Management and Production Engineering Review
Volume 7 • Number 1 • March 2016 • pp. 21–32
DOI: 10.1515/mper-2016-0003

OFF-LINE AND DYNAMIC PRODUCTION SCHEDULING – A COMPARATIVE CASE STUDY

Andrzej Bożek, Marian Wysocki

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

Corresponding author:
Andrzej Bożek
Rzeszów University of Technology
Department of Computer and Control Engineering
W. Pola 2, 35-959 Rzeszów, Poland
phone: (+48) 17 865-15-92
e-mail: abozek@kia.prz.edu.pl

Received: 2 January 2016
Accepted: 16 February 2016

Abstract
A comprehensive case study of manufacturing scheduling solutions development is given. It includes highly generalized scheduling problem as well as a few scheduling modes, methods and problem models. The considered problem combines flexible job shop structure, lot streaming with variable sublots, transport times, setup times, and machine calendars. Tabu search metaheuristic and constraint programming methods have been used for the off-line scheduling. Two dynamic scheduling methods have also been implemented, i.e., dispatching rules for the completely reactive scheduling and a multi-agent system for the predictive-reactive scheduling. In these implementations three distinct models of the problem have been used, based on: graph representation, optimal constraint satisfaction, and Petri net formalism. Each of these solutions has been verified in computational experiments. The results are compared and some findings about advantages, disadvantages, and suggestions on using the solutions are formulated.

Keywords
flexible job shop, lot streaming, off-line scheduling, dynamic scheduling, tabu search, constraint programming, dispatching rules, Petri nets, multi-agent systems.

Introduction

Development of manufacturing planning and scheduling algorithms and systems is often a challenging task. For instance, the scheduling problems typically belong to the NP-hard class of computational complexity. Attempting to extract and solve specific problems, majority of works ignore a comprehensive approach. As a result, these works are of minor practical importance, because the partial solutions have to be combined into a single one which becomes a new challenge itself and further research is needed before the implementation.

The main purpose of this work is to show that it is possible, using modern methods of computer science, to develop versatile solutions dedicated for manufacturing scheduling and control. Firstly, the generalized scheduling problem, potentially adaptable for many real-world manufacturing systems, is formulated. Secondly, solutions for a few important scheduling modes are developed and compared.

The considered scheduling problem is an extension of the well-known flexible job shop (FJS) [1]. The following features are additionally taken into account:

- Lot streaming, i.e., splitting lots into smaller parts (sublots) which can be processed and transported separately. The most general case of lot streaming is used, namely the one with variable sublots. In such a case, the sublot size can be different in consecutive operations of a job, thus the sublots have to be repacked between the operations. Because of the lot streaming and possibility of sublots intermingling, three variants of schedules are distinguished.

- Availability of machines is limited and defined by individual calendars.
- Sequence-dependent setup times are considered between the sublots belonging to different operations.
- Transport times depending on the pair of source and target machines are also taken into account.

The considered problem has been named as *flexible job shop with lot streaming and repacking* (LSR-FJS), because the lot streaming with variable sublots is the most characteristic extension used in its definition. The problem formulation is modeled after manufacturing organization in a fastener factory.

Two off-line scheduling algorithms based on different models and methods have been implemented, as well as two on-line scheduling algorithms that represent completely reactive and reactive-predictive modes. Each of the algorithms has been tested using the same benchmark instances which makes it possible to compare them in the aspect of the objective value. However, not only the objective value characterizes scheduling modes, hence other features of them have also been discussed. The work points the advantages and disadvantages of selected scheduling modes and methods on the basis of the concrete LSR-FJS problem. The results may be useful for choice of appropriate solutions for other manufacturing systems and may provide the suggestion about a favorable model and method.

Related work

Extended formulations of the FJS problem are considered by many authors. In some works, the features incorporated into the LSR-FJS definition are taken into account: lot streaming [2–5], limited machines availability [6–11], setup times [3, 5, 12–17], and transport times [3, 13, 17]. However, these extensions are most often considered separately. For example, only two of the mentioned works [3, 5] consider lot streaming and setup times simultaneously, whereas all the works considering limited machine availability do not involve any other extensions.

The work [18] is a detailed study of the lot streaming method, however it relates to the flow shop problem. There are three types of sublots defined: *equal* (all of the same size), *consistent* (the size of a sublot does not change in consecutive operations), and *variable* (the sublots which are not consistent – the most general type). Additionally, the lot streaming may be *without intermingling*, if all sublots of a single lot have to be processed one by one, or *with intermingling* in other case. Existing works consider the FJS extended by lot streaming with equal sublots [2, 4, 5] and sparsely with consistent ones [3].

The novelty of the LSR-FJS problem definition is that it includes all the considered extensions together and that it takes into account the lot streaming with variable sublots.

Four modes of scheduling are usually distinguished [19, 20]: *off-line* (also called as *predictive*), *completely reactive*, *predictive-reactive*, and *robust*. The completely reactive and predictive-reactive modes are often jointly referred to as *dynamic scheduling*. Algorithms supporting any of these modes are developed for the (flexible) job shop problem [6, 21, 22], however the off-line scheduling is the most popular.

Predictive scheduling algorithms dedicated for NP-hard problems are typically based on metaheuristics. In the case of the FJS, the tabu search (e.g. [12, 13]) and genetic algorithms (e.g. [3, 4, 6, 8, 14]) are probably the most popular methods. The mathematical programming in the form of the *mixed-integer linear programming* (MILP) [14, 15] is also often used. Recently, some *constraint programming* solvers [23, 24] have attained excellent performance in the FJS optimization.

Dispatching rules [19, 20] are the primary method of completely reactive scheduling. Dynamic scheduling solutions, especially the predictive-reactive ones, are not infrequently implemented as multi-agent systems (e.g. [21, 22]).

Two software tools have been used in this work for developing on-line scheduling solutions. CPN Tools [25] is the software supporting *hierarchical timed colored Petri nets* (HTCPN) formalism [26], useful for modeling and analysis of discrete time systems, successfully applied in many projects related to manufacturing modeling and scheduling (e.g. [27, 28]). JADE (*Java Agent DEvelopment Framework*) [29] is a comprehensive multi-agent platform. It has been used in many projects involving production scheduling, planning and control (e.g. [30, 31]).

In the works related to the extended FJS problem, single-objective optimization is typically considered. The makespan is the most often selected objective, e.g., in many of the works mentioned above [3, 6–8, 11–13, 17, 22, 24, 27, 31]. Criteria based on tardiness (average [2], total [15], or total weighted [14, 21]) are used as well. However, the solutions involving multi-objective optimization are also developed. The composition of the makespan, total workload, and workload of the critical machine is probably the most popular objective in this case [5, 9, 10]. Sometimes, a greater number of criteria are combined [5]. The Pareto front [5, 9] or linear combination [9,

10] of individual criteria are the methods commonly used in the multi-objective scheduling. The multi-objective approach has a large practical importance. Nevertheless, it is usually omitted in the works involving extended FJS, especially when a new problem formulation is considered first time, because of the formulation complexity. It is also the reason why the makespan has been chosen in this work. However, scheduling of the LSR-FJS problem using multi-objective approach is planned as a future work.

LSR-FJS problem

The LSR-FJS problem (Fig. 1) is defined as follows:

- A set of n jobs $\mathbf{J} = \{J_k : k \in \{1, \ldots, n\}\}$ is given. Each job J_k is defined by the sequence of operations $\mathbf{O}_k = \{O_{k,i} : i \in \{1, \ldots, |\mathbf{O}_k|\}\}$. The size $e_k \in \mathbb{N}$ of each job J_k is also given and represents the number of identical elements that have to be processed in each operation assigned to the job.
- A nominal sublot size $d_{k,i} \in \mathbb{N}$, $d_{k,i} \leq e_k$ is defined for each operation $O_{k,i}$. The operation is split into $g_{k,i}$ sublots, where $g_{k,i} = \lceil e_k / d_{k,i} \rceil$, and each sublot has $d_{k,i}$ elements, except a single one, which contains the rest of elements and has the size of $e_k - (g_{k,i} - 1)d_{k,i}$.
- A set of r machines $\mathbf{M} = \{M_p : p \in \{1, \ldots, r\}\}$ is given. For each machine an availability calendar $\mathbf{C}_p$ is defined. The calendar describes consecutive time intervals in which the machine is available and unavailable alternately. Machine unavailability pauses setup and processing activities performed by it.
- A non-empty set $\mathbf{M}_{k,i}$ is assigned to each operation which includes the machines that can process elements in this operation.
- The processing time of a single element in the operation $O_{k,i}$ on the machine $M_p \in \mathbf{M}_{k,i}$ is represented by the parameter $p_{k,i,p}$, hence, the processing time of a sublot of size d equals $d \cdot p_{k,i,p}$.
- The setup time needed for the machine M_p to changeover from processing the operation $O_{k1,i1}$ to the operation $O_{k2,i2}$ is defined as $s_{p,k1,i1,k2,i2}$.
- The parameter $v_{p1,p2}$ represents the time of transport of any set of elements from the machine M_{p1} to the machine M_{p2}.
- Any sublot is processed as a whole, which means that all the elements assigned to the sublot have to be transported to the target machine before the processing starts and none of the elements can be taken away from the machine before the processing finishes.
- Any machine can be involved in at most one setup or processing activity at time.
- All kinds of activities, i.e., transport, setup and processing, have to be performed on a sublot without interruption. However, the pauses caused by a machine unavailability are not considered to be the interruptions.
- The objective is to minimize the makespan ($C_{\max}$), i.e., the finish time of processing of the last sublot.

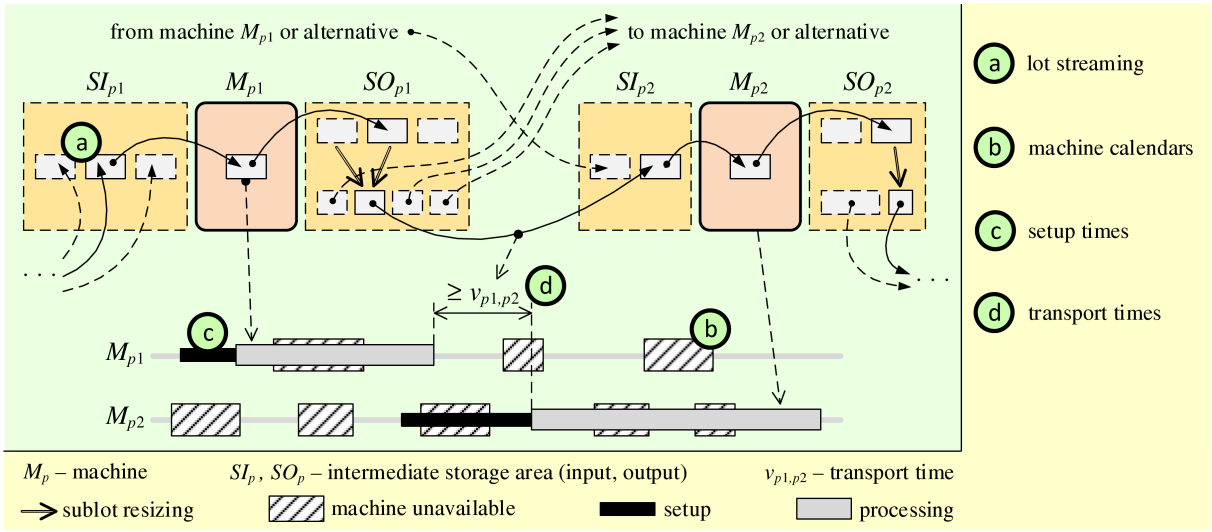

Fig. 1. Model of the LSR-FJS scheduling problem.

The above given problem statement does not force any concrete assignment of elements to sublots, except that the sublot sizes are imposed. Therefore, the scheduling method and algorithm have to determine the assignment of elements to sublots, the assignment of sublots to machines, and sequence of processing the sublots on every machine. These decisions represent a solution unambiguously, provided that a left-shifted schedule is considered.

Benchmark instances

Parameters of the LSR-FJS problem benchmark instances are presented in Table 1. Except for the number of jobs $n = 100$ and machines $r = 50$, all values have been generated randomly using uniform discrete probability distribution and given min/max limits. The random selection of the parameter values has been performed once and the obtained values have been saved to the benchmark files. Thus, the benchmarks represent completely deterministic instances. The first subset of the parameters (nos. 1–6 in Table 1) represents quantities, and the second one (nos. 7–11) concerns time intervals. The values have been chosen that approximately reflect features of the selected real-world production environment, i.e., the fastener factory. Jobs consist of 3 to 7 operations. The number of parallel machines available for an operation varies from 4 to 8. There are between 80 and 120 elements processed in a job, and between 10 and 30 elements in a sublot. This implies that the expected value of the number of sublots per operation is 5, whereas minimum is 3 and maximum is 12. The time parameters should be interpreted as expressed in seconds. Therefore, the transport time values vary from 15 to 25 minutes. The setup time has the values from 1 to 3 hours. The processing time related to a single element equals at least 7.5 minutes, and at most 22.5 minutes. A single interval of machine availability has the length between 14 and 18 hours. The lengths of unavailability intervals vary from 6 to 10 hours.

Table 1
Parameters of the benchmark instances.

No.	Parameter	Value		
		min	precisely	max
1	n		100	
2	r		50	
3	$\lvert\mathbf{O}_k\rvert$	3		7
4	$\lvert\mathbf{M}_{k,i}\rvert$	4		8
5	e_k	80		120
6	$d_{k,i}$	10		30
7	$v_{p1,p2}$	900		1500
8	$s_{p,k1,i1,k2,i2}$	3600		10800
9	$p_{k,i,p}$	450		1350
10	t^{ON}	50400		64800
11	t^{OFF}	21600		36000

The benchmark instance consisting of 100 jobs and based on the above mentioned parameters will be denoted as P100. Additionally, the second instance, P25, has been introduced by simple restriction of P100 to the first 25 jobs.

Assuming that the time parameters are expressed in seconds, the obtained makespan values are of the order of days and, therefore, a day will be used as the makespan unit in the remainder of the paper.

Scheduling approaches

The comparison of several scheduling approaches is the primary goal of this work. The approaches have been differentiated on the basis of two aspects, namely schedule variants and scheduling modes. The variants and modes are unrelated, thus any combination of a chosen variant and mode can be considered as a separate scheduling approach.

Schedule variants

In the case of lot streaming combined with flexible processing, a choice of intermingling configuration impacts on a schedule variant. In this work, three variants have been distinguished and denoted as A, B, and C (Fig. 2).

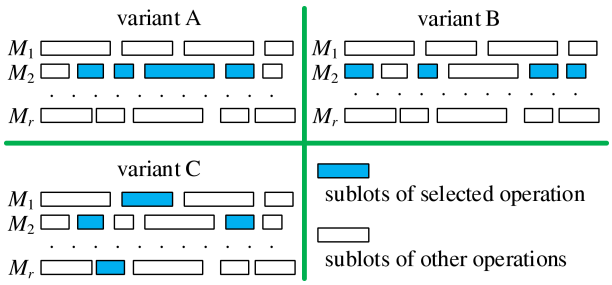

Fig. 2. Schedule variants.

Variant A represents the standard schedule without intermingling in which all sublots of any operation have to be processed one by one on a single machine. Variant B allows intermingling of sublots belonging to different operations, however, all sublots of each operation have also to be processed on a single machine. There are no specific restrictions in variant C, i.e., each sublot can be processed on any machine available for it and at any position of the processing queue.

The specification of the variants implies the following observations:

- The variants in the sequence C, B, A are characterized by increasing level of constraints. It is easy to see that the constraints impose the relation $\mathbb{S}_A \subseteq \mathbb{S}_B \subseteq \mathbb{S}_C$, where $\mathbb{S}_i$ is the solution space related to the variant i.
- The above given relation between solution spaces implies that $v_C \leq v_B \leq v_A$, where v_i denotes the optimal value of any minimized objective, in particular $C_{\max}$, related to the variant i. Therefore, the best solution is expected to exist in the variant C, but this variant has the largest solution space, hence, it is the most difficult to optimize.
- The technological and organizational aspects of manufacturing may force or exclude some of the variants.

The conclusion is that no variant dominates explicitly over others. For that reason, it is justified to implement and compare scheduling algorithms for all the variants.

Scheduling modes

Three modes of scheduling are considered in the work:

- Predictive scheduling in which all parameters are defined precisely and do not change during the scheduling process. In this mode, the manufacturing execution starts after the scheduling is finished, thus the scheduling algorithm can rearrange all planned activities.
- Completely reactive scheduling which is the opposition of the previous one. In this mode there is no plan generated in advance. Decisions are taken as the reaction to events. Typically, when a machine becomes free a next waiting activity (e.g. sublot processing) is chosen and assigned to it.
- Predictive-reactive scheduling, which is a kind of combination of the above mentioned modes, and inherits the advantages of the both. The algorithm constructs optimized partial schedule in advance for a limited time horizon. Additionally, the algorithm reacts to unpredictable events and repairs the partial schedule in case of disruption.

Scheduling methods

A few methods have been used for supporting the considered scheduling variants and modes. The tabu search metaheuristic and constraint programming have been chosen for the predictive scheduling. The completely reactive scheduling has been based on dispatching rules, whereas a multi-agent system has been developed for the predictive-reactive scheduling. The implementation of these methods is presented in dissertation [32] in details. Here, only essential description is given.

Tabu search

The tabu search metaheuristic, introduced by Glover [33], is a method of discrete optimization derived from the local search procedure. The main feature of it is the tabu list, which stores attributes of last visited solutions. The solutions having attributes on tabu list are forbidden. It prevents the search from becoming stuck at a local minimum.

The tabu search is known to be very efficient in scheduling problems, despite relatively simple implementation [34]. In order to use this metaheuristic, several components have to be prepared: a problem model, moves and neighborhood definitions, and the tabu search procedure itself.

The model in the form of a vertex-weighted directed graph has been used for the tabu search, which is the typical formalism of this method. There are a few subsets of vertices in the defined graph structure, one subset for each of the activities, i.e., processing, transport, and setup, as well as two special vertices: the source and the sink. A processing vertex represents processing of a single sublot. The number of sublots is determined by a scheduled instance and does not change during optimization, thus the subset of processing vertices is fixed in the graph model. The problem formulation does not define explicitly how to group elements in sublots. However, in order to define the model precisely, a specific grouping has been introduced. The sublots of each operation and elements in these sublots have been thought as numbered sequentially, i.e., if the last element of the i-th sublot has the number k, then the first element of the $(i{+}1)$-th sublot has the number $(k{+}1)$. Then, if sublots of the next operation of the same job have the size h, the j-th sublot consists of elements numbered from $1 + (j-1)h$ to jh, except for the last sublot, which can be smaller. This transport organization has also been applied in all other scheduling methods presented in this work. Under the given assumption, the set of transport vertices is also well defined by a problem instance and fixed in the graph model. A setup is performed once before all sublots of an operation if intermingling is forbidden (variant A), but in the other variants the number of setups depends on a specific solution, thus the subset of setup vertices is not fixed. In general, the construction of the graph has been divided into three phases:

- Instantiation of the elements that are independent on the solution variant and the solution itself, i.e., the processing and transport vertices.

- Insertion of the elements which depend only on the solution variant, among others: the arcs between processing and transport vertices (there are some redundant arcs in the variants A and B that can be left out, while it is impossible in the variant C), the setup vertices in the variant A (in the variants B and C these vertices depend on a solution).
- Instantiation of all other elements that depend on a concrete solution.

The first and the second phases are executed once when the algorithm starts. Modifications of a solution during the search process affects only the elements inserted in the third phase. Makespan is represented by the length of the longest path from the source to the sink in calculation which the weights of the processing and setup vertices are adjusted to represent the real activity duration on the basis of machine calendars.

Several types of moves have been defined:

- *Simple moves* relocate a single sublot from an initial position to another one.
- *Block moves* transfer a coherent block of sublots belonging to the same operation.
- *Near moves* change the position of an element or a block on the same machine.
- *Far moves* transfer an element or a block to other machine.

The neighborhood is a combination of the defined moves, adapted to a given variant. It is obvious, for example, that simple moves are inappropriate for the variant A. The moves relocate only sublots from the critical path, because it is well known that other transfers cannot improve the value of makespan.

Other aspects of the algorithm implementation are quite typical. Among others, the aspiration criterion has been introduced which accepts unconditionally all solutions better than the current global optimum. The variable length of tabu list has been used which is determined in proportion to the mean number of feasible solutions per iteration.

Constraint programming

Among the vast number of methods applicable for off-line scheduling the constraint programming has been chosen as the second method used in the work. The characteristic of this method is that the problem has to be explicitly described in terms of decision variables and its domains, as well as constraints. In this aspect, the method is similar to the another one, mathematical programming (for example MILP), which is very often used in scheduling. The constraint programming is less popular in scheduling of the JS and FJS problems and their extensions, however, the choice of this method is not motivated directly by its general features, but rather relates to the advantages of a specific software tool.

In this work, IBM ILOG CP Optimizer solver has been used. Two advantages of this tool are important. Firstly, it shows excellent efficiency in makespan optimization of the classic FJS problem. This has been proved by the results published by Quintiq [24] that are probably the best in comparison with any other announced results. Some of the results on the Quintiq list have been obtained directly by the IBM solver and it has been checked that this solver generates results close to the best ones also for other instances. There are many methods of solving CP problems [35] and implementation of an efficient solver is a hard task. Thus, it is valuable that the tool exists which has the good performance for the classic FJS and this allows to expect that results will also be satisfying in the case of the LSR-FJS problem. The second advantage of the IBM solver is the syntax and semantic extension which supports modeling of scheduling problems. Modeling of such problems using only scalar decision variables is sophisticated. A few different MILP models can be formulated even for the classic FJS [36]. Development of such a model for the LSR-FJS problem could be enormous challenge. Nevertheless, this model has been implemented with the support of the extended formalism of the IBM solver in a quite easy way.

An interval decision variable is the primary extension introduced in the IBM solver for modeling scheduling problems. This variable represents interval of time related to some activity. It is described mainly by *start* and *end times*, but also other parameters are defined, as *presence*, *length*, and *size*. High level constraints can be defined on the interval variables. For example, the *span constraint* indicates that some interval spans over all present intervals from a given set, i.e., it starts together with the first interval from the set and ends together with the last one. A considerable subset of the solver extensions accessible for modeling of scheduling problems has been exploited in this work, among others, *interval* (obligatory and optional types) as well as *interval sequence* decision variables, the constraints of types: *span*, *alternative*, *end before start*, *no overlap*. The machine calendars specific to the LSR-FJS problem have been transformed into *intensity functions*.

A separate CP model has been designed for each variant. Similarly as for the graph models, some elements of the CP models are common and others are dedicated for the individual variants.

Dispatching rules

Dispatching rules are commonly used method for the completely reactive scheduling. For each sublot a value is calculated, called priority. Whenever any machine becomes free, a sublot waiting for the machine with the highest priority is chosen for processing on it.

When the dispatching rules are used, the schedule execution is considered step by step and the priorities are dynamically determined and applied. Therefore, a simulation model of the problem is needed for implementation of dispatching rules and, more generally, for development of any dynamic scheduling method. In the work, such a model has been built using *hierarchical timed colored Petri nets* formalism [26]. Each of the specific formalism features has been widely exploited. The most important is the *timed* character of the formalism which makes it possible to represent time relations between events in the schedule. *Coloring* of the net means that tokens take values (colors) of defined types and these values are processed when transitions are executed according to some expressions. Coloring enhances expressiveness of the formalism significantly. For example, the calculation of priorities and selection of the highest one have been implemented in the form of an expression in the colored Petri net model. *Hierarchization* makes it possible to build a net by connecting separate modules using horizontal and vertical compositions. The model of the LSR-FJS problem has a three-level hierarchical structure. On the first (top) level two main modules are embedded, namely *production simulator* and *control rules*. The internal implementation of these modules is nested in the second level. In the case of the *simulator* module, some parts that represent, e.g., machine calendars and sublot resizing logic are additionally nested in the third level. The *simulator* module reflects indeed behavior of the production system described by a scheduling problem instance, therefore, it is the same for all configurations. On the other hand, the *control rules* module has a different implementation for each dispatching rule and schedule variant.

A few dispatching rules have been prepared:

- SET (*minimum setup time*). The highest priority is assigned to the sublot which needs the shortest setup time before processing.
- FIF (*FIFO order*). The priorities are determined according to the order of the entry of sublots into a machine processing queue – the higher priority is assigned to the sublot which is earlier in the queue.
- LIF (*LIFO order*). The inverse of the FIF rule – the higher priority is assigned to the sublot which is later in the queue.
- PES (*longest remaining processing time*, *pessimistic*). The priority of a sublot is equal to the total time needed for processing all remaining (i.e., not processed yet) sublots of the same job. Processing times are determined pessimistically.
- OPT (*longest remaining processing time*, *optimistic*). As in the case of the PES rule, but processing times are determined optimistically.
- AVG (*longest remaining processing time*, *average*). As in the case of the PES rule, but average values of processing times are used.

Each of the above mentioned rules has been implemented for each of the variants A, B and C. The three versions of the *longest remaining processing time* rule relate to the fact that sublots can be processed on different machines with various times and it is not known in advance which machine will be chosen, thus the cases of minimum (OPT), maximum (PES), and average (AVG) times have been distinguished. Execution of the rules is based on *non-delay* assignment, i.e., just when a machine becomes free the next waiting sublot can be and has to be chosen for processing on the machine.

Multi-agent system

Agents are units of a software system typically characterized by autonomy, intelligence, adaptation, co-operation [37] and reactivity [38]. The reactivity is the feature that makes an agent ready for receiving external messages and responding to them in a sufficiently short time. The autonomous agents perform their activities independently and asynchronously, but results of their work can be exchanged using co-operation mechanisms. The features of agents' reactivity and autonomy have strongly facilitated implementation of the scheduling system, supporting its reactive and predictive functions, respectively. Three types of agents have been defined in the multi-agent system architecture: *machine*, *coordinator* and *connector*.

One *machine agent* is instantiated for each machine defined in the scheduling problem. These agents concurrently perform two behaviors:

- Each *machine agent* stores on-going version of the global schedule and continuously executes the optimization procedure which is based on the tabu search algorithm implemented earlier for predictive scheduling. However, the set of moves is restricted in this case and the algorithm is allowed to relocate only the sublots assigned to the related machine. The algorithm can also take away sublots from other machine under special condi-

tions, but it never transfers a sublot to another machine. These restrictions reduce search space and make the agent possible to find new better solutions faster, thus, its responsiveness is improved. The agent does not modify the schedule itself, but sends change propositions to the *coordinator*.

- The agent receives propositions of the schedule modification prepared by other *machine agents* and evaluates them. As the result, a *vote* is generated in the form of a value normalized to the interval $[-1, 1]$.

In the both above mentioned cases the *machine agents* do not use the global makespan but the local objective function, defined as the end processing time of the last sublot assigned to the related machine.

The *coordinator agent* has specifically reactive implementation. It receives notices about changes of the production process state from the *connector* and dispatches them to the *machine agents*. When a new schedule is evolved, the *coordinator* delivers it to the *connector*. The *coordinator agent* is also responsible for marshalling of the schedule modifications procedure. It checks change propositions sent by *machine agents* against the global objective function and drops the propositions worsening a solution, whereas accepts the ones that improve the makespan. If a new proposed solution does not change the makespan, the *coordinator* organizes voting. Each *machine agent* is asked to evaluate the solution and to deliver a vote, then the coordinator takes a decision on the basis of the sum of votes.

The *connector agent* intermediates between the multi-agent system and the production environment. Its implementation has to be adjusted to an actual organization of a production system. In this work, a simulation model of production has been used. The *connector agent* scans the state of the production system and sends to it recent schedules periodically.

The multi-agent system has been implemented with the use of JADE platform. The HTCPN model of the LSR-FJS problem, the same as in the previous case of completely reactive scheduling, has been used for production process simulation. The system uses a *master plan* which is a feasible schedule, not necessarily optimized, pointing preliminary assignment of sublots to machines and processing order. The agents actively modify and optimize this plan using the following mechanism:

- the sublots scheduled to start in time not longer than t_b are blocked and cannot be further relocated,
- arrangement of n_s sublots following the blocked ones is controlled by the agents and optimized,
- when consecutive sublots become blocked, the new ones are imported from the master plan to the controlled set,

where t_b and n_s are parameters of the system.

The multi-agent system has been created only for variant B, because its implementation is more time-consuming in comparison with other methods and details are strongly related to the variant. Moreover, this method seems to be not very useful for variant A, because of remarkably limited control over a schedule when intermingling is forbidden.

Results

The combinations of modes, methods, models and problem variants for which computational experiments have been conducted are presented in Fig. 3.

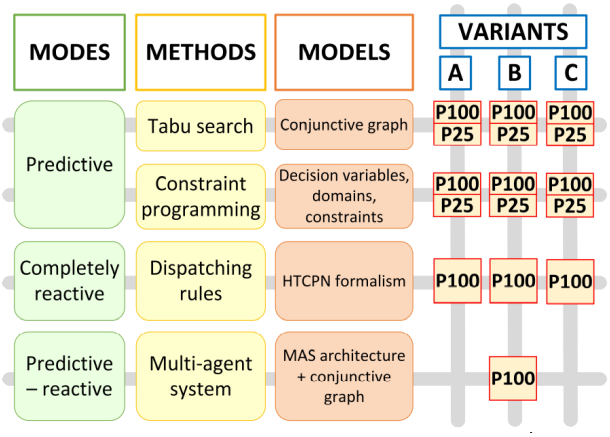

Fig. 3. Computational experiments scope overview.

In the case of the predictive scheduling methods, the experiments have involved the bigger instance P100 and the smaller one P25. It was needed, because the size of a problem instance affects the results of predictive scheduling significantly. On the other hand, the dispatching rules method uses negligible amount of computational resources during real-time execution, while the multi-agent system constructs a schedule for a limited number of sublots in advance. Therefore, the size of a problem instance does not have a major impact on efficiency of the considered dynamic scheduling methods. For that reason, only the instance P100 has been used to test them.

The comparison of the results obtained using predictive scheduling method is shown in Fig. 4. The time of algorithm execution was 24 hours in the case of P100 and 1 hour in the case of P25, for both methods and each variant. These times are a few or several times shorter than related makespan, hence correctly chosen for off-line scheduling.

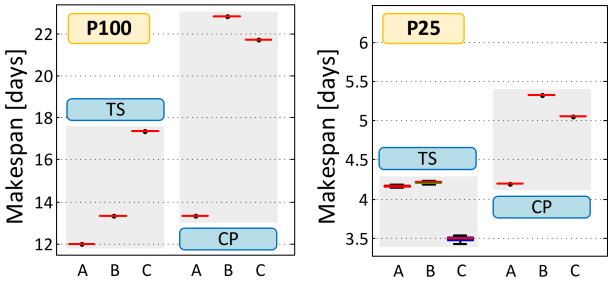

Fig. 4. Comparison of predictive scheduling results: TS – tabu search, CP – constraint programming.

The scheduling based on constraint programming has been executed once for each of the six configurations, because a deterministic algorithm is used in the solver and repetitions give the same results. The used tabu search implementation is also deterministic, but initial solutions for it were generated randomly, thus the final results are also conditioned randomly. For each variant of the instance P25 the tabu search algorithm has been repeated 10 times. The values of makespan obtained in these repetitions are very well concentrated (Fig. 4). The optimization of the instance P100 based on the tabu search has also been executed once for each variant, because of the long time of this procedure.

It is a curious result for the tabu search that the makespan is definitely the worst in variant C for P100 and definitely the best in the same variant for P25 (Fig. 4). It confirms the previously formulated hypothesis that it is difficult to point the most suitable variant in advance. One should rather execute optimization using all these variants and select the best result.

The effects of the makespan optimization are definitely worse in the case of the constraint programming (Fig. 4). The results are satisfactory and similar to those obtained by the tabu search only in variant A. In the other variants the makespan is unacceptable. The combinatorial complexity increased by introduction of intermingling in these variants has become too high for the solver.

The results of using of dispatching rules are shown in the form of a box plot in Fig. 5. Each dataset consists of 100 values obtained by repetitions of simulation for a given variant and rule. The results lead to the following findings:

- The makespan values are patently dispersed. It is the effect of partial indeterminism caused by random selection of a machine sequence in which the rules are applied if many machines are waiting for sublots. Nevertheless, in general, medians differ significantly and better or worse rules can be indicated.
- No rule prevails in all variants. In variant A rule PES is the best, but rules FIF, OPT and AVG give similar results. In variant B and especially in variant C rule SET dominates definitely.
- The worst results have been obtained in variant B. It is characteristic that the similar relation also exists in the case of majority of predictive scheduling results. Therefore, it seems to be pointless to extend the manufacturing organization from the model based on variant A to the one based on variant B considering use of the proposed dispatching rules.

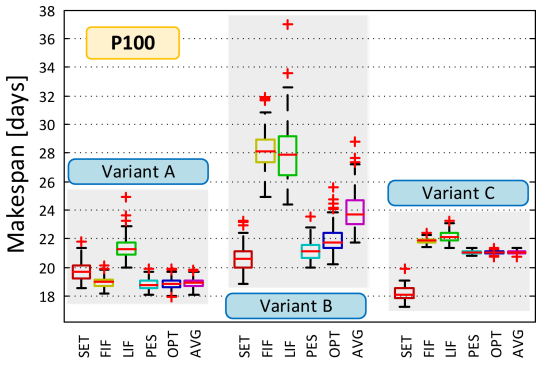

Fig. 5. Comparison of dispatching rules.

In Fig. 6 the results obtained using multi-agent system are presented and all results for the instance P100 are collected. The multi-agent system has been configured with the parameters: $t_b = 24$ h, $n_s = 500$. Four experiments have been conducted using different versions of the master plan:

- RAND – A randomly generated schedule with the makespan equal to about 32 days has been used as the master plan for dynamic scheduling.

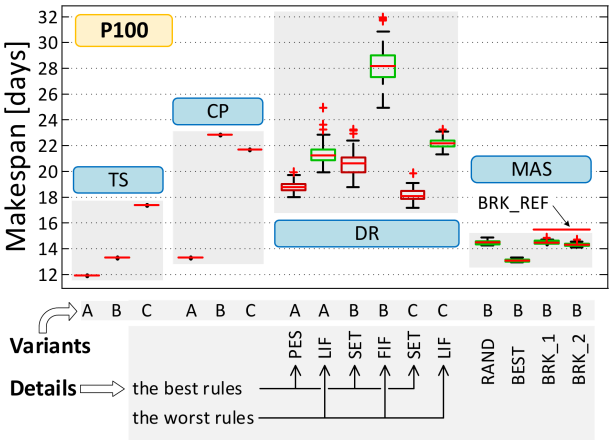

Fig. 6. Collection of results for the instance P100: TS – tabu search, CP – constraint programming, DR – dispatching rules, MAS – multi-agent system.

- BEST – The best solution obtained using off-line scheduling (tabu search, variant A) with the makespan equal to about 12 days has been used as the master plan.
- BRK_1 – The master plan in the form of the best solution has also been used, but the plan execution has been disturbed by additional machine unavailability intervals. This prolonged makespan of the original best schedule to the value BRK_REF (Fig. 6). The additional unavailability intervals have not been known for the multi-agent scheduling system.
- BRK_2 – The same configuration as in the case of BRK_1, but the additional unavailability intervals have been known for the scheduling system.

The experiments have been repeated 10 times for each version. It is evident that the results of predictive-reactive scheduling are, in general, very successful and close to the ones obtained by the tabu search (Fig. 6). Even if a schedule is not preliminarily optimized off-line, the multi-agent system achieves low values of the makespan (RAND), significantly smaller than in the case of the completely reactive mode. It proves that the predictive layer of the system works effectively. The system is also able to make use of an optimized master plan, as it points the comparison BEST vs RAND. Moreover, the system improves a schedule disrupted by disturbances (BRK_REF vs BRK_1 and BRK_2). This improvement is only slightly better when the system knows the disturbances in advance (BRK_1 vs BRK_2) which suggests that the reactive layer is of leading importance in this process.

Comparison of all results obtained for the instance P100 (Fig. 6) shows that the completely best makespan has been achieved by the tabu search in variant A. The effects of combinations of methods and variants (TS, B), (CP, A), (MAS, B) are only slightly worse. There are meaningful differences between the worst and the best dispatching rules used for each variant. In particular, the median makespan value which characterizes the best rule in variant C is similar to that obtained by the tabu search in the same variant, although the dispatching rule has the considerably simpler implementation and the advantage of reactivity. The best dispatching rules lead to definitely better results than constraint programming in variants B and C, so the used CP method seems to be practically useless in these cases.

Resume of scheduling methods

The comparison presented above involves only one aspect of the considered methods, namely the objective value. The findings should be supplemented by other observations.

The tabu search is a method suitable for predictive scheduling. It generally ensures very effective optimization which should be, however, verified in a concrete application. For example, the implementation used in this work has transpired to have different relative effectivity for tested instances and variants (Fig. 4). Moreover, the implemented algorithm is significantly better than dispatching rules in variants A and B, but not in variant C (Fig. 6). The repair of a schedule is time-consuming in the process of predictive scheduling, hence this mode is preferable in the manufacturing systems in which disturbance level is relatively low. It can also be used for generation of master plans dedicated for predictive-reactive production control systems.

The constraint programming, complemented with extensions for scheduling problem modeling, is also a powerful tool for implementation of predictive scheduling solutions. It is especially usable when a problem is characterized by a large set of constraints that are not known precisely in advance and may change in time. In such a situation the constraint programming model can be adjusted much more easily than a graph model. It is even possible to define a set of constraints configurable by the end user. The disadvantage of this method is, in turn, that the effectivity of optimization may be poor and it is worth comparing it with alternative methods before final implementation.

The dispatching rules, as the method of completely reactive scheduling, can be preferable in the systems characterized by high level of disruptions. In such a case the advantage of this method relates to the fact that no schedule is constructed in advance. This schedule would be damaged by disturbances, hence the unnecessary effort is omitted. Sometimes, the results of reactive and predictive scheduling may be similar, even if no disruptions are present, the tabu search and the rule SET applied to the instance P100 in variant C are the example (Fig. 6).

The implemented multi-agent system combines predictive and reactive functionalities and their advantages. It can operate standalone as a comprehensive scheduling module, however, the master plan can be additionally optimized off-line at first to further improve the result.

Conclusion

In this work, different manufacturing scheduling approaches have been compared. The results can be summarized by several major findings:

- The tabu search algorithm provides very high performance in optimization of complex scheduling

problems. Its implementation requires preparation of a detailed graph model of the problem which is not trivial. Nevertheless, the overall implementation is quite simple and even typical basic configuration often leads to satisfactory effects. Therefore, the tabu search is proposed as an initial method for research with a new scheduling problem and as a source of referential results.

- The performance of different dispatching rules is highly diversified. It suggests that intensified research is needed for looking for the best rules. The prepared HTCPN model of the LSR-FJS scheduling problem makes it possible to develop and test dispatching rules in an easy way. The simulation problem model can be quickly modified or extended. The restrictions which impose a specific scheduling variant or a concrete rule may be adjusted by net substructures inserted into the model. Future work is planned, based on the developed HTCPN model, in which more sophisticated composite dispatching rules will be prepared and configured by an optimization algorithm, e.g., genetic programming. The goal of this research is to verify how effectively the advanced dispatching mechanisms would be able to perform reactive control in comparison with the simple rules developed so far.
- The implemented multi-agent system has revealed excellent performance in predictive-reactive scheduling. On the one hand, the results of scheduling are close to that obtained by off-line optimization, although the system works in real-time and controls only a subset of sublots at any moment. On the other hand, the reactive action is evident and considerable, as the system repairs disturbed solutions. Even though it does not know disruptions in advance, it detects them in real-time. Because of the successful implementation, further work is planned. The robustness of the system to various disturbances is going to be investigated more extensively. It is also planned to verify the algorithm performance using scheduling problem instances from a real-world manufacturing system.

The work provides suggestions about method selection and algorithm implementation for production scheduling, taking into account the most popular scheduling modes. The proposed solutions have been verified on the LSR-FJS problem. This problem combines flexible job shop structure, lot streaming with variable sublots, transport times, setup times, and machine calendars. Therefore, this formulation is very versatile and can be adapted to many real-world cases. Majority of the findings presented in this work are, however, not related to very specific features of the LSR-FJS problem and they will also be true for other problems, if appropriate models are developed for them.

References

[1] Pinedo M.L., *Scheduling. Theory, Algorithms, and Systems*, Springer-Verlag, New York, 2012.

[2] Calleja G., Pastor R., *A dispatching algorithm for flexible job-shop scheduling with transfer batches: an industrial application*, Prod. Plan. Control, 25, 2, 93–109, 2014.

[3] Defersha F.M., Chen M., *Mathematical model and parallel genetic algorithm for hybrid flexible flow-shop lot streaming problem*, Int. J. Adv. Manuf. Tech., 62, 1, 249–265, 2012.

[4] Demir Y., İşleyen S.K., *An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations*, Int. J. Prod. Res., 52, 13, 3905–3921, 2014.

[5] Jun-jie B., Yi-guang G., Ning-sheng W., Dun-bing T., *An Improved PSO Algorithm for Flexible Job Shop Scheduling with Lot-Splitting*, International Workshop on Intelligent Systems and Applications, 2009.

[6] Al-Hinai N., ElMekkawy T.Y., *Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm*, Int. J. Prod. Econ., 132, 2, 279–291, 2011.

[7] Hasan S.M.K., Sarker R., Essam D., *Genetic algorithm for job-shop scheduling with machine unavailability and breakdowns*, Int. J. Prod. Res., 49, 16, 4999–5015, 2011.

[8] He W., Sun D.-h., *Scheduling flexible job shop problem subject to machine breakdown with route changing and right-shift strategies*, Int. J. Adv. Manuf. Tech, 66, 1, 501–514, 2013.

[9] Li J.-Q., Pan Q.-K., Tasgetiren M.F., *A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities*, Appl. Math. Model., 38, 3, 1111–1132, 2014.

[10] Wang S., Yu J., *An effective heuristic for flexible job-shop scheduling problem with maintenance activities*, Comput. Ind. Eng., 59, 3, 436–447, 2010.

[11] Zribi N., Borne P., *Hybrid Genetic Algorithm for the Flexible Job-Shop Problem Under Maintenance Constraints*, Advances in Natural Computation, Wang L., Chen K., Ong Y. [Eds.], LNCS 3612, Springer Berlin Heidelberg, 259–268, 2005.

[12] Bożejko W., Uchroński M., Wodecki M., *Multi-machine scheduling problem with setup times*, Archives of Control Science, 22, 4, 441–449, 2012.

[13] Gröflin H., Pham D.N., Bürgy R., *The flexible blocking job shop with transfer and set-up times*, J. Comb. Optim., 22, 2, 121–144, 2011.

[14] Kunadilok J., *Heuristics for Scheduling Reentrant Flexible Job Shops with Sequence-dependent Setup Times and Limited Buffer Capacities*, PhD Thesis, Clemson University, 2007.

[15] Mousakhani M., *Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness*, Int. J. Prod. Res., 51, 12, 3476–3487, 2013.

[16] Rohaninejad M., Kheirkhah A., Fattahi P., *Simultaneous lot-sizing and scheduling in flexible job shop problems*, Int. J. Adv. Manuf. Tech., 78, 1, 1–18, 2015.

[17] Rossi A., *Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships*, Int. J. Prod. Econ., 152, 253–267, 2014.

[18] Sarin S.C., Jaiprakash P., *Flow Shop Lot Streaming*, Springer US, 2007.

[19] Aytug H., Lawley M. A., McKay K., Mohan S., Uzsoy R., *Executing production schedules in the face of uncertainties: A review and some future directions*, Eur. J. Oper. Res., 161, 1, 86–110, 2005.

[20] Ouelhadj D., Petrovic S., *A survey of dynamic scheduling in manufacturing systems*, J. Sched., 12, 4, 417–431, 2009.

[21] Liu N., Abdelrahman M., Ramaswamy, S., *A Complete Multiagent Framework for Robust and Adaptable Dynamic Job Shop Scheduling*, IEEE Trans. Syst., Man, Cybern., Part C: Applications and Reviews, 37, 5, 904–916, 2007.

[22] Lou P., Liu Q., Zhou Z., Wang H., Sun S.X., *Multi-agent-based proactive–reactive scheduling for a job shop*, Int. J. Adv. Manuf. Tech., 59, 1–4, 311–324, 2012.

[23] Laborie P., *IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems*, Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Hoeve W.-J., Hooker J.N. [Eds.], LNCS 5547, Springer Berlin Heidelberg, 148–162, 2009.

[24] Quintiq. World records: Flexible job shop scheduling problem, http://www.quintiq.com/optimization/fjssp-world-records.html [accessed on December 14, 2015].

[25] CPN Tools Homepage, http://cpntools.org, [accessed on December 14, 2015].

[26] Jensen K., Kristensen L.M., *Coloured Petri Nets – Modelling and Validation of Concurrent Systems*, Springer-Verlang Berlin Heidelberg, 2009.

[27] Zhang H., Gu M., Song X., *Modeling and Analysis of Real-Life Job Shop Scheduling Problems by Petri nets*, Simulation Symposium, ANSS 2008, 41st Annual, 279–285, 2008.

[28] Aized T., *Modelling and performance maximization of an integrated automated guided vehicle system using coloured Petri net and response surface methods*, Comput. Ind. Eng., 57, 3, 822–831, 2009.

[29] JADE. Java Agent Development Framework, http://jade.tilab.com, [accessed on December 14, 2015].

[30] Leitão P., Restivo F., *ADACOR: A holonic architecture for agile and adaptive manufacturing control*, Comput. Ind., 57, 2, 121–130, 2006.

[31] Nouiri M., Bekrar A., Jemai A., Niar S., Ammari A.C., *An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem*, J. Intell. Manuf., 1–13, 2015 [published online, DOI: 10.1007/s10845-015-1039-3].

[32] Bożek A., *Using Petri nets, multi-agent techniques and computational intelligent methods in production planning and control* (in Polish), PhD thesis, Rzeszów University of Technology, 2015.

[33] Glover F., *Artificial intelligence, heuristic frameworks and tabu search*, Managerial & Decision Economics, 11, 365–378, 1990.

[34] Watson J.-P., Beck J.C., Howe A.E., Whitley L.D., *Problem difficulty for tabu search in job-shop scheduling*, Artif. Intell., 143, 2, 189–217, 2003.

[35] Rossi F., van Beek P., Walsh T. [Eds.], *Handbook of Constraint Programming*, Elsevier, Pisa, Italy, 2006.

[36] Demir Y., İşleyen S.K., *Evaluation of mathematical models for flexible job-shop scheduling problems*, Appl. Math. Model., 37, 3, 977–988, 2013.

[37] Leitão P., *Agent-based distributed manufacturing control: A state-of-the-art survey*, Eng. Appl. Artif. Intel., 22, 7, 979–991, 2009.

[38] Sycara K.P., *Multiagent systems*, AI Magazine, 19, 2, 79–92, 1998.