Flexible job shop with continuous material flow

A. Bożek* and M. Wysocki

Department of Computer and Control Engineering, Rzeszów University of Technology, Rzeszów, Poland

(Received 16 April 2014; accepted 11 August 2014)

A new scheduling problem, the continuous flow flexible job shop (CF-FJS) is proposed. The formulation combines the well-known flexible job shop (FJS) problem and a dedicated continuous material flow model (MFM). In the MFM, operations are represented by material flow functions derived by integration of arbitrarily defined speed patterns. Two main concepts of the MFM formalism, i.e. variable speed of processing and continuous material flow, lead to position-dependent processing times and overlapping in operations which extend standard FJS formulation. Properties of the CF-FJS are investigated. A tabu search scheduling algorithm utilising these properties is proposed. Effective neighbourhood functions are defined based on elimination approaches. Two auxiliary procedures: search intensification level switching and fast feasibility detection are added to improve algorithm efficiency. The algorithm is verified using dedicated benchmark instances which comprise non-trivial representations of the CF-FJS specific features, i.e. machine efficiency patterns and minimum inter-operation buffers. The research is motivated by task scheduling in a fastener factory, but the presented results can be useful in many domains, such as production of granular goods, steel details, glass and fluids. The solution can be used in real-world applications. The published results can be helpful in testing new CF-FJS scheduling algorithms.

Keywords: flexible job shop; production modelling; material flow; tabu search; critical path; block approach

1. Introduction

In this work, the continuous flow flexible job shop (CF-FJS) scheduling problem is introduced that combines the flexible job shop (FJS) production environment and dedicated continuous material flow model (MFM). The continuous material flow results from a continuous process or it approximates flow of small details processed in large number. The MFM formalism is based on the assumption that processing speed is defined by a piecewise constant time-varying function. Changes of processing speed can be an effect of shift plans, energy tariffs, variable number of workers and so on. In particular, the process or machine speed can be equal to zero in some time ranges that represent unavailability.

The proposed CF-FJS scheduling problem can be concisely characterised by enumeration of similarities and disparities between it and the standard FJS. The CF-FJS is similar to the FJS in the following aspects:

- There is a set of jobs and each job is defined by an individual and arbitrary given sequence of operations. Processing order has to follow these sequences.
- For each operation, a set of available machines is defined.
- There is the resource constraint which allows any machine to process at most one operation at a time.
- Minimisation of the makespan is the objective.

There are, however, important differences between the two formulations:

- In the FJS, machines are available without restrictions and have no individual parameters. In the CF-FJS, each machine has assigned an efficiency pattern, which is a function of time, such that the product of the efficiency pattern and a nominal processing speed of an operation represents the processing speed function of this operation.
- Each operation in the FJS is represented by a fixed (i.e. position independent) interval of time, whereas operations in the CF-FJS are modelled by material flow functions (integrals of processing speed functions) and resulting processing time is, in general, position dependent.
- In the FJS, a consecutive operation of a job can start when the previous one is completed. Because of continuous processing, in the CF-FJS, sequential operations can overlap, providing that minimum buffer between their material flow functions is maintained.
- The CF-FJS is generalisation of the FJS and it can represent FJS instances as special cases.

*Corresponding author. Email: abozek@kia.prz.edu.pl

© 2014 Taylor & Francis
A tabu search-based scheduling algorithm has been implemented for the CF-FJS. Properties of the CF-FJS have been taken into account in this implementation. The neighbourhood definition exploits proved features of the CJ-FJS to exclude from search a subset of the moves which cannot result in immediate objective improvement. A mechanism of search intensification level switching (SILS) has been implemented. It intensifies density of neighbourhood gradually while the objective is minimised. A procedure of fast feasibility detection (FFD) has been added to the algorithm. The FFD makes it possible to predict feasibility of a new solution obtained by a single move with constant time complexity.

The research is motivated by task scheduling in a fastener factory. In this factory, lots of tiny steel elements are processed. For each lot, a sequence of operations is defined. Each operation has a predefined set of adequate machines and one of them has to be chosen for processing this operation. Machines have assigned efficiency calendars based on shift plans, planned maintenance tasks and other conditions. The elements are transported between consecutive operations constantly and a minimum transport buffer between them should be maintained. This production organisation corresponds to the CF-FJS definition. We believe that the presented solution will be useful in many domains, e.g. production of granular goods, steel details, glass, fluids and others. The features of scheduling problems considered in this work are relevant in the context of real-world scheduling applications. Leading software packages give the possibility of defining efficiency calendars for resources. Such calendars can be considered as piecewise constant time-varying functions with the interpretation analogous to the speed and efficiency pattern concepts used in this paper. For example, in the SAP™ ERP these calendars are referred to as shift sequences, and in the Preactor™ scheduling system as resource calendars. Support for overlapping in operations is also present. In the SAP APO-PP/DS module, the overlapping can be configured by means of the two parameters: minimum send-ahead quantity and minimum overlap time. In Preactor, it results from splitting batches into smaller lots and transferring them individually across consecutive processes (transfer batching) that is a variant of the well-known lot streaming method. This work introduces a new enhanced model of overlapping in which the material buffer determines the minimum quantity between overlapped operations for all the time of overlapping, not at the start of the next operation, as in the case of the minimum send-ahead quantity parameter. The overlapping model, the complete CF-FJS production environment, and the scheduling algorithm for it can be used for development of new or extending functionality of existing scheduling software.

2. Related work

Currently, the research on the FJS and similar scheduling problems makes for studying the problems with modifications and extensions that move them closer to the industrial usability. For example, the problems with limited machine availability and breakdowns (Sawik 2012), operations blocking (Gröflin, Pham, and Bürgy 2011), limited resources (Rajkumar et al. 2011), multiple process plans (Doh et al. 2013) and assembly (Gomes, Barbosa-Póvoa, and Novais 2013) are considered. It is quite common to complement the FJS formulation with set-up times (Mousakhani 2013) and also with transfer times (Gröflin, Pham, and Bürgy 2011). Reactive scheduling can be considered to deal with dynamic FJS problems (Gomes, Barbosa-Póvoa, and Novais 2013). Broad range of criteria is used, e.g. makespan, mean and total tardiness (Mousakhani 2013), maximum lateness (Vilcot and Billaut 2011), total flow time, the number of tardy jobs (Doh et al. 2013), earliness, in-process inventory (Gomes, Barbosa-Póvoa, and Novais 2013), total and maximum workload of machines (Rajkumar et al. 2011).

In the context of the FJS problem, a feature of overlapping in operations is also considered (Alvarez-Valdez et al. 2005; Fattahi, Jolai, and Arkat 2009; Fattahi and Mehrabad 2009; Farughi et al. 2011). In the mentioned works, the overlapping is controlled by an operation-related parameter which defines the proportion of the operation that has to be processed before its successor can start. This overlapping model may not be powerful enough to deal with complex time relations in continuous processing systems. On the other hand, scheduling of continuous processes has its own solutions that are considered especially in the context of chemical industry (Floudas and Lin 2004; Shaik and Floudas 2007; Shaik et al. 2009). The effect of operation overlapping is also present in lot streaming problems (Lei and Guo 2013), however, processing is not continuous in these problems. It has been rather unusual so far to join advantages of continuous processing models and the (flexible) job shop environment, although it is possible and has been applied e.g. for algorithm benchmarking (Sourd and Rogerie 2005).

Many different algorithms are applied for solving the FJS and related problems (as job shop or flexible flow shop problems), e.g. priority rules (Doh et al. 2013; Tay and Ho 2008), mathematical programming (Gomes, Barbosa-Póvoa, and Novais 2013; Sawik 2012), tabu search (Gröflin, Pham, and Bürgy 2011; Vilcot and Billaut 2011), genetic algorithms (Na and Park 2013), genetic programming (Tay and Ho 2008), agent-based systems (Rajabinasab and Mansour 2011), bee colony (Lei and Guo 2013), greedy randomised procedures (Rajkumar et al. 2011), iterated local search (Mousakhani 2013) and others.

Some scheduling problems can be solved more efficiently if a well-suited neighbourhood is used, e.g. based on the critical path and the operations block ideas. Such approach is used for the single-machine scheduling with release dates and due dates as objective (Grabowski, Nowicki, and Zdrzalka 1986), the job shop with makespan minimisation (Nowicki and Smutnicki 1996; Perregaard and Clausen 1998; Pezzella and Merelli 2000) and the single-machine scheduling with the weighted sum
of earliness and tardiness minimisation (Wodecki 2009). The algorithm using the block approach is also implemented for the FJS problem (Bożejko, Uchroński, and Wodecki 2010; Li, Pan, and Liang 2010).

FJS scheduling approaches based on tabu search are often improved by additional mechanisms. Grabowski and Wodecki (2005) use fast prediction of solution feasibility that exploits graph representation of a problem, estimation of objective value by means of its lower bound, dynamic length of tabu list and mechanism of perturbation in move selection. Li, Pan, and Liang (2010) propose a hybrid tabu search algorithm (HTSA) in which the basic metaheuristic is extended by a variable neighbourhood search (VNS), a dynamic length tabu list and a left-shift function that decode an encoding solution to an active schedule.

The solution proposed in this paper relates to the above-mentioned concepts and approaches, i.e. the FJS problem, the continuous processing model, as well as the tabu search algorithm with improved optimisation mechanisms. The introduced CF-FJS is generalisation of the standard FJS which allows to solve patently broader classes of problems that are combination of the well-known FJS with continuous processing in which variable machines efficiency, position-dependent processing times and overlapping in operations are present. The paper contributes to the research field of production scheduling by formulating the CF-FJS problem, identifying its properties and proposing a tabu search-based scheduling algorithm. A set of benchmark instances of the CF-FJS are also prepared and the results of scheduling experiments based on them are presented.

3. Problem formulation
The CF-FJS problem has a two-level formulation. On the lower level, single operations are modelled using the MFM formalism. These operation representations are embedded into the FJS for modelling complete production environment on the higher level (Figure 1).

3.1 Material flow model (MFM)
In the MFM, operations are represented by material flows (mf) derived by integration of the speed patterns (sp). The material flows are described by piecewise linear time-varying functions, whereas the speed patterns are described by piecewise constant time-varying functions. In the example on Figure 1, a single operation Ok,i,j processed on a machine Mp,i is represented by means of these functions. The speed pattern spk,i,p is defined for the pair (Ok,i,j, Mp,i). The speed pattern can be distinct for each pair (operation, machine) on the level of the MFM, but it is expressed in the CF-FJS as the result of multiplication of the operation-related constant nominal speed and the machine-related efficiency pattern (ep). For the pair (Ok,i,j, Mp,i), the minimum material buffer bfk,i,p is also defined. Providing that the material flow mfk,i−1 of the previous operation Ok,i−1 of the same job is known (dashed line on Figure 1), the MFM makes it possible to determine the material flow mfk,i of the operation Ok,i. This flow is calculated by integration of spk,i,p in the limits from an arbitrary chosen start time to the time at which the flow reaches a maximum value assigned to the job, these limits are interpreted as the start and completion time of the operation Ok,i. There are two examples of such flows on Figure 1, i.e. mfk,i and mfk,j. The flow mfk,i starts at the time tes called earliest start time. It is the minimum possible start time that ensures the required difference between mfk,i−1 and mfk,i equal to at least bfk,i,p. The time tes is used as the sequence constraint assigned to an operation, i.e. the operation can start at any time t ≥ tes, e.g the flow mfk,i starts at t′ ≥ tes. In general, the consecutive operations modelled by the MFM can

![Figure 1. The idea of the CF-FJS problem.](image-url)
overlap \((O_{k,i-1} \text{ modelled by } mf_{k,i-1} \text{ overlaps with } O_{k,i} \text{ modelled by } mf_{k,i}^a \text{ or } mf_{k,i}^b)\) and processing time of an operation is position-dependent, e.g. \(t_c - t_{es} \neq t'_c - t'_s\) (Figure 1).

Formally, the MFM is defined as 10-tuple of concepts

\[
\text{MFM} = (T_D, F_D, S_D, D_D, MF_S, SP_S, EP_S, OMF_P, CMT_P, EST_P),
\]

where

- \(T_D, F_D, S_D, \text{ and } D_D\) are domains of scalars representing time, flow, speed, and dimensionless (or percentage) values, respectively,
- \(MF_S, SP_S, \text{ and } EP_S\) are sets of data structures representing material flows, speed patterns, and efficiency patterns, respectively,
- \(OMF_P, CMT_P, \text{ and } EST_P\) are computational procedures of the MFM.

The procedure \(OMF_P\) computes output material flow \(mf_{out}\) that is the result of processing performed by a resource with a given speed pattern \(sp\) characterised by start time \(t_s\) and total amount of processed material \(f_{max}\)

\[
OMF_P : SP_S \times T_D \times F_D \rightarrow MF_S, \quad mf_{out} = OMF_P (s_p, t_s, f_{max}).
\]

The procedure \(CMT_P\) takes the same arguments as \(OMF_P\), but it returns completion time \(t_c\) of the related operation

\[
CMT_P : SP_S \times T_D \times F_D \rightarrow T_D, \quad t_c = CMT_P (s_p, t_s, f_{max}).
\]

The procedure \(EST_P\) computes earliest start time \(t_{es}\) of an operation on the basis of a speed pattern of the operation \(sp_{out}\), a speed pattern \(sp_{in}\) and start time \(t_{s\_in}\) of a preceding operation, total amount of processed material \(f_{max}\) and a minimum material buffer between the operations \(bf\)

\[
EST_P : T_D \times SP_S \times SP_S \times D_D^{[0,1]} \times F_D \rightarrow T_D, \quad t_{es} = EST_P (t_{s\_in}, sp_{in}, sp_{out}, bf, f_{max}).
\]

The following two properties of the MFM have a significant impact on features of scheduling problems defined on the basis of it.

**MFM Property 1** The completion time of an operation is a non-decreasing function of its start time, i.e.

\[
\forall \quad s_{p}\in SP_S, t_{s\_s} \in T_D, f_{max} \in F_D, f_{max} > 0 \quad t_{s\_s} > t_{s\_1} \implies CMT_P (s_{p}, t_{s\_s}, f_{max}) \geq CMT_P (s_{p}, t_{s\_1}, f_{max}).
\]

**MFM Property 2** The earliest start time of an operation is a non-decreasing function of the start time of the previous one, i.e.

\[
\forall \quad s_{p\_in}, f_{max} \in SP_S, t_{s\_s}, t_{s\_1} \in T_D, t_{es} \in D_D^{[0,1]}, f_{max} \in F_D, f_{max} > 0 \quad t_{s\_s} > t_{s\_1} \implies EST_P (s_{p\_in}, s_{p\_out}, bf, f_{max}) \geq EST_P (s_{p\_in}, s_{p\_out}, bf, f_{max}).
\]

A detailed description of the MFM formalism is available as supplemental online material (Appendix A).

### 3.2 CF-FJS problem

Definitions of sets and parameters used for the CF-FJS problem formulation are presented in Table 1. The problem is formulated as follows

1. There is a non-empty set \(M\) of machines (processors) \(M_p\) which has \(r\) elements.
2. For each machine \(M_p \in M\) an efficiency pattern \(e_p \in EP_S\) is defined.
3. There is a non-empty set \(J\) of jobs \(J_k\) which has \(n\) elements.
4. For each job \(J_k \in J\) a non-empty set \(O_k\) of operations \(O_{k,i}\) is given which has \(js\) elements.
5. For each operation \(O_{k,i} \in O_k\) a non-empty set \(M_{k,i}\) of machines \(M_{k,i,p}\) which are allowed for processing the operation is defined. This set has \(ms\) elements.
6. For each pair \((O_{k,i}, M_p)\) a parameter \(p_{k,i,p} \in T_D\) expresses a nominal processing time of the operation \(O_{k,i} \in O_k\) on the machine \(M_p \in M_{k,i}\).
(7) For each pair \( (O_{k,i}, M_p) \), such that \( i \geq 2 \) (i.e. except for the first operations of jobs), a parameter \( b_{k,i,p} \in D^{[0,1]}_D \) expresses a material buffer required for processing the operation \( O_{k,i} \in O_k \) on the machine \( M_p \in M_{k,i} \).

(8) The solution of the CF-FSJ problem will be represented as a function

\[
S : \bigcup_{k=1}^{n} O_k \rightarrow M \times D_n,
\]

\[
(M_{\rho(k,i)}, t_{s,(k,i)}) = S \left( O_{k,i} \right), \quad O_{k,i} \in O_k, \quad M_{\rho(k,i)} \in M_{k,i}, \quad t_{s,(k,i)} \in T_n.
\]

(9) The solution is feasible if and only if it is possible to represent it by the function \( S \) (1) and the following relations and conditions are met

\[
\forall_{k \in \{1, \ldots, n\}, \ i \in \{1, \ldots, js(k)\}, \ v = \rho(k,i)} t_{c,(k,i)} = \text{CMT}\left( e_v/p_{k,i,v}, t_{s,(k,i)}, 1 \right), \quad (2a)
\]

\[
\forall_{k \in \{1, \ldots, n\}} t_{s,(k,1)} = 0, \quad (2b)
\]

\[
\forall_{k \in \{1, \ldots, n\}, \ i \in \{1, \ldots, js(k)\}} t_{s,(k,i)} = \text{EST}\left( t_{s,(k,i-1)}, e_v/p_{k,i-1,v}, e_v/p_{k,i,v}, b_{k,i,v}, 1 \right), \quad (2c)
\]

\[
\forall_{k \in \{1, \ldots, n\}, \ i \in \{1, \ldots, js(k)\}} c = \rho(k,i-1), v = \rho(k,i) \Rightarrow t_{s,(k,i)} \geq t_{s,(k,i)}, \quad (2d)
\]

\[
M_{\rho(k_1,i_1)} = M_{\rho(k_2,i_2)} \implies t_{c,(k_1,i_1)} \leq t_{s,(k_2,i_2)} \lor t_{c,(k_2,i_2)} \leq t_{s,(k_1,i_1)}, \quad (2e)
\]

(10) The objective is to find the solution \( S^* \) such that

\[
\max_{k \in \{1, \ldots, n\}} \{ t_{c,(k,js(k))} \} \rightarrow \min,
\]

where the values \( t_{c,(k,js(k))} \) are calculated according to the expression (2a) on the basis of \( S^* \). This is the objective commonly referred to as makespan and denoted as \( C_{\text{max}} \).

(11) The set of all CF-FJS problems will be denoted by \( P_{\text{CF}} \). The set of all solutions of the problem \( P \in P_{\text{CF}} \) will be denoted by \( S^P \).
The solution graph represents a single semi-active (Pinedo 2008) solution of the scheduling problem. The semi-active solution will be denoted by $\bar{S}$ in the remainder of the work.

Let us define a processing order $\pi = (k, i)$, where $\kappa (p, q)$ and $\iota (p, q)$ are the functions that return indices $k$ and $i$, respectively, if the operation $O_{k, i}$ is to be processed by the machine $M_p$ on the position $q$ of its processing queue. The number of operations in the processing queue of machine $M_p$ is expressed by $\psi (p)$. We assume that each operation is assigned exactly once to some machine.

The actual processing time of each operation $p_{k, i}^{\text{act}} = t_{c, (k, i)} - t_{s, (k, i)}$ is split into two components $p_{k, i}^{A}$ and $p_{k, i}^{B}$ (see Figure 2), such that

\begin{itemize}
  \item The index of the machine assigned by a solution to the operation $O_{k, i}$ is denoted as $\rho (k, i)$. The earliest start time, the start time and the completion time of this operation are denoted as $t_{s, (k, i)}$, $t_{s, (k, i)}$ and $t_{c, (k, i)}$, respectively.
  \item It has been arbitrarily assumed that $f_{\text{max}} = 1$ for all operations and this value is put into the expressions (2a) and (2c). Hence, the material flow is normalised to the range $[0, 1]$. The constraint (2a) specifies that the completion time $t_{c, (k, i)}$ of each operation $O_{k, i}$ is determined by its start time $t_{s, (k, i)}$, the efficiency pattern $e_i$ of the assigned machine $M_v$ and the nominal processing time $p_{k, i, v}$, with the use of the procedure CMT. The first operation $O_{k, 1}$ of each job $J_k \in \mathcal{J}$ can start at time 0 (2b), however, the earliest start time $t_{s, (k, i)}$ of any other operation $O_{k, i}, i > 1$ is restricted by the parameters of the previous operation $O_{k, i-1}$ of the same job (i.e. its start time $t_{s, (k, i-1)}$, the nominal processing time $p_{k, i-1, v}$ and the efficiency pattern $e_i$ of the assigned machine $M_v$), as well as the nominal processing time $p_{k, i, v}$ of the operation $O_{k, i}$ itself, the efficiency pattern $e_i$ of the machine $M_v$ assigned to it and related material buffer $b_{k, i, v}$. This restriction is defined by means of the procedure EST (2c). The actual start time $t_{s, (k, i)}$ of any operation $O_{k, i}$ in the solution is lower bounded by $t_{s, (k, i)}$ (2d). The expressions (2b)–(2d) together constitute the sequence constraint that allows overlapping of operations. The condition (2e) represents the resource constraint that permits only one operation to be processed at a time by a single machine.

4. CF-FJS problem properties

4.1 Solution graph

The solution graph represents a single semi-active (Pinedo 2008) solution of the scheduling problem. The semi-active solution will be denoted by $\bar{S}$ in the remainder of the work.

Let us define a processing order $\pi = (k, i)$, where $\kappa (p, q)$ and $\iota (p, q)$ are the functions that return indices $k$ and $i$, respectively, if the operation $O_{k, i}$ is to be processed by the machine $M_p$ on the position $q$ of its processing queue. The number of operations in the processing queue of machine $M_p$ is expressed by $\psi (p)$. We assume that each operation is assigned exactly once to some machine.

The actual processing time of each operation $p_{k, i}^{\text{act}} = t_{c, (k, i)} - t_{s, (k, i)}$ is split into two components $p_{k, i}^{A}$ and $p_{k, i}^{B}$ (see Figure 2), such that

\begin{itemize}
  \item The index of the machine assigned by a solution to the operation $O_{k, i}$ is denoted as $\rho (k, i)$. The earliest start time, the start time and the completion time of this operation are denoted as $t_{s, (k, i)}$, $t_{s, (k, i)}$ and $t_{c, (k, i)}$, respectively.
  \item It has been arbitrarily assumed that $f_{\text{max}} = 1$ for all operations and this value is put into the expressions (2a) and (2c). Hence, the material flow is normalised to the range $[0, 1]$ or $[0, 100\%]$.
  \item The parameter $p_{k, i, p}$ is referred to as nominal processing time, because it is used as the inverse of the nominal speed for the CF-FJS formulation in (2a) and (2c). It is the analogue of the processing time in classic scheduling problems

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure2.png}
\caption{Operation processing times.}
\end{figure}
A and the function \(w\) graph (4d).

Algorithm 1:

**Data:** \(P, \pi, G(\pi) = (N, A)\)

\[ \forall k \in \{1, 2, \ldots, n\}, \quad p_{k,i}^A = \begin{cases} \min \{t_{s,k,i} - t_{s,k,i+1}, t_{c,k,i} - t_{s,k,i}\} & \text{if } i < j(s) \\ t_{c,k,i} - t_{s,k,i} & \text{if } i = j(s) \end{cases} \]  

The value \(p_{k,i}^A\) represents the processing time of \(O_{k,i}\) elapsing before \(O_{k,i+1}\) starts and the value \(p_{k,i}^B\) represents the rest of the processing time of \(O_{k,i}\).

The solution graph \(G(\pi) = (N, A, w)\) related to the processing order \(\pi\) is defined by the set of nodes \(N\), the set of arcs \(A\) and the function \(w : N \rightarrow T_D\) that assigns a weight to each node, according to the formulas (4).

\[
\begin{align*}
N_1 &= \bigcup_{k=1}^{n} N_{k,i}^A, & N_2 &= \bigcup_{p=1}^{r} \bigcup_{q=1}^{ps(p)-1} N_{k,i}^B \bigg|_{k = \pi(p,q), \ i = (p,q)}^{} \bigcup_{i=1}^{n} N_{s}, \\
A_1 &= \bigcup_{k=1}^{n} \left( N_{s}, N_{k,i}^A \right) \cup \left( N_{k,i}^A, N_{c} \right), & A_2 &= \bigcup_{k=1}^{n} \bigcup_{i=1}^{n} N_{k,i}^A, N_{k,i+1}^A, \\
A_3 &= \bigcup_{p=1}^{r} \bigcup_{q=1}^{ps(p)-1} \left( N_{k,i}^A, N_{k,i}^B \right) \bigcup \left( N_{k,i}^B, N_{k,i+1}^A \right) \bigg|_{k = \pi(p,q), \ i = (p,q)}^{} \bigcup_{i=1}^{n} N_{c}, \\
& \forall N_{k,i}^A \in N_1 \quad w \left( N_{k,i}^A \right) = p_{k,i}^A, & \forall N_{k,i}^B \in N_2 \quad w \left( N_{k,i}^B \right) = p_{k,i}^B, & w \left( N_{s} \right) = w \left( N_{c} \right) = 0.
\end{align*}
\]

The nodes \(N_{k,i}^A \in N_1\) and \(N_{k,i}^B \in N_2\) represent the parts of operations that correspond to the times \(p_{k,i}^A\) and \(p_{k,i}^B\), respectively. The dummy start and final nodes \(N_{s}, N_{c} \in N_3\) are also defined, as usually in solution graphs. The arcs \(A_1 \cup A_2\) represent sequence constraints, connecting the nodes that represent consecutive operations of jobs. The arcs \(A_3\) represent resource constraints and reflect processing queues of machines. Values of the function \(w\) are defined for each node in the graph (4d).

**Algorithm 1:** Construction of complete solution graph.

**begin**

\(\xi() \leftarrow \text{generateFunctionXi}(\pi, G(\pi));\) /* \(\xi()\) returns \(k,i,p,q\) for a given node */

\(U := \text{getTopologicalOrder}(G(\pi));\)

foreach \(u \in U\) do

if \(u \notin N_1\) then continue;

\([k, i, p, q] := \xi(u);\)

if \(i > 1\) then select \(v \in N_1 \wedge (v, u) \in A; \{k, \tilde{i}, \tilde{p}, \tilde{q}\} := \xi(v);\)

if \(q > 1\) then select \(b \in N_2 \wedge (b, u) \in A; \{k, \tilde{i}, \tilde{p}, \tilde{q}\} := \xi(b);\)

if \(i = 1\) then \(t_{q} = 0\) else \(t_{q} := \text{EST}(t_{q}(k, \tilde{i}), \epsilon_{p}/p_{k,i,p}, \epsilon_{p}/p_{k,i,p}, b_{k,i,p}, 1);\)

if \(q = 1\) then \(t_{i} = 0\) else \(t_{i} := t_{c}(k, \tilde{i});\)

\(t_{c}(k, i) := \max(t_{c}, t_{i});\)

\(w(N_{k,i}^A) := \text{calculate}_pA(t_{c}, t_{c}, a);\) /* the functions calculate_pA and calculate_pB */

end

foreach \(a \in N_1\) do

\([k, i, p, q] := \xi(a);\)

\(w(N_{k,i}^B) := \text{calculate}_pB(t_{c}, t_{c}, a);\) /* are based on the expression (3) */

end

\(w(N_{s}) := w(N_{c}) := 0;\)

return \(G(\pi) := (N, A, w);\)

**end**
The sequence of all operations \( CP \equiv \{O_1^C, O_2^C, \ldots, O_i^C, \ldots, O_{r_{cpl}}^C\} \) that belong to the longest path of \( G(\pi) \) will be referred to as critical path. It is obvious that \( C_{\max} = t_c \left( O_{cpl}^C \right) \). The critical path is the well-known concept in scheduling theory, however, its properties need to be inspected in the context of the CF-FJS problem.

Let us assume that the start time of some operation \( O_i^C \in CP \), previously equal to \( t_s \left( O_i^C \right) \), changes to \( t_s' \left( O_i^C \right) > t_s \left( O_i^C \right) \), as the result of some additional constraint. Adjacent operations on the critical path either are processed by the same machine or belong to one job. In the first case, the resource constraint is active (see 2e), hence \( t_s \left( O_{i+1}^C \right) = t_s \left( O_i^C \right) \) and on the basis of MFM Property 1 we have

\[
f_{C} \left( O_{i+1}^C \right) > t_s \left( O_{i+1}^C \right) \implies t_s' \left( O_{i+1}^C \right) \geq t_s \left( O_i^C \right) \implies t_s' \left( O_{i+1}^C \right) > t_s \left( O_i^C \right) . \tag{5}\]

In the second case, the sequence constraint is active (see 2d), thus \( t_s \left( O_{i+1}^C \right) = t_{es} \left( O_i^C \right) \) and on the basis of MFM Property 2 we obtain

\[
f_{C} \left( O_{i+1}^C \right) > t_s \left( O_{i+1}^C \right) \implies t_{es} \left( O_{i+1}^C \right) \geq t_{es} \left( O_i^C \right) \implies t_s' \left( O_{i+1}^C \right) \geq t_s \left( O_i^C \right) . \tag{6}\]

The consequence of (5) and (6) is that

\[
\forall_{i \in \{1, 2, \ldots, r_{cpl} - 1\}} t_s \left( O_i^C \right) \implies t_s' \left( O_i^C \right) > t_s \left( O_i^C \right) \implies t_s' \left( O_{i+1}^C \right) \geq t_s \left( O_i^C \right) . \tag{7}\]

On the basis of mathematical induction we have

\[
\exists i \{1, 2, \ldots, r_{cpl}\} t_s \left( O_i^C \right) > t_s \left( O_i^C \right) \wedge S(7) \implies \forall_{i \in \{1, 2, \ldots, r_{cpl}\}} t_s' \left( O_i^C \right) \geq t_s \left( O_i^C \right) .
\]

Finally, regarding (2a) and MFM Property 1, we conclude

\[
f_{C} \left( O_{cpl}^C \right) \geq t_s \left( O_{cpl}^C \right) \implies t_s' \left( O_{cpl}^C \right) \geq t_s \left( O_{cpl}^C \right) \implies C'_{\max} \geq C_{\max} .
\]

Thus, the first property of the critical path can be formulated as follows

**CF-FJS Property 1** If \( \tilde{\mathbf{S}} \) is a semi-active solution of the CF-FJS problem \( \mathbf{P} \) which has a critical path \( CP_{\tilde{\mathbf{S}}} \) and a second solution \( \mathbf{S}' \) is created by delaying the start time of an operation \( O_i^C \in CP_{\mathbf{S}} \) of \( \tilde{\mathbf{S}} \), i.e. \( t_s' \left( O_i^C \right) > t_s \left( O_i^C \right) \), then \( C_{\max}(\mathbf{S}') \geq C_{\max}(\tilde{\mathbf{S}}) \).

The processing order can be modified by moving some operation \( O_{k,i} \) from its initial position to a new one, defined by the machine index \( p \in M_k \) and the position \( q \) of the operation in the machine queue. Such the move will be denoted as \( (k, i \rightarrow p, q) \) and the processing order \( \pi \) changed by this move will be denoted as \( \pi(k, i \rightarrow p, q) \). The notation \( \Psi_{k,i} (\pi) \) will represent the set of all moves allowed for \( O_{k,i} \) and the processing order \( \pi \).

Let us assume that there is a CF-FSJ problem \( \mathbf{P} \) with a semi-active solution \( \tilde{\mathbf{S}}_{\pi} \) based on the processing order \( \pi \), which has a critical path \( CP_{\tilde{\mathbf{S}}_{\pi}} \) and makespan equal \( C_{\max} \). Let us change processing order by moving a freely selected operation \( O_{k,i} \notin CP_{\tilde{\mathbf{S}}_{\pi}} \) to a new freely selected position that results in the new processing order \( \pi' = \pi(k, i \rightarrow p, q) \) that leads to a new semi-active solution \( \tilde{\mathbf{S}}'_{\pi} \) with a new makespan \( C'_{\max} \). It is impossible to obtain \( C'_{\max} < C_{\max} \) in this way. This is because each operation on the critical path has an active constraint related to antecedent operations on this path. These constraints cannot be relaxed without moving any operation from the critical path. Moving other operations may at most activate additional
constraints which cannot result in makespan reduction, according to CF-FJS Property 1. Hence, the second property can be formulated.

**CF-FJS Property 2**

\[ \forall p \in BF \left[ \pi' = \pi(k,i \rightarrow p,q) \land C_{\text{max}}(\tilde{S}_{\pi'}) < C_{\text{max}}(\tilde{S}_{\pi}) \right] \implies O_{k,i} \in CP_{S'_\pi}. \]

### 4.3 Block property

It is proved below that the block elimination property (Grabowski, Nowicki, and Zdražka 1986; Li, Pan, and Liang 2010; Pezzella and Merelli 2000) is also relevant to the CF-FJS problem.

The block in the CF-FSJ problem will be defined as a non-empty subsequence of operations of the critical path \( B^s = (O^s_{F_1}, O^s_{1}, O^s_{2}, \ldots, O^s_{f}, \ldots, O^s_{bs(s)-2}, O^s_{L}) \subseteq CP \), such that all the operations are processed on a single machine consecutively without idles and the predecessor of \( O^s_{F_1} \) as well as successor of \( O^s_{L} \) either does not exist or is processed on other machine. The blocks are indexed by \( s = 1, 2, \ldots, bn \) which denotes the number of the block on the critical path. The block \( B^s \) has \( bs(s) \) operations. The interior of the block \( B^s \) is defined as \( B^s = (O^s_{1}, O^s_{2}, \ldots, O^s_{f}, \ldots, O^s_{bs(s)-2}) \subseteq B^s \).

Let us select some block \( B^s \) from a semi-active solution \( \tilde{S} \) of the CF-FJS problem, such that \( bs(s) > 2 \) (i.e. \( B^s \) is non-empty). Let us modify the solution \( \tilde{S} \) by transposition of the operations from \( B^s \), according to a freely chosen non-identical permutation \( \sigma \in S_c \), at which \( c = bs(s) - 2 \). After the transposition, new constraints can be activated in the solution, such that idle intervals \( \Delta_f \geq 0 \), \( f = 1, 2, \ldots, c + 1 \) can appear between the operations previously belonging to \( B^s \) (Figure 3). Constructing of the new solution can be divided into two steps, resulting in the solutions \( S^s \) and \( S^{s^*} \), respectively. In the first step, the transposition is executed, but sequence constraints for the operations from \( B^s \) are omitted, thus \( \Delta_f = 0 \), \( f = 1, 2, \ldots, c + 1 \). In the second step, the additional constraints are taken into account.

After the first step, it is only assured that \( t_k(O^s_{s(1)}) = t_k(O^s_{1}) \), but start, completion and processing times of other operations from \( B^s \) can change. On the basis of the MFM formulation and (2a), the following integrals can be written for the operations from \( B^s \)

\[
\int_{t_k(0^s_{1})}^{t_k(0^s_{f})} e_p(\tau)/p_{s(1)} d\tau = 1 \quad \ldots \quad \int_{t_k(0^s_{f-1})}^{t_k(0^s_{f})} e_p(\tau)/p_{s(f)} d\tau = 1 \quad \ldots \quad \int_{t_k(0^s_{f})}^{t_k(0^s_{c+1})} e_p(\tau)/p_{s(c)} d\tau = 1.
\]

where \( f = 2, 3, \ldots, c - 1 \), the parameter \( e_p \) is the efficiency pattern of the machine \( M_p \) that processes the operations and \( p_{s(f)} \) substitutes \( p_{k,i,p} \), as the index \( s(f) \) unambiguously identifies an operation in the solution. The integrals can be multiplied by the constant parameters \( p_{s(z)} \) and summed as follows

\[
\int_{t_k(0^s_{1})}^{t_k(0^s_{f})} e_p(\tau) d\tau + \ldots + \int_{t_k(0^s_{f})}^{t_k(0^s_{f-1})} e_p(\tau) d\tau + \ldots + \int_{t_k(0^s_{f})}^{t_k(0^s_{c+1})} e_p(\tau) d\tau = \sum_{z=1}^{c} p_{s(z)}.
\]

The left-hand side of (8) can be simplified to a single integral

\[
\int_{t_k(0^s_{1})}^{t_k(0^s_{f})} e_p(\tau) d\tau = \sum_{z=1}^{c} p_{s(z)}.
\]

Figure 3. Permutation of operations inside a block.
Three neighbourhood functions are defined.

The scheduling algorithm used in this work is based on the tabu search metaheuristic (Barnes, Laguna, and Glover 1995). This metaheuristic is useful in optimisation problems for which a discrete neighbourhood is defined. The tabu search is quite broad, hence it is denoted as the highest level of search intensification (L3) and the additional levels are introduced according to CF-FJS Property 2 (Section 4.2) and CF-FJS Property 3 (Section 4.3). This neighbourhood transpired to be surprisingly efficient for solving the JS/FJS problems and, despite relatively simple implementation, it often yields gainful results (Watson et al. 2003). It can be easily improved by additional mechanisms which reduce neighbourhood and make computations faster (Grabowski and Wodecki 2005).

5. Tabu search-based scheduling algorithm

The scheduling algorithm used in this work is based on the tabu search metaheuristic (Barnes, Laguna, and Glover 1995). This metaheuristic is useful in optimisation problems for which a discrete neighbourhood is defined. The tabu search is surprisingly efficient for solving the JS/FJS problems and, despite relatively simple implementation, it often yields gainful results (Watson et al. 2003). It can be easily improved by additional mechanisms which reduce neighbourhood and make computations faster (Grabowski and Wodecki 2005). The mentioned advantages of the tabu search are utilised in the presented implementation.

5.1 Moves, neighbourhood and tabu attribute

In the tabu search algorithm, the same move definition is used as that introduced in the description of the property of critical path (Section 4.2).

For defining the neighbourhood, additional definitions are introduced:

- $M(B^s)$ denotes the machine $M_p \in M$ that processes the operations from the block $B^s$.
- $V_{k,i}(\pi, \tilde{B}^{Fs}_{FL})$ is the set of non-redundant moves that shuffle an operation $O_{k,i} \in B^P$ out of the block $B^s$. This set is empty if $bs(s) = 1$, has two elements if $bs(s) > 2$ and $O_{k,i} \in B^P$ has and one element otherwise.
- $RS(x)$ denotes random selection of exactly one element from a set or sequence $x$ with uniform distribution.

Three neighbourhood functions are defined:

$$N^{L1}(\pi) = \bigcup_{M_p \in M_{k,i}} \left\{ V_{k,i}(\pi, \tilde{B}^{Fs}_{FL}) \right\} \bigcup_{M_p \in M_{k,i}} \left\{ RS(V_{k,i}(\pi)) \right\} \quad \text{if } M_p = M(B^s), \quad \text{otherwise} \quad (10a)$$

$$N^{L2}(\pi) = \bigcup_{B^s \subseteq CP} \bigcup_{M_p \in M_{k,i}} \left\{ V_{k,i}(\pi, \tilde{B}^{Fs}_{FL}) \right\} \bigcup_{M_p \in M_{k,i}} \left\{ RS(V_{k,i}(\pi)) \right\} \quad \text{if } M_p = M(B^s), \quad \text{otherwise} \quad (10b)$$

$$N^{L3}(\pi) = \bigcup_{O_{k,i} \in CP} \left\{ V_{k,i}(\pi, \tilde{B}^{Fs}_{FL}) \right\} \bigcup_{O_{k,i} \in CP} \left\{ V_{k,i}(\pi) \right\} \quad \text{if } \exists \exists B^P \in CP O_{k,i} \in B^P, \quad \text{otherwise} \quad (10c)$$

The neighbourhood $N^{L3}(\pi)$ (10c) comprises all the moves that give the chance of immediate objective improvement, according to CF-FJS Property 2 (Section 4.2) and CF-FJS Property 3 (Section 4.3). This neighbourhood transpired to be quite broad, hence it is denoted as the highest level of search intensification (L3) and the additional levels are introduced by restriction of the set of moves from $N^{L3}$ using some random choices, such that $N^{L1}(\pi) \subseteq N^{L2}(\pi) \subseteq N^{L3}(\pi)$. In the neighbourhood $N^{L2}(\pi)$ only a single operation from every block is chosen randomly and moved just before and just after
After changing processing order of a feasible solution by a move an unfeasible solution can arise. The solution is unfeasible if and only if its solution graph is cyclic (Pinedo 2008). The unfeasibility can be detected while the solution graph is created in constant time with limited accuracy. Here, the method proposed by Grabowski and Wodecki (2005) is adapted to the CF-FJS problem.

Let us assume that an initial feasible solution \( \tilde{\mathbf{S}}_\pi \) is to be changed by the move \((k, i \rightarrow p, q)\). To perform the fast verification of feasibility of the target solution \( \tilde{\mathbf{S}}_{\pi'} \), where \( \pi' = \pi_{(k,i \rightarrow p,q)} \), the following data are needed:

1. \( \pi \), the initial processing order,
2. \( G(\pi) \), the acyclic solution graph based on the processing order \( \pi \),
3. \( H(n) = L(N_s, n) \), the function that returns length of the longest path between the nodes \( N_s \) and \( n \),
4. \( T(n) = L(n, N_c) \), the function that returns length of the longest path between the nodes \( n \) and \( N_c \),
5. \((k, i \rightarrow p, q)\), the move that is to be performed.

The data from the points (2), (3) and (4) can be derived with the complexity \( O(h) \). It is enough to obtain these data once for checking all moves that can modify the solution \( \tilde{\mathbf{S}}_\pi \).

On Figure 4, the modification in the solution graph is shown that represents the move \((k, i \rightarrow p, q)\). Three arcs from the graph \( G(\pi) \) are removed and three new arcs \( A_1, A_2, A_3 \) are inserted. The path \((u_B, x_A, x_B, v_A)\) is reinstated by the arc \( A_1 \) and the arc \((p_B, s_A)\) is reinstated by the path \((p_B, x_A, x_B, s_A)\). Hence, these reconstructed connections cannot close a cycle. However, the cycle can be created by the arc \( A_2 \) together with the hypothetically existing path \( d_1 \) or \( d_2 \), as well as by the arc \( A_3 \) together with the hypothetically existing path \( d_3 \) or \( d_4 \). If any of the paths \( d_1, \ldots, d_4 \) exists in the graph \( G(\pi) \) then the solution \( \tilde{\mathbf{S}}_{\pi'} \) is unfeasible.

Length of any directed path in a solution graph has to be non-negative. Hence, if the path \( d_1 \) exists the following holds

\[
T(\gamma(x)) \geq T(\beta^A(p)) + w(\gamma(x)). \quad (11)
\]

**Algorithm 2:** Fast feasibility detection (FFD).

**Data:** \( \pi, G(\pi) = (N, A, w), H(), T(), k, i, p, q \)

begin

[\[
\begin{align*}
g(x), p^A, a(p), \beta^A(p), a(s), s^A, \gamma(s), \delta^A(s) & := getAmbientNodes(\pi, G(\pi), k, i, p, q); \\
\psi() & := generateFunctionPsi(\pi, G(\pi)); \quad /\* \psi() returns k, i for a given node */ \\
k^p, i^p & := \psi(p^A); k^p, i^p & := \psi(\beta^A(p)); k^s, i^s & := \psi(s^A); k^g, i^g & := \psi(\gamma^A(s)); \\
/\* checking cycles based on re-circulated operations: */ \\
& \text{if } [k^p = k \land i^p > i] \lor [k^p = k \land i^p > i] \text{ then return false;}
& \text{if } [k^s = k \land i^s < i] \lor [k^g = k \land i^g < i] \text{ then return false;}
& \text{if } T(\gamma(x)) \geq T(\beta^A(p)) + w(\gamma(x)) \text{ then return false;}
& \text{if } T(\alpha(s)) \geq H(\gamma(s)) + w(\alpha(s)) \text{ then return false;}
& \text{if } T(\beta(s)) \geq H(\delta^A(s)) + w(\alpha(s)) \text{ then return false;}
& \text{return true; } \quad /\* \text{the target solution must be feasible */}
\end{align*}
\]

end

Thus, (11) is necessary for existing the path \( d_1 \) and the cycle closed by this path. Analogous conditions can also be formulated for the paths \( d_2, d_3 \) and \( d_4 \). These conditions are used in the FFD procedure (Algorithm 2).
This algorithm rejects all unfeasible moves and it also rejects some feasible moves, because the verification is based on the necessary condition of unfeasibility. The algorithm checks separately the cycles closed by re-circulated operations (re-circulation is not forbidden by the CF-FJS formulation), because the conditions based on the paths $d_1, \ldots, d_4$ are not applicable for them.

### 5.3 Algorithm implementation

Issues specific for the actual implementation of the algorithm are described below.

#### Length of the tabu list

Length of the tabu list $tl$ is delimited dynamically as follows

$$tl = \max \left\{ \min \left\{ \left\lfloor \frac{FMe}{tl^R} \right\rfloor, tl^{UB} \right\}, tl^{LB} \right\}, \quad FMe = \frac{\sum_{i=1}^{t^n} fmc_{(-i)}}{tl^n}.$$  

The value $fmc_{(-i)}$ (feasible move counter) represents the number of feasible moves from $i$th iteration backwards. The parameter $tl^*$ denotes previous length of the tabu list. Hence, the new length is a fraction of the mean number of feasible moves from last $tl^n$ iterations, with additional lower and upper bounds. The value $tl$ simultaneously defines that the new tabu list length will be used for next $tl$ iterations. When the algorithm starts, the tabu list has the initial length given by the parameter $tl^{INIT}$.

#### Level switch condition

This condition is the base of the SILS procedure. The algorithm starts with the search intensification level L1 (neighbourhood $N^{L1}$) and switches to the higher level when the ability of objective improvement expires for the given neighbourhood. The level switch condition from the level $lv$ to the higher one is derived as the following boolean parameter

$$lsc_{lv} = \left[ \left( n_{lv} > R_{lv}^{MIN} \right) \land \left( \frac{n_{lv}^{WI}}{n_{lv}} > S_{lv} \right) \right] \lor \left( n_{lv} > R_{lv}^{MAX} \right), \quad lv \in \{L1, L2, L3\},$$

where $n_{lv}$ is the number of all iterations in the current intensification level $lv$, $n_{lv}^{WI}$ is the number of last iterations without objective improvement, the parameters $R_{lv}^{MIN}, R_{lv}^{MAX} \in \mathbb{N}$ define absolute minimum and maximum number of iterations and $S_{lv} \in (0, 100\%)$ defines relative switching ratio.

#### Fast feasibility detection

The FFD procedure (Section 5.2) is used for elimination of unfeasible moves when neighbourhood is inspected. If a move is determined as unfeasible by the FFD it is dropped without further evaluation, however, its attribute is not added to the tabu list.

#### Aspiration criterion

The simple aspiration criterion is applied which accepts a forbidden move if it improves the global best result.

#### Stop condition

The stop condition is defined as the boolean parameter

$$sc = lsc_{L3} \lor \left( t \geq t^{MAX} \right), \quad (12)$$

![Figure 4. Moving an operation in the solution graph.](image-url)
where \( t \) is the algorithm running time. Thus, the algorithm stops when the last level of search intensification expires or the given maximum scheduling running time \( t_{MAX} \) is reached.

### 6. Simulation experiments

The algorithm parameters have been tuned experimentally. The following values have been used in all simulations:

- The parameters related to the tabu list length control: \( tLINT = 10, tL = 10, tLB = 5, tUB = 100 \).
- The parameters related to the intensification level switch conditions: \( R_{L1}^{MIN} = 500, R_{L2}^{MIN} = R_{L3}^{MIN} = 2,000, R_{L1}^{MAX} = R_{L2}^{MAX} = R_{L3}^{MAX} = 50,000, S_{L1} = S_{L2} = S_{L3} = 40\% \).
- The maximum running time: \( t_{MAX} = 1 \text{ h} \).

#### 6.1 Dedicated benchmarks

A set of ten benchmarks has been prepared to verify algorithms for solving the CF-FJS problem. For each benchmark instance, the number of jobs (from 30 to 150) and the number of machines (from 20 to 120) have been defined arbitrarily. Other parameters have been drawn from given ranges. Detailed benchmarks description and files with generated instances are available as supplemental online material (Appendix B) and also on the web page [http://cffjs.kia.prz.edu.pl](http://cffjs.kia.prz.edu.pl).

#### 6.2 Results

Scheduling algorithm with the primary stop condition (12) was executed 100 times for each benchmark instance. The set of solutions for the instance \( N \) is denoted as \( S_{prim}^N \) and the set of all solutions, that is \( \bigcup_{N=0}^{10} S_{prim}^N \), is denoted as \( S_{prim} \). One scheduling without time limitation (\( t_{MAX} \rightarrow \infty \)) was also executed for each instance. The results of this scheduling for the instance \( cff_{N} \) and for all instances are denoted as \( S_{wtl}^N \) and \( S_{wtl} \), respectively. Makespan and scheduling time collation are presented in Table 2. For each set \( S_{prim}^N \), the minimum (best), maximum, mean and standard deviation values of the results are calculated. The makespan values are well concentrated and maximum coefficient of variation (i.e. \( \frac{\text{stdev}}{\text{mean}} \)) equals 4.21% for \( cff_{04} \). The coefficient of variation of the scheduling time is more influential and reaches values up to 130% for \( cff_{01} \).

Comparison of the best makespan from \( S_{prim}^N \) and \( S_{wtl}^N \) presented in column Cmp (Table 2) is calculated according to the relation

\[
Cmp^N = \left( \frac{\min_{p \in S_{prim}^N} \{C_{max}(p)\}}{\min_{q \in S_{wtl}^N} \{C_{max}(q)\}} - 1 \right) \times 100\%.
\]

### Table 2. Makespan and computation time, results from \( S_{prim} \)

| Inst. | Time Limitation (\( S_{prim}^N \)) | No Time Lim. (\( S_{wtl}^N \)) | BKS |
|-------|-----------------------------------|--------------------------------|-----|
|       | Min | Max | Mean | Stdev | Min | Max | Mean | Stdev | Min | Cmp | Time |
| 01    | 433.6 | 468.8 | 453.1 | 7.0 | 2.8 | 60.0 | 7.6 | 9.9 | 457.7 | -5.3 | 4.4 | 433.6 |
| 02    | 590.1 | 610.5 | 600.1 | 4.1 | 2.8 | 38.6 | 6.3 | 5.3 | 595.5 | -0.9 | 8.4 | 590.1 |
| 03    | 399.3 | 422.7 | 409.0 | 4.7 | 3.6 | 60.0 | 11.9 | 10.9 | 415.0 | -3.8 | 5.8 | 399.3 |
| 04    | 486.1 | 575.4 | 517.1 | 21.8 | 0.8 | 5.2 | 2.1 | 0.8 | 509.4 | -4.6 | 1.8 | 486.1 |
| 05    | 602.1 | 632.2 | 614.8 | 7.5 | 10.5 | 60.0 | 40.0 | 18.0 | 607.6 | -0.9 | 65.1 | 602.1 |
| 06    | 655.2 | 676.4 | 665.1 | 4.2 | 18.3 | 60.0 | 57.8 | 6.9 | 653.7 | 0.2 | 350.5 | 653.7 |
| 07    | 881.4 | 945.9 | 898.2 | 10.6 | 38.7 | 60.0 | 59.7 | 2.3 | 865.5 | 1.8 | 353.1 | 865.5 |
| 08    | 388.5 | 399.3 | 394.0 | 2.1 | 45.4 | 60.0 | 59.6 | 2.1 | 387.6 | 0.3 | 150.5 | 387.6 |
| 09    | 503.0 | 541.4 | 517.7 | 8.6 | 60.0 | 60.0 | 60.0 | 0.0 | 485.1 | 3.7 | 320.0 | 485.1 |
| 10    | 941.8 | 968.4 | 956.2 | 5.5 | 60.0 | 60.0 | 60.0 | 0.0 | 852.7 | 10.5 | 1778.6 | 852.7 |

Mksp - makespan, Cmp - comparison, BKS - best known solution.
A. Bożek and M. Wysocki

It shows that the maximum scheduling time (1 h) limits quality of results for benchmark instances from cff_06 to cff_10. In the column BKS, the minimum makespan values are inserted, selected from $S_{N}^{prim}$ or $S_{N}^{wtl}$ for each instance, thus, they should be considered as the first best known solutions of the CF-FJS.

To verify efficiency of the SILS procedure, additional simulations have been performed. Instead of using level switching $L_1 \rightarrow L_2 \rightarrow L_3$, simulations have been run only with $L_1$, $L_2$ or $L_3$ separately, with 20 repetitions for each level. Other conditions have been determined by the primary algorithm configuration. The sets of solutions for the benchmark instance cff_$N$ are denoted by $S_{N}^{L_1}$, $S_{N}^{L_2}$, $S_{N}^{L_3}$ and for all instances by $S_{L_1}$, $S_{L_2}$, $S_{L_3}$. For these solutions, makespan statistics and makespan comparisons with reference solutions from $S_{prim}$ are prepared and presented in part (a) of Table 3. The comparisons are based on the relation

$$Cmp_{N}^{S_{N}^{prim}, S_{N}^{L_{f}}} = \left( \frac{fnc_{p \in S_{N}^{L_{f}}} \{ C_{max}(p) \}}{fnc_{q \in S_{N}^{prim}}} - 1 \right) \times 100\%, \quad fnc \in \{ \text{min, mean} \}, \quad S_{N}^{L_{f}} \in \{ S_{N}^{L_1}, S_{N}^{L_2}, S_{N}^{L_3} \}.$$  

The scheduling based only on $L_1$ gives definitely poor results. The solutions based on $L_2$ are closer to the reference makespan values. The results obtained with $L_3$ are very good for some benchmark instances, even slightly better than the reference values. However, they are the worst for other instances (cff_09, cff_10). The general finding is that the level switching $L_1 \rightarrow L_2 \rightarrow L_3$ is definitely better than using constant search intensification. The SILS gives propitious results for every cff instance, thus, the SILS improves scalability of the algorithm. This is evident from Figure 5. Comparisons of iteration numbers and simulation times with decomposition into three search intensification phases for solutions from $S_{prim}$ are shown in part (b) of Table 3. The data are expressed as mean absolute value and in percentage. The iteration number is most often the largest for $L_2$, but all levels for almost all benchmark instances are active at least by 10% of iterations. In the time domain, level $L_3$ dominates distinctly. This is because the iteration to time ratio ($I/T$) decreases largely for consecutive levels.

Simulations have also been performed to verify impact of the FFD procedure. The algorithm modified by deactivation of the FFD has solved each benchmark instance 20 times. The sets of solutions for the instance cff_$N$ and for all instances are denoted $S_{N}^{wffd}$ and $S_{wffd}$, respectively. The FFD has not been used for selection of feasible moves, but predictions of the FFD have been registered for statistic purposes. The results from $S_{wffd}$ are compared with reference solutions from $S_{prim}$. The results (Table 4) show that the FFD has classified moves as feasible or unfeasible in proportion about 1:2 for each problem instance (columns F and U). The procedure detects typically from 50 to 60% of actually feasible moves (column $F / S$). This reduces neighbourhood and shortens the time of a single algorithm iteration. The final effect is profitable, especially for larger instances of the scheduling problem, as the comparison in Table 4 shows, which is calculated according to the relation

**Figure 5.** Search intensification levels comparison.
The CF-FJS scheduling problem is a generalisation of the standard FJS. This fact has been used to perform an additional test of the implemented algorithm using the classic FJS benchmarks mk_01..mk_10 (Brandimarte 1993). Scheduling has been repeated 100 times for each benchmark. A set of results for the benchmark mk_n is denoted as \( S^N \). The results are shown in Table 5. The best known values of lower bound (LB) and upper bound (UB) of makespan are referred from the work of Geiger (2012). The comparison (Cmp) is calculated according to the relation

\[
Cmp^N = \left( \frac{\min_{p \in S^N_{mk}} \{C_{max}(p)\}}{UB^N} - 1 \right) \times 100\%.
\]

### 6.3 FJS benchmarks

The CF-FJS scheduling problem is a generalisation of the standard FJS. This fact has been used to perform an additional test of the implemented algorithm using the classic FJS benchmarks mk_01..mk_10 (Brandimarte 1993). Scheduling has been repeated 100 times for each benchmark. A set of results for the benchmark mk_n is denoted as \( S^N \). The results are shown in Table 5. The best known values of lower bound (LB) and upper bound (UB) of makespan are referred from the work of Geiger (2012). The comparison (Cmp) is calculated according to the relation

\[
Cmp^N = \left( \frac{\min_{p \in S^N_{mk}} \{C_{max}(p)\}}{UB^N} - 1 \right) \times 100\%.
\]
This paper presents the comprehensive study on the new scheduling problem, i.e. the CF-FJS. The solution comprises the

Inst. Makespan Cmp. $S_{wffd}$ vs. $S_{prim}$

| Inst. | Objective calculations | Percentage values | Makespan |
|-------|------------------------|-------------------|----------|
|       | F×$10^4$ | U×$10^4$ | S×$10^4$ | F* | U* | S* | F/S | Min | Mean | Std | Min ref | Mean ref | Min [%] | Mean [%] |
| 01    | 186 | 364 | 306 | 33.8 | 66.2 | 55.7 | 60.8 | 448 | 459 | 7.7 | 434 | 453 | 3.3 | 1.3 |
| 02    | 95 | 150 | 162 | 38.9 | 61.1 | 65.9 | 59.1 | 595 | 602 | 4.7 | 590 | 600 | 0.9 | 0.3 |
| 03    | 124 | 250 | 219 | 33.1 | 66.9 | 58.6 | 56.5 | 403 | 412 | 3.9 | 399 | 409 | 1.0 | 0.7 |
| 04    | 63 | 104 | 121 | 37.8 | 62.2 | 72.7 | 52.0 | 485 | 496 | 14.6 | 486 | 518 | -0.2 | -4.2 |
| 05    | 154 | 278 | 291 | 35.6 | 64.4 | 67.2 | 53.0 | 615 | 625 | 4.8 | 602 | 615 | 2.2 | 1.6 |
| 06    | 116 | 286 | 228 | 28.9 | 71.1 | 56.8 | 50.9 | 677 | 683 | 3.6 | 655 | 665 | 3.3 | 2.7 |
| 07    | 82 | 159 | 148 | 34.2 | 65.8 | 61.2 | 55.8 | 924 | 962 | 33.6 | 881 | 898 | 4.9 | 7.1 |
| 08    | 105 | 199 | 202 | 34.5 | 66.5 | 66.6 | 51.8 | 404 | 412 | 4.2 | 389 | 394 | 4.0 | 4.5 |
| 09    | 80 | 122 | 132 | 39.7 | 60.3 | 65.5 | 60.6 | 556 | 576 | 9.4 | 503 | 518 | 10.6 | 11.4 |
| 10    | 42 | 58 | 63 | 42.3 | 57.7 | 62.7 | 67.4 | 1035 | 1049 | 9.2 | 942 | 956 | 9.9 | 9.7 |

F - feasible; the moves classified as feasible by the FFD.
U - unfeasible; the moves classified as unfeasible by the FFD.
S - successful; the moves which were actually feasible.
ref - reference value from $S_{prim}$.
* percentage values related to F + U, i.e. the total number of objective calculations.

Table 5. FJS benchmarks.

| Inst. of mk | Min | Max | Mean | Stddev | Min | Max | Mean | Stddev | LB | UB | Cmp [%] |
|-------------|-----|-----|------|--------|-----|-----|------|--------|----|----|--------|
| 01          | 40  | 42  | 41.1 | 0.9    | 0.5 | 3.3 | 1.9  | 0.5    | 36 | 39 | 2.52  |
| 02          | 26  | 28  | 27.0 | 0.2    | 5.8 | 17.3| 11.1 | 1.5    | 24 | 26 | 0.0   |
| 03          | 204 | 204 | 204.0| 0.0    | 2.4 | 8.5 | 4.0  | 1.1    | 204| 204| 0.0   |
| 04          | 60  | 67  | 63.3 | 2.8    | 3.3 | 11.1| 5.5  | 1.6    | 48 | 60 | 0.0   |
| 05          | 173 | 177 | 173.8| 1.0    | 11.2| 73.5| 27.4 | 11.6   | 168| 172| 0.58  |
| 06          | 58  | 64  | 60.9 | 1.3    | 31.3| 326.5| 47.0 | 31.2   | 33 | 58 | 0.0   |
| 07          | 142 | 155 | 148.3| 2.7    | 35.6| 301.3| 57.3 | 34.6   | 133| 139| 2.16  |
| 08          | 523 | 523 | 523.0| 0.0    | 4.1 | 16.1| 10.6 | 3.6    | 523| 523| 0.0   |
| 09          | 307 | 315 | 307.3| 1.2    | 5.2 | 26.8| 13.1 | 4.2    | 299| 307| 0.0   |
| 10          | 201 | 210 | 205.2| 1.8    | 80.0| 403.3| 157.2| 80.9   | 165| 197| 2.03  |

LB - lower bound, UB - upper bound, Cmp - comparison.

The results are satisfactory. The upper bound has been reached for 6 of 10 instances and for other instances the relative aberration does not exceed 3%.

7. Conclusion
This paper presents the comprehensive study on the new scheduling problem, i.e. the CF-FJS. The solution comprises the original approach concerning overlapping in operations that does not depend on fixed overlapping proportions, but is based on dynamic relations represented by MFM. The two main concepts of the MFM formalism, i.e. the variable speed of processing and the continuous material flow lead to the pair of the CF-FJS features: position-dependent processing times and overlapping in operations. Instances of the CF-FJS problem can be flexibly parameterised and can represent e.g. hybrid systems with both batch and continuous operations.

The tabu search algorithm for the CF-FJS problem is also proposed and the results of computational experiments are presented. The efficient optimisation methods are used, based on critical path, block approach, switching of search intensification levels and fast feasibility forecasting. The authors’ contribution is to adapt these methods to the new scheduling problem and to embed them into the tabu search algorithm implementation.
The advantageous impact of the main auxiliary procedures, i.e. the SILS and FFD, has been verified by experiments. These procedures lead to better objective values obtained without meaningful increase of computational time cost. The SILS mechanism also enhances scalability of the algorithm. The scalability is improved by dynamic length of the tabu list as well. The evidence of the good scalability is that all benchmark instances from the cff and Brandimarte sets with very diversified sizes can be solved by the algorithm with fixed parameters and the values of the objective are well concentrated. It suggests no influential problems with blocking in searching cycles or in local optima.

The proposed solution can be used in real-world applications. The published benchmark instances and the presented optimisation results can be helpful in testing new scheduling algorithms for the CF-FJS.

Future work on the presented scheduling problem is intended. In the formulation of the CF-FJS problem, it is assumed that exactly one machine is chosen for each operation. However, because of continuous material flow, it is possible and may be advantageous in practice to distribute processing of a single operation over several machines. To introduce such an extension, material flow splitting and joining procedures have to be added to the CF-FJS. The splitting is ambiguous, because for each machine available for an operation, the decision has to be taken whether the machine should be involved in processing the operation and in what interval of time. It implies that the extended formulation will have greater computational complexity. The splitting and joining procedures will also make it possible to model of operations precedence other than linear in which, for example, a number of parallel processing paths inflow into a single machine. It is also planned to extend the problem formulation with additional parameters, important in practice, such as release times, set-up times and transport delays.

Acknowledgements
The authors thank the anonymous referees for their valuable comments and suggestions which helped to improve the article.

Supplemental data
Supplemental data for this article can be accessed here http://dx.doi.org/10.1080/00207543.2014.955925.

References
Alvarez-Valdez, R., A. Fuertes, J. M. Tamarit, G. Giménez, and R. Ramos. 2005. “A Heuristic to Schedule Flexible Job-shop in a Glass Factory.” European Journal of Operational Research 165 (2): 525–534.

Barnes, J. W., M. Laguna, and F. Glover. 1995. “An Overview of Tabu Search Approaches to Production Scheduling Problems.” In Intelligent Scheduling Systems, Vol. 3, Operations Research/Computer Science Interfaces Series. edited by D. E. Brown and W. T. Scherer, 101–127. Springer.

Bożejko, W., M. Uchoński, and M. Wodecki. 2010. “Parallel Hybrid Metaheuristics for the Flexible Job Shop Problem.” Computers & Industrial Engineering 59 (2): 323–333.

Brandimarte, P. 1993. “Routing and Scheduling in a Flexible Job Shop by Tabu Search.” Annals of Operations Research 41 (3): 157–183.

Doh, H.-H., J. M. Yu, J.-S. Kim, D.-H. Lee, and S.-H. Nam. 2013. “A Priority Scheduling Approach for Flexible Job Shops with Multiple Process Plans.” International Journal of Production Research 51 (12): 3748–3764.

Farughi, H., B. Y. Yegane, H. Soltanpanah, F. Zaheri, and F. Naseri. 2011. “Considering the Flexibility and Overlapping in Operation in Job Shop Scheduling Based on Meta-heuristic Algorithms.” Australian Journal of Basic and Applied Sciences 5 (11): 526–533.

Fattahi, P., F. Jolai, and J. Arkat. 2009. “Flexible Job Shop Scheduling with Overlapping in Operations.” Applied Mathematical Modelling 33 (7): 3076–3087.

Fattahi, P., and M. S. Mehrabad. 2009. “A New Approach in Job Shop Scheduling: Overlapping Operation.” Journal of Optimization in Industrial Engineering 2 (3): 25–32.

Fioudas, C. A., and X. Lin. 2004. “Continuous-time Versus Discrete-time Approaches for Scheduling of Chemical Processes: A Review.” Computers & Chemical Engineering 28 (11): 2109–2129.

Geiger, M. J. 2012. The Instances for the Flexible Job Scheduling Problem with Work Centers. Research Report. Helmut-Schmidt University, Hamburg, Germany.

Gomes, M. C., A. P. Barbosa-Póvoa, and A. Q. Novais. 2013. “Reactive Scheduling in a Make-to-order Flexible Job Shop with Re-entrant Process and Assembly a Mathematical Programming Approach.” International Journal of Production Research 51 (17): 5120–5141.

Grabowski, J., and M. Wodecki. 2005. A Very Fast Tabu Search Algorithm for Job Shop Problem. In Metaheuristic Optimization via Memory and Evolution. Vol. 30, Operations Research/Computer Science Interfaces Series, edited by C. Rego and B. Alidaee, 117–144. Springer.

Grabowski, J., E. Nowicki, and S. Zdrzala. 1986. “A Block Approach for Single-machine Scheduling with Release Dates and Due Dates.” European Journal of Operational Research 26 (2): 278–285.

Gröflin, H., D. N. Pham, and R. Bürgy. 2011. “The Flexible Blocking Job Shop with Transfer and Set-up Times.” Journal of Combinatorial Optimization 22 (2): 121–144.
Lei, D., and X. Guo. 2013. “Scheduling Job Shop with Lot Streaming and Transportation Through a Modified Artificial Bee Colony.” International Journal of Production Research 51 (16): 4930–4941.

Li, J.-Q., Q.-K. Pan, and Y.-C. Liang. 2010. “An Effective Hybrid Tabu Search Algorithm for Multi-objective Flexible Job-shop Scheduling Problems.” Computers & Industrial Engineering 59 (4): 647–662.

Mousakhani, M. 2013. “Sequence-dependent Setup Time Flexible Job Shop Scheduling Problem to Minimise Total Tardiness.” International Journal of Production Research 51 (12): 3476–3487.

Na, H., and J. Park. 2013. “Multi-level Job Scheduling in a Flexible Job Shop Environment.” International Journal of Production Research. doi:10.1080/00207543.2013.848487.

Nowicki, E., and C. Smutnicki. 1996. “A Fast Taboo Search Algorithm for the Job Shop Problem.” Management Science 42 (6): 797–813.

Perregaard, M., and J. Clausen. 1998. “Parallel Branch-and-bound Methods for the Job-shop Scheduling Problem.” Annals of Operations Research 83 (1–4): 137–160.

Pezzella, F., and E. Merelli. 2000. “A Tabu Search Method Guided by Shifting Bottleneck for the Job Shop Scheduling Problem.” European Journal of Operational Research 120 (2): 297–310.

Shaik, M. A., and C. A. Floudas. 2007. “Improved Unit-specific Event-based Continuous-time Model for Short-term Scheduling of Continuous Processes: Rigorous Treatment of Storage Requirements.” Industrial & Engineering Chemistry Research 46 (6): 1764–1779.

Shaik, M. A., C. A. Floudas, J. Kallrath, and H.-J. Pitz. 2009. “Production Scheduling of a Large-scale Industrial Continuous Plant: Short-term and Medium-term Scheduling.” Computers & Chemical Engineering 33 (3): 670–686.

Tay, J.-C., and N.-B. Ho. 2008. “Evolving Dispatching Rules using Genetic Programming for Solving Multi-objective Flexible Job-shop Problems.” Computers & Industrial Engineering 54 (3): 453–473.

Vilcot, G., and J.-C. Billaut. 2011. “A Tabu Search Algorithm for Solving a Multicriteria Flexible Job Shop Scheduling Problem.” International Journal of Production Research 49 (23): 6963–6980.

Watson, J.-P., J. C. Beck, A. E. Howe, and L. D. Whitley. 2003. “Problem Difficulty for Tabu Search in Job-shop Scheduling.” Artificial Intelligence 143 (2): 189–217.

Wodecki, M. 2009. “A Block Approach to Earliness-tardiness Scheduling Problems.” The International Journal of Advanced Manufacturing Technology 40 (7–8): 797–807.