexible Flowshop with Transportation Times

Eddie Armstrong
Johnson & Johnson Research Centre, Limerick, Ireland

Michele Garraffa ✉
Confirm SFI Research Centre for Smart Manufacturing, Limerick, Ireland
School of Computer Science, University College Cork, Ireland

Barry O'Sullivan ✉
Confirm SFI Research Centre for Smart Manufacturing, Limerick, Ireland
School of Computer Science, University College Cork, Ireland

Helmut Simonis ✉
Confirm SFI Research Centre for Smart Manufacturing, Limerick, Ireland
School of Computer Science, University College Cork, Ireland

Abstract

This paper presents the hybrid, flexible flowshop problem with transportation times between stages, which is an extension of an existing scheduling problem that is well-studied in the literature. We explore different models for the problem with Constraint Programming, MILP, and local search, and compare them on generated benchmark problems that reflect the problem of the industrial partner. We then study two different factory layout design problems, and use the optimization tool to understand the impact of the design choices on the solution quality.

2012 ACM Subject Classification Software and its engineering → Constraints; Computing methodologies → Planning and scheduling

Keywords and phrases Constraint Programming, scheduling, hybrid flowshop

Digital Object Identifier 10.4230/LIPIcs.CP.2021.16

Supplementary Material *Dataset*: `https://zenodo.org/record/5168966`

Funding This project has received funding from a research grant from Science Foundation Ireland (SFI) under Grant Number 16/RC/3918.

1 Introduction

An important category of production systems comprises hybrid flowshop environments, where the input jobs need to be processed by different production stages, and multiple identical machines are available at each stage. In this context, the problem consisting of finding schedules of minimum length is a well-known NP-hard combinatorial problem, known as Hybrid Flowshop Scheduling (HFS) [36]. An extension of this problem, which has been the object of recent studies, is the case where a subset of the jobs are required to skip some of the stages. This problem is usually denoted as Hybrid Flexible Flowshop Scheduling (HFF) [33]. Since many decades, the solution of HFS/HFFs problems has been fundamental to increase efficiency of many real world manufacturing systems, related to a wide variety of production areas like electronics [25, 10], glass [32], textile [22], packaging [1], etc.

Recently, an increasing number of production systems are structured such that the machines are located in different buildings of the same production site, or even at separate sites. This may occur for multiple reasons, such as the risk of contamination of materials during some production stages. As a consequence, the machines may be located at diverse locations, requiring a non-negligible time to transport a job from one stage to the next


© Eddie Armstrong, Michele Garraffa, Barry O'Sullivan, and Helmut Simonis;
licensed under Creative Commons License CC-BY 4.0



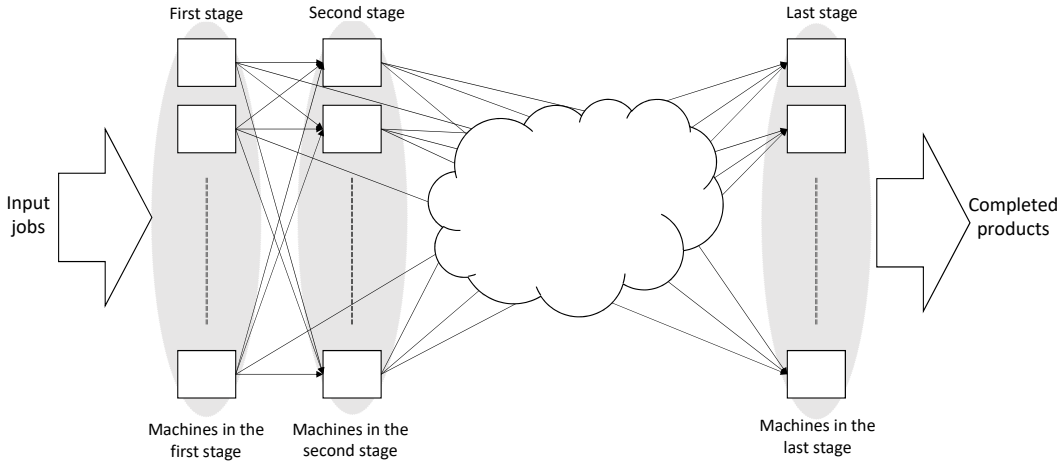


Figure 1 Logical structure of a hybrid flowshop.

one. These transportation times are known as inter-stage transportation times, in order to make a distinction with the ones required to deliver the products or acquire raw materials. Moreover, these transportation times are usually machine-dependent, in the sense that they are proportional to the physical distance between the two machines involved. In this work, we introduce the Hybrid Flexible Flowshop with Transportation Times (HFFTT), as an extension of the HFF where inter-stage, machine-dependent transportation times are taken into account.

The diagram depicted in Figure 1 summarizes the logical structure of a HFFTT problem, where multiple machines are associated with each stage and each arrow connecting a couple of machines (m_a, m_b) is indicating the fact that a certain time is needed to transport a job from m_a to m_b. The main characteristics of the HFFTT are:

- An arbitrary number of production stages are considered.
- Each stage has multiple identical machines working in parallel.
- All jobs are following the same production flow, with the exception that some jobs may skip some stages.
- Each job has a fixed processing time for each production stage, independent of the machine used.
- Transportation times are required to move a job from one machine of a stage to another machine of the following stage.
- The vehicles transporting the jobs are not taken into consideration, since they are assumed to be enough to avoid any waiting delay. This is justified by the fact that current production lines are fully automated and the schedules are defined a priori, so that the transporters can move the jobs in time with no delays.
- The objective function value is the maximum completion time, i.e. the makespan of the schedule. The makespan is a measure of the system efficiency, it is considered in the standard definition of the HFS and widely used in scheduling.

Other assumptions are that any machine can process only one operation at a time with no preemption, while all jobs can be processed by only one machine at a time. All tasks are released at time 0, setup times are negligible or can be aggregated to the jobs' processing times. The capacity of the buffers between stages is unlimited, there are no additional resources needed to run a job on a machine and, finally, the processing times and the transportation times are deterministic parameters that are known a priori.

Surprisingly, there are no previous studies focusing exactly on this problem definition, to the best of the authors' knowledge. However, studies tackling similar problems are discussed in Section 2. The HFFTT can be indicated with $HFFm|tt|C_{max}$ by means of the three field notation, which is the most common notation used to identify scheduling problems [21].

Our motivation for studying the HFFTT lies in a real-world practical application arising in fully-automated pharmaceutical manufacturing, where millions of lots are produced every year. One of the authors, working for the research and development team of the firm, provided requirements that we reproduced when we defined the problem and generated our instances. The aim of this work is twofold. On the one hand, increasing the throughput of a production site by a small percentage (say 1%) brings a significant economic benefit, due to the very large volume of production. We formulate the problem in terms of Constraint Programming (CP), Mixed Integer Linear Programming (MILP), and Local Search in order to develop optimization approaches based on off-the-shelf solvers and test them on these realistic scenarios.

On the other hand, the introduction of transportation times leads to another research question, which is of particular interest for the industry partner. It consists of determining the positions to be assigned to the machines in order to achieve a high system efficiency. Such problem belongs to a wide category of optimization problems known as facility layout [14, 4], with hundreds of papers published in recent years [41]. We address this challenge by using CP-based scheduling to quantify the performances of different layouts alternatives, hence providing insights for driving business-related decisions in the real world scenario.

The rest of this document is arranged as follows. Section 2 collocates the problem in the state of the art, by analyzing similar scheduling problems studied in the past.

The problem admits multiple alternative formulations, which are presented and discussed in Section 3. Section 3.2 presents a CP formulation for the HFTT. Section 3.3 discusses how to extend the MILP formulations available in the state of the art for the HFF, in order to take into account transportation times. Section 3.4 provides a local search procedure, which is used to compute good feasible solutions and initialize the solvers. Section 4 describes the computational experiments that we conducted. These experiments are performed on realistic instances, whose structure comes from the real world scenarios discussed with the pharmaceutical firm, as mentioned before. The instance generation and the scenarios are discussed in Section 4.1 and Section 4.2, respectively. The computational results, describing the performance of the proposed models, are presented in Section 4.3. Section 5 concludes the paper.

2 Related Work

As discussed in the introduction, the HFFTT is an extension of the HFF, which is a HFS problem where certain jobs are required to skip some production stages. The HFS is a well-studied problem, whose literature includes many problem variants. It is NP-hard in its general form and in most of its restrictions. As an example, the case where there is one machine per stage is a flowshop problem, which is known to be NP-hard [20]. Some polynomially solvable special cases are arising when some special properties and precedence relationships are verified [13].

The reader is referred to [36] for an in-depth survey, which describes papers published up to 2010. Another, more recent review [40] is covering the works published from 2010 to 2020, with a special focus on metaheuristic approaches. The literature of the HFF is less rich with respect to the HFS. Four different MILP formulations are proposed in [33]

and assessed computationally. A general discussion about the impact of including different realistic constraints is given in [37]. The most recent metaheuristic developed for the HFF is an iterated greedy approach based on a hyperheuristic [8]. Finally, the use of CP approaches in HFS/HFF problems is tackled by [26] and [45]. [26] considers a HFS with multiprocessor tasks and proposes a memetic algorithm, which uses a CP-based branch and bound algorithm as a local search engine. [45] studies a HFF problem inspired by a real-world application, formulates it in terms of CP and solves it utilizing Gecode.

The HFFTT is more general than the HFF and the HFS, hence it is NP-Hard as its special cases. The HFFTT includes inter-stage, machine-dependent transportation times, but they are not the only type of transportation times studied by the scheduling community. In fact, the first scheduling paper taking transportation into account is [31], which dates back to 1980, and considers purely job-dependent transportation times. The authors consider a two-machine flowshop problem with a sufficient number of transporters, so that whenever a job is completed on the first machine it can be transported to the second machine immediately. Another paper [29] focuses purely on the job transportation in a HFS with two stages. Other HFSs with some form of job transportation have been considered in other works, the reader is referred to Section 2 of [30] for a recent and detailed review about this topic. A recent paper [44] studies two problems where there is only one transporter in a HFS with two stages. These problems are very close to the HFFTT, with the two main differences being the fact that the transporter is seen as a further resource to be shared among the jobs and the number of stages is restricted to 2. Another similar problem to the one defined in this paper is presented in [30]. The main differences are that buffers are assumed to be finite, there is only one transporter per stage, and the jobs are not skipping any stage. [42] deals with another HFS with transportation times, differing from our problem because it considers a limited number of transporters with finite capacities and because the transportation times are purely stage-dependent, since they do not depend on the machine positions. [15] studies a single-transporter HFS with machine eligibility where the transportation times are not machine-dependent. The same authors propose in [16] and [17] a MILP model and a metaheuristic for a cyclic HFS, with multiple robots devoted to perform the transportation operations. The main difference with the HFFTT is the fact that they deal with a cyclic problem and they use as an objective a throughput rate maximization, that is the number of parts completed in the cycle time. [24] presents a two-centers hybrid flowshop, where the machines of each stage are aggregated in centers. Contrarily to our problem, here only two stages are considered and the transportation times are purely job-dependent.

Other recent works are dealing with scheduling problems with transportation times, which consider different types of production environment, such as a groupshop [3], a flexible jobshop [35] and a batch-flowshop [6].

A large number of studies have been devoted to the design of facility layouts in order to improve the system productivity and/or reduce costs. However, we did not find any study related to designing facility layouts for HFS. A survey about facility layout problems is provided in [4], where a survey of the mathematical optimization approaches is presented. These approaches are based on solving a variety of optimization problems, that are expressed in terms of MILP or conic programming. In some cases, simulation and optimization interoperate to find the best layouts. [5] presents a simulation-based genetic algorithm encoding each single layout as a string and evaluating the performances of each layout by means of simulation. [19] discusses the case where simulation-based optimization is used for facility layout optimization under uncertainty. CP has been adopted to solve different types of layout problems. An example is [39], where the authors show the effectiveness of CP for

designing manufacturing cells. Another related study is [43], where the authors propose a CP approach for multi-objective wind farm layout optimization. To the best of the authors' knowledge, there are no previous studies using constraint-based scheduling for the design of facility layouts.

3 Models

We now present different models for the selected problem, using various solver technologies. We only explain one CP model in full detail due to space considerations.

3.1 Notation

In this section, we introduce the notation used in the different problem formulations. Let J be a set of jobs, M be a set of machines and S be a set of production stages. Each job $j \in J$ requires a processing time $p_{j,s}$ to complete a stage $s \in S$. The set of all jobs that need to be processed in stage $s \in S$ is indicated with $J_s \subseteq J$, while the stage before $s \in S$ where a job $j \in J$ needs to be processed is indicated with $prec(s, j)$. The transportation time $\delta_{m,m'}$ is required to move a job from machine $m \in M_s$ to $m' \in M_{s'}$, where $s, s' \in S$ and s' is successive to s in the production system. By following the general concept used in CP for scheduling problem including the HFS [44], we define a task as the execution of a job at some production stage. Hence, one task per stage used is associated to each job. We indicate the set of all tasks with T. Finally, the set of all the tasks associated with a certain stage $s \in S$ is denoted as T_s and the set of all tasks associated with a certain job $j \in J$ is denoted as T_j.

3.2 CP

For this type of scheduling problem with precedence and machine choice there are two main choices of modelling in Constraint Programming. The first approach uses a two-dimensional `diffn` [7] constraint, where the first (x) dimension represents time, and the second (y) dimension represents the machine allocated to a task. Each task is represented by a rectangle whose width is the duration of the task, and the height is one, while the x start position represents the start time, and the y value indicates the machines on which it is run. This constraint states that tasks are either scheduled on different machines, or do not overlap in time. This type of model goes back to CHIP [12], and can be expressed in many CP systems, we use it in our MiniZinc [34] and SICStus Prolog [9] formulations. The transport time between two consecutive tasks of the same job is expressed by a `table` constraint, which links the machine variables of the tasks to the transport time, by enumerating all possible combinations. This provides maximum flexibility in expressing the transport time constraint, while allowing for efficient, domain consistent constraint propagation.

The second approach uses optional interval task variables [28] to represent the machine choices. For each task and each machine on which the task can be scheduled, there is an optional interval variable which represents the choice whether the task is run on this machine. Exactly one of the optional variables for a task must be selected (using an `alternative` constraint), while all optional tasks on the same machine are constrained to be non-overlapping. This model is required by IBM's CPOptimizer [28], but can also be expressed with MiniZinc [1]. Expressing the transport time between tasks is more complex,

[1] Unfortunately, the MiniZinc model with interval variables is not competitive solving even the smallest problem instances, and is not used in the experiments.

IBM provides a specific constraint to deal with sequence dependent setup times on the same machine, but not for expressing transport times between consecutive tasks of the same job on different machines. We have to link additional machine assignment variables via `presenceOf` constraints to the optional tasks, and use `allowedAssignments` constraints to handle the transport times, linking the machine assignment variables and the transport time of consecutive tasks. The default search in CPOptimizer automatically switches between different search methods during the search phase, and is, like our custom SICStus Prolog search described below, independent of the selected time resolution.

3.2.1 Diffn Model

We now describe the first constraint model of the problem in full detail, using MiniZinc as the description language. We first define the constants to use, and the arrays that hold the input data. The set T is the index-set of all tasks, for every task we have the fixed duration, the stage to which it belongs, and the set of machines on which it can run. We also have a set P of precedence relations between tasks that will be used to set up precedence constraints. Note that this formulation is more generic than the flowshop problem, and allows to handle other generic scheduling problems as well.

```
include "globals.mzn";
% constant definitions
  set of int:T; % tasks
  set of int:P; % precedences between tasks
  set of int:M; % machines
  set of int:S; % stages
  set of int:tuples; % transport time table size
  int:lb; % lower bound on the cost
  int:ub; % upper bound on the cost
% constant arrays
  array[T] of int:duration;
  array[T] of int:stage;
  array[P,1..2] of int:prec; % precedence pairs
  array[T] of set of int:machines; % machine domains
  array[tuples,1..3] of int:transportTime;
% data file
  include "data/data.dzn";
```

For every task, we have three variables, the start of the task, the machine on which it is run, and the travel time from the machine of the previous task to its own machine. In addition, we use the variable *cmax* as the overall cost.

```
% variables
  array[T] of var 0..ub:start;
  array[T] of var M:machine;
  array[T] of var 0..ub: travel;
  var lb..ub:cmax;
```

We have five constraint types in the model. The first links the makespan variable to the end of all tasks. The second handles the `precedence` constraints within a job, linking the first to the second task by the duration of the first task, and the required travel time between the machines used. The third constraint expresses that each task can only be run on a specific subset of the machines. The fourth constraint is the `diffn` constraint for each stage, representing the tasks as rectangles. Note that a single `diffn` constraint for all tasks would suffice, but multiple constraints with disjoint subsets of tasks lead to better performance. In

addition, a `cumulative` [2] constraint linking the tasks and the number of available machines of each stage may lead to additional propagation in some systems. The final constraint generates the `table` constraint for the travel times, linking the machines assigned and the time required, via a single table of all allowed travel time combinations.

```
% constraints
  constraint forall (i in T)
    (cmax >= start[i]+duration[i]);
  constraint forall (p in P)
    (start[prec[p,2]] >= start[prec[p,1]]+
                         duration[prec[p,1]]+
                         travel[prec[p,2]]);
  constraint forall(i in T) (machine[i] in machines[i]);
  constraint forall(s in S)
    (diffn([start[i]|i in T where stage[i]=s],
           [machine[i]|i in T  where stage[i] = s],
           [duration[i]|i in T where stage[i] = s],
           [1|i in T where stage[i] = s]));
  constraint forall (p in P)
    (table([machine[prec[p,1]],
            machine[prec[p,2]],
            travel[prec[p,2]]],transportTime));
% solve
  solve minimize cmax;
```

With the free search of Chuffed [11] we find solutions for smaller problem sizes, but it initially finds solutions with high costs. When giving a conservative value for *ub*, Chuffed spends most of its time decreasing the cost value one by one from that initial high value. We need an alternative to find good solutions quickly, the strategy chosen will assign tasks from left to right, fixing start and machine of the task together. Note that other strategies, like assigning all start values before the machine assignment or vice versa, do not lead to viable solutions, as the machine assignment determines the transport time between tasks, and therefore has a direct effect on the start times. Conversely, if we fix the start times to their smallest value, we remove too much flexibility in the machine assignment to complete the assignment.

We use the `priority_search` annotation of Chuffed [18] to assign start and machine together. We select a task with the smallest value in the domain of its start, and fix the start and then the machine of the task to values in their domain. Note that the sequence here is important to place tasks to their left-most position.

```
include "chuffed.mzn";

  solve ::priority_search(start,
    [int_search([start[i],machine[i]],input_order,indomain_min)| i
      in  T],
     smallest,complete) minimize cmax;
```

For the search in the SICStus Prolog [9] model, which uses the same constraints, we define a custom search routine that iteratively increases the start time considered, and for each time point, decides if the remaining tasks should start at this time or not. Once the start is fixed, we try to find a machine on which the task will be run. The search times out after a limited amount of backtracking, and restarts with a different initial task ordering, until either the lower bound is reached, or the global time budget is exceeded.

3.3 MILP

The HFFTT admits multiple MILP formulations. It is possible to extend the HFF formulations in [33] so that they include inter-stage transportation times. In fact, Model 1, Model 3 and Model 4 of [33] can be extended by including continuous time variables $\Delta_{j,s} \in \mathbb{R}_+$, representing the transportation time required by job $j \in J$ to reach stage $s \in S$. Analogously, Model 2 of [33] can be adapted by including continuous time variables $\Delta_{j,s,m} \in \mathbb{R}_+$, representing the transportation time required by a job $j \in J$ to reach a machine $m \in M$, belonging to stage $s \in S$. The variables $\Delta_{j,s}$ and $\Delta_{j,s,m}$ need to be linked with the real transportation times $\delta_{m,m'}$. This can be done by imposing that they are greater than or equal to $\delta_{m,m'}$, in the case the job $j \in J$ is processed by the machines $m, m' \in M$, which is determined by the binary variables related to the machines' assignments. Finally, the constraints linking the jobs' start and completion times at the different stages need to be modified in order to include the transportation times.

We implemented all four adapted models and performed a set of preliminary tests, where Model 4 yielded the best results. Hence, we focus on the modified version of Model 4. To simplify notation, let us define a dummy stage s_0 preceding all other stages, with no machines $M_{s_0} = \emptyset$ and performed by all the jobs with 0 duration. The decision variables are:

- $x_{i,j,s} \in \{0,1\}$ equal to 1 if job $i \in J_s$ comes after job $j \in J_s$ at a certain stage $s \in S$, 0 otherwise.
- $y_{j,s,m} \in \{0,1\}$ equal to 1 if the job $j \in J_s$ is processed by machine $m \in M_s$ at stage $s \in S \cup \{s_0\}$, 0 otherwise.
- $\Delta_{j,s} \in \mathbb{R}_+$ is the inter-stage transportation time required by job $j \in J_s$ to reach stage $s \in S$ from the previous one.
- $C_{j,s} \in \mathbb{R}_+$ is the completion time of job $j \in J_s$ in stage $s \in S \cup \{s_0\}$.
- $C_{max} \in \mathbb{R}_+$ represents the makespan.

The following formulation of the HFFTT holds:

$$\min C_{max} \tag{1}$$

subject to:

$$C_{j,s_0} = 0 \qquad \forall j \in J \tag{2}$$

$$y_{j,s_0,m} = 0 \qquad \forall j \in J, m \in M \tag{3}$$

$$\sum_{m \in M_s} y_{j,s,m} = 1 \qquad \forall j \in J_s, s \in S \tag{4}$$

$$C_{j,s} \geq C_{j,prec(s,j)} + p_{j,s} + \Delta_{j,s} \qquad \forall j \in J_s, s \in S \tag{5}$$

$$C_{j,s} \geq C_{i,s} + p_{j,s} - \Lambda(3 - x_{j,i,s} - y_{j,s,m} - y_{i,s,m}) \qquad \forall i \neq j \in J_s, s \in S, m \in M_s \tag{6}$$

$$C_{i,s} \geq C_{j,s} + p_{i,s} - \Lambda x_{j,i,s} - \Lambda(2 - y_{j,s,m} - y_{i,s,m}) \qquad \forall i \neq j \in J_s, s \in S, m \in M_s \tag{7}$$

$$C_{max} \geq C_{j,s} \qquad \forall j \in J_s, s \in S \tag{8}$$

$$\Delta_{j,s} \geq \delta_{m',m}(y_{j,s,m} + y_{j,prec(s,j),m'} - 1) \qquad \forall m' \in M_{prec(s,j)}, m \in M_s, j \in J_s, s \in S \tag{9}$$

Please note that the parameter $\Lambda \in \mathbb{R}_+$ indicates a large enough real constant. Furthermore, $prec(s,j)$ indicates the stage where $j \in J$ was processed before stage $s \in S$. The objective is to minimize the makespan (1). Constraints (2) and (3) are related to the dummy stage, which is the first stage for all the jobs and requires no time for its completion and no machine usage. Constraints (4) ensure each job $j \in J_s$ is processed by exactly one machine at each stage $s \in S$. Constraints (5) link the completion time of a job in a stage with the

completion time of the same job in the previous stage. Please note that such constraints reduce to $C_{j,s} \geq p_{j,s}$ in the case where $prec(s,j) = s_0$, since $C_{j,s_0} = 0$ and no time is required to reach the first stage after s_0 ($\Delta_{j,s} = 0$). Constraints (6) and (7) determine the completion times of the jobs in a stage, by taking into account the precedence and the machines availability. Constraints (7) enforce that C_{max} is greater than or equal to the completion time of all the jobs at any stage. Constraints (8) link the values of the transportation time variables with the time needed to transport each job in order to achieve a machine at a certain stage. Please note that $\Delta_{j,s}$ is not involved in any constraint if $prec(s,j) = s_0$, since $M_{s_0} = \emptyset$.

3.4 Dispatch Rule and Local Search

In order to evaluate the results of our models in a broader context, we also introduced two incomplete methods of finding solutions. The Dispatch model takes the jobs in some given order, and schedules each job in sequence, placing the tasks at the first available time, on the first available machine. This can be done efficiently by keeping track of the allocated time on each machine, and by considering the required precedences and travel time when looking for the next available machine. In our system, we randomly permute the job order, and rerun the Dispatch model repeatedly, keeping improved solutions. We stop the search when we either reach the given lower bound, or hit a timeout limit.

We've added a Local Search variant of this model, by allowing swaps of two jobs and insertion of jobs into the job sequence at a different place. When we run out of possible moves, we restart the search with a new permuted job order, and continue until we reach the lower bound or the time limit.

Note that neither variant uses constraints, or estimates about the achievable makespan given a partial solution, while both rely on the simplicity of the method to evaluate many possible job sequences.

3.5 Lower Bound Calculation

A good lower bound on the makespan can be useful to stop the search having found an optimal solution reaching that lower bound, or to understand the remaining optimality gap of the instances tested. We have extended the bounds in [23] to deal with the transportation time between stages, while adding one more bound based on the job type distribution. The following gives an intuition of the bounds used.

An obvious lower bound on the makespan is the minimum duration of any job to be scheduled. This consists of the sum of the duration of the tasks of the different stages, and an estimate of the transport time required between consecutive tasks. The easiest approximation of this is using the smallest transport time between each pair of stages as an estimate, a more complex model can use Dynamic Programming to find the shortest path from start to finish over all machines. This job related lower bound is quite strong if the number of jobs is small, but will become useless as the number of jobs in the schedule increases. Note that the constraint models will infer the first version of this lower bound from the precedence and travel time constraints, this then becomes the initial lower bound of the makespan variable.

For larger instances we can do better than this, by considering a stage based bound. If we consider any set of tasks belonging to the same stage, we can compute a lower bound on the overall makespan as the minimum time to start one of the tasks of the set (`minStart`), the time to process all tasks in the set on the machines for that stage, and the minimum time that is required from the end of a task in the set to the overall project end (`minEnd`).

While it is clearly not possible to evaluate this bound for every subset of the tasks for each of the stages, we can group tasks by their `minStart` and `minEnd` values.

We still have to define how to estimate the time required to process all tasks of a set on the machines of the selected stage. Ideally, we can solve a separate bin-packing constraint to find the best solution, but that itself is a hard problem, and therefore not really appropriate for a lower bound. We use three lower bounds on the bin packing problem instead:

- The maximum duration of any task in the set is a bound on the time required to process all tasks of the set.
- The total duration of all tasks in the set, divided by the number of available machines, is another lower bound.
- If we order the tasks of the stage by decreasing duration, then the sum of the kth and $k+1$th element of the task list is a lower bound on the total duration, if the stage has k machines.

Finally, we consider the most common product types, for which there are multiple jobs in the order set. When minimizing the overall makespan, these jobs are identical, and in our test scenarios, quite common. The most common product accounts for up to 25% of all orders. It is worthwhile to obtain a lower bound for these jobs. We consider the sub problem of scheduling only a single product type, in that case we do not have to allow all permutations of the jobs, since they are all identical. The solution to this simpler problem is a valid lower bound for the overall makespan. In the selected scenarios, this bound very rarely dominates the other bounds, but it becomes useful when the power-law distribution of product types is changed to consider fewer, more common products.

Given these bounds, we still observe rather wide optimality gaps (up to 20%) for some medium sized instances. Further improvements could consider:

- The total surface estimate assumes that there are no other tasks that overlap the selected period, but there may be longer tasks starting earlier that are still active at this time. A study of energetic reasoning for the cumulative constraint [2] may be appropriate to understand if we can apply some of the reasoning here.
- Some tasks may be required to achieve the shortest `minStart` and shortest `minEnd` time at the same time. But it may not be possible to use the same tasks for both roles, so that more time either at the start or the end is required.
- There can be a conflict between lower bounds for different stages, which require different sets of jobs to be scheduled early to avoid loosing too much production capacity at the start or end of the schedule. By considering multiple, non-consecutive bottleneck stages at the same time we may be able to improve the lower bounds significantly, without having to solve hard combinatorial sub-problems to optimality.

4 Computational Experiments

In this section we describe how we generate sample problem instances based on assumptions on the manufacturing process, which design alternatives we want to explore, and which results we achieve. The results allow us to understand the impact of the different models and solving technologies, and how the factory design choices affect the overall solution quality for different solvers. A final step, not presented here, then integrates our results with a financial analysis of the costs and risks of the design alternatives, to help the decision making process for the industrial partner.

4.1 Instance Generation

We have built an instance generator for the problem, which produces fully parametrized instances of the problem. The generated test cases will be made available as an appendix in the final publication. In our experiments we selected some parameters to be typical of the situation of the industrial partner, while allowing for flexibility in other parameters. All experiments shown use 8 stages, where stage 4 and 8 can be skipped with 10% probability. We allow 10 machines at each stage, and consider two scenarios, one, where each stage has the same number of machines, and one where the number of machines per stage is adjusted downwards based on the total workload for each stage. For space reasons we only report results for the uniform machine number case. We vary the number of jobs between 20 and 400, noticing that the larger instances may require large amounts of memory for some solvers.

For machines within the same building, we consider a lane model (see Figure 2a), where machines are arranged in a grid pattern, all machines of the same stage being placed in the same column, and machines in the same row requiring the smallest transport time between them. Transport time increases with the difference of the lanes in which the machines are placed. Note that this layout design is only one of the choices in the instance generator, and does not directly affect the models, which use a table of transport times between machines.

Each job belongs to a given job type, jobs of the same type have the same manufacturing constraints. We select the job type based on a discrete power law (Zipf) distribution, with exponent 1.05. This means that some products are more common than others, but many products only have a single job in the order set. This choice corresponds to the semi-custom production model for which the factory is being designed. The task duration is randomly chosen within a range of 1 to 10, while the transport times vary from 1 to 9, and an inter-building transport requires 10 time units. The instances generated are available at [38].

4.2 Scenarios

We use our models to answer two design questions, one designing the transport inside a single factory, the other dealing with the potential split of the production between sites.

4.2.1 Reach of Transport between Stages

This problem considers the lane based layout inside a single factory as a starting point. The question is how far the transport should reach between lanes. If each job stays within its initial lane during production, then transport requirements are minimal, but there is little flexibility in the scheduling. If, on the other hand, we allow transport between any two lanes, a lot of infrastructure needs to be provided, which may or may not pay off in improved solution quality. The result will be an understanding of how transport flexibility affects solution quality, and if using different solvers for the experiments lead to different results.

4.2.2 Single Factory vs. Multiple Locations

Another important design question is whether it is better to have a single facility where production is concentrated, or whether (for example) two locations should be selected. We consider five alternatives:

1. We use a single facility where all products are made. Transport is between lanes of consecutive stages of production within the location.

2. Consider two facilities that run sequentially, all production of the early stages is done in one location, the intermediate products are then moved to another location, where the later stages of production are performed.
3. We use two facilities which run in parallel in relatively close proximity, each handling part of the workload. Intermediate products can be moved between facilities at a (high) inter-building transport cost.
4. Take two facilities in different geographical regions, with no intermediate product movement between them. For each order, the choice of facility where it is manufactured is left to the scheduling algorithm.
5. We choose the layout as in the previous case, but 80% of the orders are preassigned to a facility due to customer location, only 20% of the orders can be assigned freely.

4.3 Results

We now present some results on the solution quality obtained with the different models and solvers.

4.3.1 Selected Solvers

We compare the results for the following solver alternatives, all running on a Windows 10 laptop with i7-6920HQ CPU running at 2.90GHz, and 64 GB of memory. All solvers are using a single core, to ease comparison. CPOptimizer and Cplex allow to use multiple cores, while Chuffed and SICStus do not provide this functionality out of the box. Both MiniZinc and SICStus solutions are run as separate solver processes from the main Java application.

CP Optimizer by IBM, Version 20.1.0, task interval model, default search
SICStus Prolog Version 4.3.5, diffn model, with a custom search routine
MiniZinc Chuffed, Free Search Version 2.5.5, diffn model, free search
MiniZinc Chuffed, Priority Search Version 2.5.5, diffn model, priority search on start and machine variable
MiniZinc Cplex, Free Search Version 20.1.0, diffn model, free search
Cplex Version 20.1.0, MILP model of Section 3.3

The **Dispatch Rule** and **Local Search** solvers of Section 3.4 were implemented in Java, and are run inside the main application.

We use a timeout of 300 seconds for each solver and each instance. The best computed lower bound from Section 3.5 is provided to each solver, and we also provide an initial upper bound by running the Dispatch rule solver for 10 seconds for each instance.

Both the Cplex version of the MiniZinc model, and our own MILP model of Section 3.3 were not able to improve the given upper bound within the timeout, even for the smallest instances. This seems due to a poor initial LP relaxation, probably due to the added machine assignment choice. This contrasts with the results in [27], where MILP models were shown to be competitive for smaller problem instances of the job-shop problem type, where the machine for each task is known a priori.

Table 1 shows a comparison of the different solution approaches for a sample set of instances, ranging from 20–400 jobs.

While for the smaller instances the difference in solution quality is quite small, the difference increase as the number of jobs increases. We can see that the initial upper bound obtained in 10 seconds is quite good, with only modest further improvement made by the Dispatch and Local Search solvers. CPOptimizer with its default search finds better solutions, and is the best overall on medium sized instances, but SICStus Prolog with a custom search

Table 1 Average Cmax Value over 25 instances dependent on size (number of jobs), 8 stages, Lanes Transport, Zipf Exponent 1.05 Job Type Distribution, 10 machines/stage, best solver marked in green, – indicates no solution better than upper bound found, timeout 300s.

Size	Lower Bound	Upper Bound	CP Opt	Chuffed Free	Chuffed Priority	Dispatch Rule	Local Search	SICStus
20	61.88	63.56	62.72	63.48	63.04	63.28	63.20	62.72
25	62.84	65.96	64.24	–	64.76	65.20	64.84	64.16
30	64.12	70.24	66.68	–	68.44	69.16	68.24	66.84
40	65.32	77.36	72.56	–	75.40	76.08	75.28	73.28
50	67.24	84.52	78.40	–	82.24	83.16	82.24	79.40
100	94.72	120.12	115.16	–	116.96	118.28	118.92	113.04
200	153.08	185.16	180.48	–	181.32	182.80	184.76	176.72
300	214.96	249.12	248.96	–	248.76	246.96	248.88	240.96
400	275.36	311.60	311.28	–	–	308.76	311.40	303.16

on average provides the best solution for larger instances. The 32 bit version of SICStus used here runs out of memory when we increase the problem size even further. The Chuffed free search only finds a better solutions than the given upper bound for the smallest problem size, the priority search in Chuffed scales better, but also times out for larger problem instances.

4.3.2 Scenario 1

Table 2 shows the result for a scenario with 200 jobs solved with different solvers. For each solver, we show the average makespan for a given parameter value, as well as the percentage increase over the best result obtained for that solver. We see that increasing the reach of the transport improves the quality of the schedule that can be reached for all solvers except CPOptimizer. CPOptimizer is much less affected by the transport restriction, and finds the best solutions for the strongly constrained cases, but is not as successful for the more relaxed cases. The custom search for SICStus finds the best solutions in the more relaxed cases, but only finds rather poor solutions in the more restricted scenarios. Comparing the best results for all solvers, not allowing transport beyond the initial lanes (limit 0) incurs a cost of 4.92%, while only allowing movement to neighboring lanes (limit 1) costs 2.66% over the unrestricted case. Note that for some instances better solutions were found by imposing a limit on the transport.

4.3.3 Scenario 2

Figure 2 illustrates the different layout alternatives studied in Scenario 2, with the relevant transport times indicated for one machine in stage 3, which is linked to machines in stages 4 and 5, as stage 4 is skipped for certain products. In Scenario 2a, we consider a single facility, where the transport time between machines is determined by their distance. In Scenario 2b, there are two facilities, transport between the facilities requires a longer inter-building transport time, marked in red. In Scenario 2c, there are two facilities working in parallel, where jobs can be moved between the buildings, at a higher cost. This transport is not allowed in Scenario 2d and 2e, jobs can only move between machines in the same facility.

Table 3 shows the results for a run of 25 instances with 200 jobs each for the different layout scenarios. We concentrate on the results with SICStus, which are better than those for the other solvers, and which differentiate the different scenarios more clearly. Splitting the production between two facilities sequentially (Scenario 2b) increases the makespan by 4.5% on average, as every job is delayed by the long transport between the facilities. On the

Table 2 Results for Scenario 1 – Changing transportLimit parameter for different solvers, average Makespan over 25 instances, 8 stages, 10 machines/stage, Zipf exponent 1.05 Job Type Distribution, average lower bound is 153.08, best average solution marked in green.

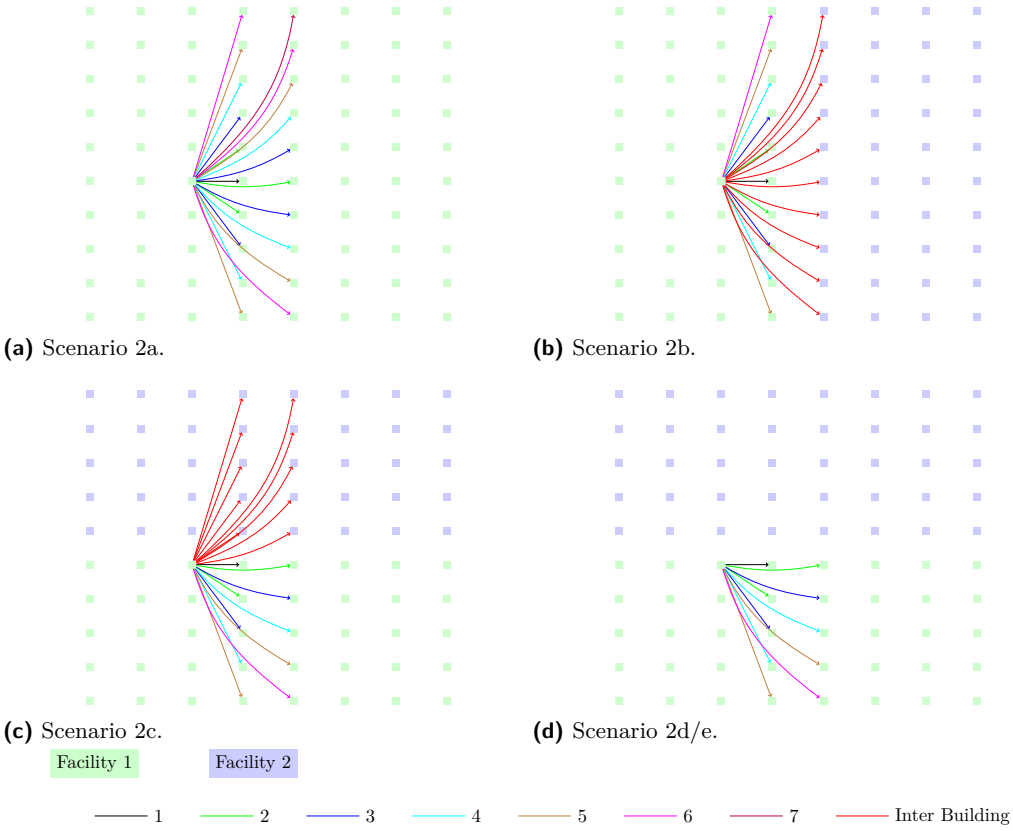
	Transport Limit Between Lanes						No
Solver	0	1	2	3	4	5	Limit
SICStus	204.40	184.20	180.44	178.52	177.88	177.40	177.36
% over Best	15.51	4.09	1.97	0.88	0.52	0.25	
CPOptimizer	186.08	182.96	180.60	180.72	180.80	180.84	180.88
% over Best	3.89	2.14	0.83	0.89	0.94	0.96	
Dispatch Rule	200.84	186.72	184.08	183.72	183.32	182.92	182.92
% over Best	9.99	2.26	0.81	0.61	0.39	0.18	
Local Search	199.32	188.56	186.20	185.48	185.12	184.68	184.80
% over Best	8.21	2.37	1.09	0.69	0.50	0.26	0.33

other hand, splitting the work between two facilities in parallel has a smaller impact. If we are still able to transport intermediate products between locations (Scenario 2c), then the makespan increases by less than 1%. This doubles to 2% if transport between facilities is no longer allowed in Scenario 2d. Restricting the problem further, by considering the assignment of jobs to facility as given for 80% of the orders, causes no further degradation in result quality. Instead, for a small number of cases, the scheduler finds an improved solution, as the domain of the machine assignment variables is cut in half, reducing the search complexity. We are of course not able to explore any of these search spaces completely. But the results for the SICStus solver are statistically significant, when tested with paired t-tests. With the exception of scenarios 2d and 2e, which cannot be distinguished, all scenarios produce significantly different results over the generated 25 instances.

The change of the schedule quality based on the layout scenario is only one aspect in evaluating the alternatives, other parts use customer specific data, and are not reported here.

Table 3 Results for Scenario 2 – Average Makespan over 25 instances with 200 jobs, different Solvers, 10 machines/stage, Zipf 1.05 Job Type Distribution, average lower bound is 159.80.

		Scenario				
Solver	Size	2a	2b	2c	2d	2e
SICStus	200	176.84	184.84	178.28	180.52	180.48
% over Best		0.00	4.52	0.81	2.08	
CPOptimizer	200	184.40	190.92	186.00	183.52	183.52
% over Best		1.23	4.81	2.11	0.75	
Dispatch	200	182.76	190.44	184.28	184.60	184.64
% over Best		0.00	4.20	0.83	1.01	
Local Search	200	184.68	192.24	185.76	186.08	185.96
% over Best		0.13	4.23	0.72	0.89	



(a) Scenario 2a.

(b) Scenario 2b.

(c) Scenario 2c.

(d) Scenario 2d/e.

Figure 2 Scenario 2 - Layout Alternatives and Sample Transport Times.

5 Future Work and Conclusion

In this paper we have presented a novel variant of a hybrid, flexible flowshop problem which adds transportation time between stages. We have explored different models for the problem, using CP and other technologies, and compared them on a number of generated problem instances, based on parameters given by the industrial partner. While some solvers seem promising on small instances, several of the solvers considered failed to handle the problem sizes required for a realistic scenario. We then considered two factory design problems and compared the solutions obtained for different problem settings. These results can be used to evaluate the impact of the design choices on the operational efficiency of a plant layout. While the results obtained are very encouraging, there is still a rather big gap between the lower bound found and the best solutions. Future work will be focused on improving the bounds, while also considering hybridisation techniques that might allow us to find better solutions for large problem instances. Finally, we believe that the problem of designing optimized facility layouts, where the positions of all the machines are computed to increase efficiency, may be of interest for future research.

References

1 Leonard Adler, Nelson Fraiman, Edward Kobacker, Michael Pinedo, Juan Carlos Plotnicoff, and Tso Pang Wu. BPSS: A scheduling support system for the packaging industry. *Operations Research*, 41(4):641–648, 1993. doi:10.1287/opre.41.4.641.

2 Abder Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17:57–73, 1993. doi:10.1016/0895-7177(93)90068-A.

3 Fardin Ahmadizar and Parmis Shahmaleki. Group-shop scheduling with sequence-dependent set-up and transportation times. *Applied Mathematical Modelling*, 38(21):5080-5091, 2014. doi:10.1016/j.apm.2014.03.035.

4 Miguel F. Anjos and Manuel V.C. Vieira. Mathematical optimization approaches for facility layout problems: The state-of-the-art and future research directions. *European Journal of Operational Research*, 261(1):1–16, 2017. doi:10.1016/j.ejor.2017.01.049.

5 Farhad Azadivar and John(Jian) Wang. Facility layout optimization using simulation and genetic algorithms. *International Journal of Production Research*, 38(17):4369–4383, 2000. doi:10.1080/00207540050205154.

6 Javad Behnamian, Seyyed Mohammad Taghi Fatemi Ghomi, Fariborz Jolai, and Omid Amirtaheri. Realistic two-stage flowshop batch scheduling problems with transportation capacity and times. *Applied Mathematical Modelling*, 36(2):723-735, 2012. doi:10.1016/j.apm.2011.07.011.

7 Nicolas Beldiceanu and Evelyne Contejean. Introducing Global Constraints in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, 1994. doi:10.1016/0895-7177(94)90127-9.

8 Fehmi Burcin Ozsoydan and Müjgan Sağir. Iterated greedy algorithms enhanced by hyper-heuristic based learning for hybrid flexible flowshop scheduling problem with sequence dependent setup times: A case study at a manufacturing plant. *Computers & Operations Research*, 125:105044, 2021. doi:10.1016/j.cor.2020.105044.

9 Mats Carlsson and Per Mildner. SICStus Prolog-the first 25 years. *Theory and Practice of Logic Programming*, 12(1-2):35–66, 2012. doi:10.1017/S1471068411000482.

10 Hyun-Seon Choi, Ji-Su Kim, and Dong-Ho Lee. Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line. *Expert Systems with Applications*, 38(4):3514–3521, 2011. doi:10.1016/j.eswa.2010.08.139.

11 Geoffrey Chu. *Improving Combinatorial Optimization*. PhD thesis, The University of Melbourne, 2011.

12 Mehmet Dincbas, Pascal Van Hentenryck, Helmut Simonis, Abderrahmane Aggoun, Thomas Graf, and Françoise Berthier. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems, FGCS 1988, Tokyo, Japan, November 28-December 2, 1988*, pages 693–702. OHMSHA Ltd. Tokyo and Springer-Verlag, 1988.

13 Housni Djellab and Khaled Djellab. Preemptive hybrid flowshop scheduling problem of interval orders. *European Journal of Operational Research*, 137(1):37–49, 2002. doi:10.1016/S0377-2217(01)00094-7.

14 Amine Drira, Henri Pierreval, and Sonia Hajri-Gabouj. Facility layout problems: A survey. *Annual Reviews in Control*, 31(2):255–267, 2007. doi:10.1016/j.arcontrol.2007.04.001.

15 Atabak Elmi and Seyda Topaloglu. Scheduling multiple parts in hybrid flow shop robotic cells served by a single robot. *International Journal of Computer Integrated Manufacturing*, 27(12):1144-1159, 2014. doi:10.1080/0951192X.2013.874576.

16 Atabak Elmi and Seyda Topaloglu. Multi-degree cyclic flow shop robotic cell scheduling problem: Ant colony optimization. *Computers & Operations Research*, 73:67–83, 2016. doi:10.1016/j.cor.2016.03.007.

17 Atabak Elmi and Seyda Topaloglu. Multi-degree cyclic flow shop robotic cell scheduling problem with multiple robots. *International Journal of Computer Integrated Manufacturing*, 30(8):805–821, 2017. doi:10.1080/0951192X.2016.1210231.

18 Thibaut Feydy, Adrian Goldwaser, Andreas Schutt, Peter Stuckey, and Kenneth Young. Priority search with MiniZinc. In *ModRef 2017: The Sixteenth International Workshop on Constraint Modelling and Reformulation*, 2017.

19 Erik Flores Garcia, Enrique Ruiz Zúñiga, Jessica Bruch, Matias Urenda Moris, and Anna Syberfeldt. Simulation-based optimization for facility layout design in conditions of high uncertainty. *Procedia CIRP*, 72:334–339, 2018. 51st CIRP Conference on Manufacturing Systems. doi:10.1016/j.procir.2018.03.227.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.

21 Ronald Graham, Eugene Lawler, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979. doi:10.1016/S0167-5060(08)70356-X.

22 Alain Guinet. Textile production systems: a succession of non-identical parallel processor shops. *Journal of the Operational Research Society*, 42(8):655–671, 1991. doi:10.1057/jors.1991.132.

23 Mohamed Haouari and Lotfi Hidri. On the hybrid flowshop scheduling problem. *International Journal of Production Economics*, 113:495-497, May 2008. doi:10.1016/j.ijpe.2007.10.007.

24 Lofti Hidri, Saneur Elkosantini, and Mohammed M. Mabkhot. Exact and heuristic procedures for the two-center hybrid flow shop scheduling problem with transportation times. *IEEE Access*, 6:21788–21801, 2018. doi:10.1109/ACCESS.2018.2826069.

25 Zhihong Jin, Katsuhisa Ohno, Takahiro Ito, and Salah E. Elmaghraby. Scheduling hybrid flowshops in printed circuit board assembly lines. *Production and Operations Management*, 11(2):216–230, 2002. doi:10.1111/j.1937-5956.2002.tb00492.x.

26 Antoine Jouglet, Ceyda Oğuz, and Marc Sevaux. Hybrid flow-shop: a memetic algorithm using constraint-based scheduling for efficient search. *Journal of Mathematical Modelling and Algorithms*, 8(3):271–292, 2009. doi:10.1007/s10852-008-9101-1.

27 Wen-Yang Ku and Christopher Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73:165–173, 2016. doi:10.1016/j.cor.2016.04.006.

28 Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250, 2018. doi:10.1007/s10601-018-9281-x.

29 Michael Langston. Interstage transportation planning in the deterministic flowshop environment. *Operations Research*, 35(4):556–564, 1987. doi:10.1287/opre.35.4.556.

30 Chuanjin Lei, Ning Zhao, Song Ye, and Xiuli Wu. Memetic algorithm for solving flexible flow-shop scheduling problems with dynamic transport waiting times. *Computers & Industrial Engineering*, 139:105984, 2020. doi:10.1016/j.cie.2019.07.041.

31 P.L. Maggu and G. Das. On 2×n sequencing problem with transportaion times of jobs. *Pure and Applied Mathematika Sciences*, 12:1-6, 1980.

32 Byungsoo Na, Shabbir Ahmed, George Nemhauser, and Joel Sokol. A cutting and scheduling problem in float glass manufacturing. *Journal of Scheduling*, 17(1):95–107, 2014. doi:10.1007/s10951-013-0335-z.

33 Bahman Naderi, Sheida Gohari, and Mehdi Yazdani. Hybrid flexible flowshop problems: Models and solution methods. *Applied Mathematical Modelling*, 38(24):5767–5780, 2014. doi:10.1016/j.apm.2014.04.012.

34 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007. doi:10.1007/978-3-540-74970-7_38.

35 Andrea Rossi. Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. *International Journal of Production Economics*, 153(C):253–267, 2014. doi:10.1016/j.ijpe.2014.03.006.

36 Rubén Ruiz and José Antonio Vázquez Rodríguez. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205:1–18, 2010. doi:10.1016/j.ejor.2009.09.024.

37 Rubén Ruiz, Funda Sivrikaya Şerifoğlu, and Thijs Urlings. Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, 35(4):1151-1175, 2008. doi:10.1016/j.cor.2006.07.014.

38 Helmut Simonis, Michele Garraffa, Barry O'Sullivan, and Eddie Armstrong. Dataset for Article: Hybrid Flexible Flowshop with Transportation Times at CP 2021, August 2021. doi:10.5281/zenodo.5168966.

39 Ricardo Soto, Hakan Kjellerstrand, Juan Gutiérrez, Alexis López, Broderick Crawford, and Eric Monfroy. Solving manufacturing cell design problems using constraint programming. In He Jiang, Wei Ding, Moonis Ali, and Xindong Wu, editors, *Advanced Research in Applied Artificial Intelligence*, pages 400–406, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

40 Ömür Tosun, Mariappan Kadarkarainadar Marichelvam, and Nedret Tosun. A literature review on hybrid flow shop scheduling. *International Journal of Advanced Operations Management*, 12(2):156-194, 2020. doi:10.1504/ijaom.2020.108263.

41 Srisatja Vitayasak, Pupong Pongcharoen, and Christian Hicks. Robust machine layout design under dynamic environment:dynamic customer demand and machine maintenance. *Expert Systems with Applications*, 3:100015, 2019. doi:10.1016/j.eswax.2019.100015.

42 Hua Xuan. Hybrid flowshop scheduling with finite transportation capacity. *Applied Mechanics and Materials*, 65:574-578, 2011. doi:10.4028/www.scientific.net/amm.65.574.

43 Sami Yamani Douzi Sorkhabi, David Romero, Christopher Beck, and Cristina Amon. Constrained multi-objective wind farm layout optimization: Novel constraint handling approach based on constraint programming. *Renewable Energy*, 126:341–353, 2018. doi:10.1016/j.renene.2018.03.053.

44 Weiya Zhong and Zhi-Long Chen. Flowshop scheduling with interstage job transportation. *Journal of Scheduling*, 18(4):411–422, 2015. doi:10.1007/s10951-014-0409-6.

45 Jinlian Zhou, Ying Guo, and Guipeng Li. On complex hybrid flexible flowshop scheduling problems based on constraint programming. In *12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 909–913, 2015. doi:10.1109/fskd.2015.7382064.