Parallel Simulated Annealing Algorithm for Cyclic Flexible Job Shop Scheduling Problem

Wojciech Bożejko[1(✉)], Jarosław Pempera[1], and Mieczysław Wodecki[2]

[1] Department of Automatics, Mechatronics and Control Systems, Faculty of Electronics, Wrocław University of Technology, Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland
{wojciech.bozejko,jaroslaw.pempera}@pwr.edu.pl
[2] Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland
mieczyslaw.wodecki@ii.uni.wroc.pl

Abstract. This paper deals with scheduling of tasks in cyclic flexible job shop scheduling problem (CFJSSP). We have proposed a new method of computing cyclic time for CFJSSP. This method is based on the known properties of the job shop problem as well as new properties of cyclic scheduling. We have developed two versions of proposed method: sequential and parallel. The parallel version is dedicated to the computing devices supporting vector processing. Finally, we have developed double paralyzed simulated annealing algorithms: fine grained - vector processing, multiple walk - multi core processing. Computation results, provided on market multicore processors, are presented for a set of benchmark instances from the literature.[1]

1 Introduction

Currently, in the vast majority of production systems there are multifunctional machines used that are configured and controlled remotely not only by industrial information systems but also by electronic drivers. Machines versatility helps in the implementation of a number of stages in the process of products manufacturing on the same machine or with the use of multiple machines which perform the most time-consuming production steps. This type of feature is called flexibility of manufacturing systems. The high flexibility of production systems supported by electronic exchange of information enables the use of advanced methods of production systems management (kanban, lean manufacturing) which adjust the schedule of the tasks execution to the needs of customers while reducing storage costs and work in progress.

Due to the high complexity of flexible manufacturing systems the efficient scheduling at the operational levels has significant importance. Operational planning guarantees conflict-free production and not only enables reduction of production costs but also increases the efficiency of the production system due to

[1] The work was supported by the OPUS grant DEC-2012/05/B/ST7/00102 of Polish National Centre of Science.

L. Rutkowski et al. (Eds.): ICAISC 2015, Part II, LNAI 9120, pp. 603–612, 2015.
DOI: 10.1007/978-3-319-19369-4_53

the application of optimization algorithms. Both of these features enhance the economic efficiency of enterprises because the production of computer support systems, in particular, optimizing the operational level are subjects of interest of many practitioners.

Even the simplest flexible manufacturing systems generate NP-hard optimization problems. For this reason, researchers focused attention on the development of heuristic algorithms based on local search methods. Among a wide variety of algorithms for flexible job shop systems, the best algorithms are based on tabu search methods: Hurink, Jurish and Thole [9], Mastrolilli and Gambardella [14]. Due to the current tendency of boosting performance by increasing the number of processing units, population algorithms, that can be in a relatively simple way parallelized, are gaining importance. Examples of such algorithms are: genetic algorithm (Yang Kacem and Borne [11]), particle swarm algorithm with simulated annealing search method (Xia and Wu [16]), genetic algorithm combined with the search algorithm with a variable environment (Jie Linyan oraz Mitsuo [10]). Dedicated parallel algorithms were proposed by Bożejko [4] and Bożejko et al. [2,3].

In many real manufacturing systems, there is a cyclic production strategy used. Cyclic manufacturing simplifies the logistics chain management for the production supplying distribution process with finished products. Scheduling of operations in such systems is still a challenge for researchers. This challenge particularly concerns the development of computational models and optimization algorithms.

The most general models of cyclic systems and detailed models for selected production systems were collected by Kampmeyer [12]. Kampmeyer and Brucker [6] used an algorithm based on tabu search method for cyclic job shop problem with no storage constraints, whereas neural networks which optimize the cycle time in job shop problem were used by Kechadi et al. [13].

2 Cyclic Flexible Job Shop Scheduling Problem CFJSP

A flexible job shop production system consists of m multifunction machines from the set of $M = \{1, ..., m\}$. In the production system the set of n tasks from the set $J = \{J_1, ..., J_n\}$ must be performed infinite number of times. Task $J_i \in J$ consists of n_i operations from the set of $O_i = \{(i,1),(i,2),...,(i,n_i)\}$. The set J consists of $o = \sum_{J_i \in J} n_i$ technological operations. For each operation $(i,k) \in O_i$, $i = 1, ..., n$, $k = 1, ..., n_i$ there is assigned a set of machines $M_{ik} \subseteq M$ on which it may be executed. If $M_i = M$ for every $J_i \in J$, then the production system is called fully flexible. Operation (i,k) is performed on the machine $l \in M_{ik}$ in $p_{ikl} > 0$ time. Each machine can perform only one operation at a time. At any given time only one operation from the task can be executed. Operations are performed on the machines continuously without interruptions.

In the cyclic production systems tasks are performed in the so-called production cycles. In one cycle, all tasks from the set of tasks J are performed. The order of operations execution on the machines in the first production cycle is reproduced in subsequent cycles.

Let $\pi_l = ((j_l(1), k_l(1)), ..., (j_l(n_l), k_l(n_l)))$ be a permutation of determining the order of operations on the machine $l \in M$, where n_l denotes the number of operations executed on machines. The set $\pi = (\pi_1, \ldots, \pi_{n_l})$ describes the sequence of operations for all machines in the production system. Note that π unambiguously describes the assignment of operations to machines. Let S_{ik}^x (C_{ik}^x) be the moment of starting (completion) of the performance of the k-th operation of J_i task in x-th $x = 0, 1, ...$ production cycle. The schedule of execution of operations in each production cycle must comply with the requirements arising from technological route and the order of operations on the machines π. Technological requirements for the tasks performed in x-th $x = 0, 1, ...$ production cycle can be formally described with the inequality:

$$S_{i,k}^x \geq C_{i,k-1}^x, J_i \in J,\ k = 2, ..., n_l, \tag{1}$$

whereas executing of operations in the cycle, in the order of π require the fulfilment of inequality:

$$S_{j_l(s),k_l(s)}^x \geq C_{j_l(s-1),k_l(s-1)}^x,\ l \in M,\ s = 2, ..., n_l, \tag{2}$$

which means that the start of execution of s-th, in the order of π_l, operation $(j_l(s), k_l(s))$ can only take place after the end of the previous operation $(j_l(s-1), k_l(s-1))$ performed on the machine l.

In addition, the schedule of operations execution for two consecutive production cycles $x-1$ and x, $x = 1, 2, ...$ must fulfill the following conditions:

$$S_{j_l(1),k_l(1)}^x \geq C_{j_l(n_l),k_l(n_l)}^{x-1},\ l \in M. \tag{3}$$

Due to the fact that operations are executed in a production system without interruptions the starting and completion of operations have the following relationship:

$$C_{ik}^x \geq S_{ik}^x + p_{ik,\mu_{ik}},\ J_i \in J,\ k = 1, ..., n_i, \tag{4}$$

where μ_{ik} denotes, resulting from π, the machine assigned to operation (i, k), $\mu_{ik} = l$ dla $(j_l(s), k_l(s))$, $s = 1, ..., n_l$, $l \in M$.

The schedule of operations execution in the production system is called *cyclic* if the following condition is met:

$$S_{ik}^x = S_{ik}^0 + \tau \cdot x \quad J_i \in J,\ k = 1, ..., n_i,\ x = 1, 2, \ldots, \tag{5}$$

where τ is a period called cycle time.

The order of operations execution in a cyclical system π is feasible if there is a solution to the inequality (1–5).

Let us denote by $\tau(\pi)$ the smallest value of the cycle time for the feasible order of π. The problem to be found is a sequence of operations execution on the machines π^* such that

$$\tau(\pi^*) = \min_{\pi \in \Pi} \tau(\pi), \tag{6}$$

where Π is the set of all allocations of operations to machines and all permissible order of operations for these assignations.

The problem of designation of a cyclic schedule for flexible job shop problem with a minimum cycle time belongs to a class of sequencing problems. Sequencing problems embrace scheduling problems in which the solution can be uniquely represented in the form of the order of operations on the machines as for the given order the value of the objective function is defined unambiguously. There is a wide range of methods for constructing algorithms for sequencing problems in which the most effective use of problem properties are being solved in order to increase the efficiency. However, the most time consuming part of these algorithms is determination of the objective function value or its estimation.

3 Determination of the Cycle Time for the Sequence π

In the section we propose an original method of determining the cycle time for a given order π. Considerations begin by analyzing the performance of left shifted schedule for execution of operations in cyclic systems. At this stage, it is required to fulfil the constraints (1–4), whereas the constraints (5) do not have to be met. The earliest completion moments of operations on the machines can be calculated on the following recursive formula:

$$C^x_{j_l(s),k_l(s)} = \max\{C^x_{j_l(s-1),k_l(s-1)}, C^x_{j_l(s),k_l(s)-1}\} + p_{j_l(s),k_l(s)}, \tag{7}$$

where $C^x_{j_l(s),0} = 0$, $C^x_{j_l(0),k_l(0)} = 0$ for $x = 0$ and $C^x_{j_l(0),k_l(0)} = C^{x-1}_{j_l(n_l),k_l(n_l)}$ for $x = 1, 2, \ldots$.

It can be easily seen that the execution of the calculations in accordance with the order Q for subsequent cycles (see Section 2) enables determination of the completion times for the operation in a sequential way because at the time of designation of the value of expression(7) the completion times of machine and technological predecessor are known, i.e. have been designated earlier.

For the operation (i, k) performed in the production cycle x there is a sequence $u^x_{i,k} = (u^0_1, ..., u^{x_s}_s, ..., u^{x_{n_u}}_{n_u})$, $u^{x_s}_s = (i_s, k_s)$, $u^{x_{n_u}}_{n_u} = (i, k)$ defined, such that $S^{x_s}_{i_s,k_s} = C^{x_s-1}_{i_{s-1},k_{s-1}}$. Obviously, the predecessor operation (i_s, k_s) in a sequence $u^x_{i,k}$ is its machine or technological predecessor. Operation $u^0_1 = (i_1, k_1)$ performed in 0 cycle will be called a source of schedule for the operation (i, k) executed in a cycle x. By $L(u^x_{i,k}) = \sum_{s=1}^{n_u} p_{i_s k_s}$ let us designate the sum of operations' execution times belonging to a sequence $u^x_{i,k}$. It is easy to observe that $C^x_{i,k} = S^0_{i_1,k_1} + L(u^x_{i,k})$.

Let us consider the sequence $u^x_{i,k}$, $J_i \in J$, $k = 1, ..., n_k$, $x = 1, ...$ with the source $u^0_{i,k}$. We have $C^x_{i,k} = S^0_{i,k} + L(u^x_{i,k}) = C^0_{i,k} - p_{i,k} + L(u^x_{i,k})$, thus, the cycle time $\tau(\pi)$ must meet the following condition:

$$\tau(\pi) \geq (L(u^x_{i,k}) - p_{ik})/x \; for \; J_i \in J, \; k = 1, ..., n_k, \; x = 1, \ldots . \tag{8}$$

Property 1 *For a given order of operations execution in a cyclic flexible job shop system* $\pi = (\pi_1, ..., \pi_m)$*, where* $\pi_l = ((j_l(1), k_l(1)), ..., (j_l(n_l), k_l(n_l)))$*,* $l \in M$*, the cycle time is:*

$$\tau(\pi) = \max\{(L(u^x_{j_l(1),k_l(1)}) - p_{j_l(1),k_l(1)})/x | l \in M,\ x = 1, ..., m-1\}, \qquad (9)$$

where $L(u^x_{j_l(1),k_l(1)})$ is a sum of times assigned to elements of a sequence $u^x_{j_l(1),k_l(1)}$ with source $u^0_{j_l(1),k_l(1)}$. Property 1 is given without a proof.

Arbitrarily selected sequence $u^{x*}_{j_{l*}(1),k_{l*}(1)}$ such that $\tau(\pi) = (L(u^{x^*}_{j_{l^*}(1),k_{l^*}(1)}) - p_{j_{l^*}(1),k_{l^*}(1)})/x^*$, $l^* \in M$, $x^* \in \{1, ..., m-1\}$ will be called *a critical sequence.*

Algorithm 1 describes, in a precise manner, the proposed method for the determination of cycle time $\tau(\pi)$ for a given order π. In the commentary the discussion of the algorithm will be limited only to steps 2 and 3.1, since the rest of the steps are obvious. In Step 2 in positions 0 in the permutation π_l there is fictional operation $-l$ inserted. Let us observe the fact that during performing the computations for the machine $l \in M$, the completion time for operation execution is initiated by a large natural number B. Since the operation $-l$ is the machine predecessor of operation $(j_l(1), k_l(1))$ therefore this initiation makes $(j_l(1), k_l(1))$ the source of schedule.

Algorithm 1. Sequential Computing $\tau(\pi)$

1. Determine sequence $Q(\pi)$
2. Set $\pi_l(0) = -l$
3. For $l = 1, \ldots, m$ do
3.1 Set $C^0_{j_l(-s),k_l(-s)} = B$ for $s = l$ and $C^0_{j_l(-s),k_l(-s)} = 0$ for $s \neq l,\ s = 1, \ldots, m$
3.2 For $x = 0, \ldots, m-1$ do
3.2.1 For $s = 1, \ldots, o$ do
3.2.1.1 Compute $C^x_{j_s,k_s}$ $((i_s, k_s) = q_s)$ from (7).
4. Determine $\tau(\pi)$ from (9).

Property 2 *The cycle time* $\tau(\pi)$ *can be determined in time* $O(om^2)$.

Proof. Step 3.2.1.1 requires $O(1)$ computation time. This step is executed $(m \cdot m \cdot o)$ times (loops 3.2, 3.2.1, 3.2.1.1). Step 3.1 requires $O(m)$ time and is executed m times. Step 4. requires $O(m^2)$ time.

By $C^{(l)x}_{i,k}$ let us designate the completion time of execution of operation (i, k) in x-th production cycle for the source schedule $(j_l(1), k_l(1))$. The sequence $\boldsymbol{C}^x_{i,k} = (C^{(1)x}_{i,k}, C^{(2)x}_{i,k}, ..., C^{(m)x}_{i,k})$ creates vector of m-elements. Algorithm 2 describes the vector processing version of the proposed method for determining the cycle time.

Algorithm 2. Vector Computing of $\tau(\pi)$

1. Determine sequence $Q(\pi)$
2. For $l = 1, ..., m$ set $\pi_l(0) = 0$, set in parallel $\boldsymbol{C}^0_{0,0} = \mathbf{0}$, set $\boldsymbol{C}^{(l)0}_{0,0} = B$.
3. For $x = 0, \ldots, m-1$ do
3.1 For $s = 1, \ldots, o$ do
3.1.1 Compute in parallel $\boldsymbol{C}^x_{j_s,k_s}$ $((i_s, k_s) = q_s)$ from (7).
4. Determine $\tau(\pi)$ from (9).

Property 3 *The cycle time $\tau(\pi)$ can be designated in time $O(om)$ on the vector processor consisting of m computing cores.*

Proof. Step 3.2.1 requires $O(1)$ computing time on vector processor. The step is performed $(m \cdot o)$ times (loops 3. and 3.1). Other steps require much less time.

4 Simulated Annealing Algorithm

One of the most effective and, at the same time, easiest to implement methods of construction of local search algorithms is Simulated Annealing (SA, see Pempera et al. [15]). In each iteration of the algorithm, on the basis of the base solution π there is a new solution π' generated. If $T(\pi') \leq T(\pi)$, then this solution is accepted unconditionally, otherwise with probability $p = \exp(-\Delta/t)$, where $\Delta = T(\pi') - T(\pi)$, whereas t is the temperature in a given iteration of the algorithm. The temperature decreases in each iteration of the algorithm according to the approved cooling scheme. The algorithm terminates computations after a predetermined number of iterations.

In the proposed algorithm, the new solution is generated by shifting a single operation. It is implemented in the three following steps:

1. randomly select operation (i, k) from the critical path,
2. randomly select machine l from the set M_{ik},
3. designate feasible positions in which operation (i, k) on machine l can be inserted and insert randomly selected.

The proposed method of generating new solutions is limited to generating feasible solutions potentially better than the current one, i.e. is based on the following property:

Property 4 *Let π' be the order of operations on the machines resulting from the order π such that $T(\pi') < T(\pi)$, then at least one operation from the critical path is performed on a different machine or in a different position.*

Property 4 is a simplification of a known, for a wide class of scheduling problems, block theory [7], [8].

At the same time the above operation is not time-consuming since the most time consuming is Step 3 performed in time $O(o)$ (see Property 5).

Property 5 *Let* (i,k), $J_i \in J$, $k = 1, ..., n_i$ *be any operation and* l, $l \in M_{ik}$ *any machine on which this operation can be performed. The range of feasible positions on machine* l *in which operation* (i,k) *can be inserted, can be designated in time* $O(o)$.

5 Results of Computational Studies

Simulated annealing algorithm described in Section 5 was implemented in 4 versions: (SA) – Single-walk Simulated Annealing,(MSA) – Multiple-walk Simulated Annealing,(PSA) – Parallel single-walk Simulated Annealing algorithms with parallel computing of the cycle time, and (MPSA) – Multiple-walk Parallel Simulated Annealing algorithms with parallel computing of the cycle time. The algorithms have been implemented in the Visual Studio 2010 environment in C++ language. The tests were conducted on an Intel I7-core 2.4GHz 4-core (Intel Hyper Threading 8-cores) computer, 4GB of RAM, managed by 32-bit Windows 7 operating system. Experimental studies were conducted on the set consisting of 21 instances proposed by Barnes and Chambers [1]. The set consists of instances containing from 10 to 15 tasks and from 11 to 18 machines with varying degrees of flexibility. In a single path of an algorithm, the simulated annealing process was carried out *rep* times. The first computations process began with the solutions generated by a construction algorithm, the remaining began with the last solutions generated by the previous process. The simulated annealing was performed with the following parameters: the initial temperature of 1000, the rate of cooling scheme $\lambda = 0.995$, the number of iterations 10000. In order to generate the initial solution there was construction algorithm used, with the priority rule: earliest completion time.

Computational study of the proposed algorithms were divided into two stages. The aim of the first phase was to examine the speedup of algorithms obtained by the use of vector processing in a real computer system, while the second assessment concerned the quality of the generated solutions, in particular the quality of the solutions generated by multipath parallel algorithms. During the computational study there were: $T(A)$ -time cycle for the best solutions found by A algorithms and $CPU(A)$ - time of computations of algorithm A, $A \in \{SA, PSA, MSA, MPSA\}$ remembered.

5.1 Assessment of Vector Processing Speedup

Today's processors produced by leading manufacturers, used in stationary and mobile computers, support parallel processing on two levels: processor instructions and multi-core processing. In case of instructions level (SSE), in one cycle of calculation one identical computational activity is performed on the number of data, remembered in computer registers as a vector consisting of a certain number of elements s. In other words, the calculations are performed by the vector processor consisting of s cores. SSE registers size is 128- bit, thus for the

Table 1. Time of running and sppedup of SA algorithms

Name	$n \times m$ (o)	one path			4 paths			8 paths		
		SA	PSA	SU	MSA	MPSA	SU	MSA	MPSA	SU
mt10c1	10×11 (100)	30.6	6.1	5.0	38.4	8.0	4.8	50.2	10.1	5.0
mt10cc	10×12 (100)	36.7	6.7	5.5	43.6	8.8	5.0	60.4	11.2	5.4
mt10x	10×11 (100)	31.6	5.9	5.3	37.2	8.2	4.6	49.8	9.9	5.0
mt10xx	10×12 (100)	36.0	6.5	5.5	41.9	8.8	4.8	59.9	11.0	5.5
mt10xxx	10×13 (100)	42.7	7.1	6.0	47.3	9.3	5.1	65.7	11.9	5.5
mt10xy	10×12 (100)	35.9	6.6	5.4	41.7	8.8	4.8	59.6	10.8	5.5
mt10xyz	10×13 (100)	42.5	7.1	6.0	49.1	8.9	5.5	70.1	12.1	5.8
setb4c9	15×11 (150)	46.6	8.6	5.4	52.8	10.7	4.9	74.0	13.4	5.5
setb4cc	15×12 (150)	55.6	9.4	5.9	63.3	11.3	5.6	86.4	14.4	6.0
setb4x	15×11 (150)	46.6	8.5	5.5	52.9	10.2	5.2	73.0	14.1	5.2
setb4xx	15×12 (150)	54.8	9.0	6.1	62.1	10.8	5.8	85.6	14.5	5.9
setb4xxx	15×13 (150)	64.4	10.0	6.4	73.2	11.8	6.2	100.8	16.0	6.3
setb4xy	15×12 (150)	54.6	9.4	5.8	61.1	11.4	5.4	86.0	14.6	5.9
setb4xyz	15×13 (150)	65.4	9.6	6.8	71.8	12.0	6.0	99.8	15.7	6.4
seti5c12	15×16 (225)	156.9	17.2	9.1	172.2	22.1	7.8	233.5	27.5	8.5
seti5cc	15×17 (225)	177.1	19.3	9.2	195.8	24.2	8.1	264.4	31.1	8.5
seti5x	15×16 (225)	157.3	17.0	9.3	172.2	21.0	8.2	234.2	27.8	8.4
seti5xx	15×17 (225)	177.4	19.8	8.9	197.6	23.7	8.3	265.9	30.7	8.7
seti5xxx	15×18 (225)	202.4	20.2	10.0	221.1	25.6	8.6	293.4	32.7	9.0
seti5xy	15×17 (225)	175.8	19.6	9.0	193.3	24.1	8.0	261.0	30.6	8.5
seti5xyz	15×18 (225)	196.5	20.7	9.5	217.1	25.7	8.4	294.3	32.4	9.1

data of Int16 type, used in the calculations, the size of the vector is $s = 8$. Undoubtedly, in case of vectors with sizes larger than s, the vector is divided into fragments of s-elements and then they are processed sequentially.

All algorithms were run with parameter $rep = 20$ while the multipath algorithms were started simultaneously at 4 and 8 cores (each realized a different path). Table 1 presents algorithms' execution times for all instances of the test. In addition, on the basis of the time of the algorithm running in the basic version and using the processing vector, there was designated the speedup of calculations $SU = CPU(SA)/CPU(PSA)$ ($SU = CPU(MSA)/CPU(MPSA)$).

The analysis of single-path algorithms shows that for certain instances the speedup is greater than the number of cores of vector processor $s = 8$. This stays in contrast to Ahmdal's Law. In fact, in a sequential and parallel processing participate other CPU instructions of varying execution time. What is more,the most frequently used function max for SSE instruction is executed in one processor cycle, while in case of sequential x86 instruction it consists of comparison and jump instructions. The use of vector processing helps to accelerate the SA algorithm running from 5 to 10 times in single-path version. In case of multipath versions the speedup is slightly smaller. The size of the speedup depends on the number of machines. The smallest speedup is observed, e.g. for a small number of machines and distant from the multiple of $s = 8$ (mt10c).

Comparing the running time of single-path PSA algorithm and 4-path MPSA algorithm it can be seen that the MPSA running time is on average by 1.3 (minimum 1.2) times longer than the SA time. I7 processor consists of 4 identical cores, the MPSA algorithm performs exactly 4 times more computations than the SA. In case of 8-track MPSA algorithm the running time is on average 1.7 times longer than the SA. In this case, we can see the beneficial effects of Hyper Threading technology.

5.2 Assessment of Algorithms Efficiency

The aim of the second phase of research was to assess the quality of solutions generated by PSA (single-path), PSA4 (4-paths) and PSA8 (8-paths) algorithms. In assessing the quality the relative deviation was used for the cycle time of solution π^A generated by the A algorithm compared to the cycle time of the best know solution π^*: $Dev(A) = (\tau(\pi^A) - \tau(\pi^*))/\tau(\pi^*)$. The algorithm was executed with the parameter $rep = 20$. Reference solutions π^* were generated by the PSA8 algorithm with $rep = 50$.

As a result of detailed analysis of the test results it was noted that for the PSA single-walk algorithm deviation was 2.1% to 8.1%, 4.7% in average, PSA4 at the same time generates solutions with the value of Dev from 0% to 4.6%, in average 2.1%. Solutions of PSA8 have an average coefficient of $Dev = 1.3\%$.

In summary, the results show that the use of vector processing significantly accelerates SA algorithm. In addition, the use of multiple-walk search yields a significantly better solutions in the same time of calculations. The average value of Dev for 4 and 8-path algorithms is more than 2 and almost 4 times smaller than the Dev for single-walk algorithm, respectively.

6 Conclusions

The work is devoted to the scheduling of tasks in a cyclic flexible production system. The paper presents new properties of the problem and the properties characteristic of the cyclic manufacturing. Based on the theoretical properties, a genuine method of the cycle time determination was proposed. Sequential and parallel (based on vector processing) implementations were presented as well as the analysis of the computational complexity of the proposed methodology.

As further research, parallel processing techniques are planned to be designed for efficient calculations on modern computational units (GPU, HPC), equipped with large number of cores.

References

1. Barnes, J.W., Chambers, J.B.: Flexible Job Shop Scheduling by tabu search, Graduate program in operations research and industrial engineering, Technical Report ORP 9609, University of Texas, Austin (1996)

2. Bożejko, W., Uchroński, M., Wodecki, M.: Parallel hybrid metaheuristics for the flexible job shop problem. Computers & Industrial Engineering 59, 323–333 (2010)
3. Bożejko, W., Uchroński, M., Wodecki, M.: The new golf neighborhood for the flexible job shop problem. In: Proceedings of the ICCS 2010. Procedia Computer Science, vol. 1, pp. 289–296. Elsevier (2010)
4. Bożejko, W.: On single-walk parallelization of the job shop problem solving algorithms. Computers & Operations Research 39, 2258–2264 (2012)
5. Bożejko, W., Pempera, J., Smutnicki, C.: Parallel Tabu Search Algorithm for the Hybrid Flow Shop Problem. Computers and Industrial Engineering 65, 466–474 (2013)
6. Brucker, P., Kampmeyer, T.: Cyclic job shop scheduling problems with blocking. Annals of Operations Research 159, 161–181 (2008)
7. Grabowski, J., Skubalska, E., Smutnicki, C.: On Flow Shop Scheduling with Release and Due Dates to Minimize Maximum Lateness. Journal of the Operational Research Society 34(7), 615–620 (1983)
8. Grabowski, J., Pempera, J.: New block properties for the permutation flow shop problem with application in tabu search. Journal of Operational Research Society 52, 210–220 (2001)
9. Hurink, E., Jurisch, B., Thole, M.: Tabu search for the job shop scheduling problem with multi-purpose machine. Operations Research Spektrum 15, 205–215 (1994)
10. Jia, H.Z., Nee, A.Y.C., Fuh, J.Y.H., Zhang, Y.F.: A modified genetic algorithm for distributed scheduling problems. International Journal of Intelligent Manufacturing 14, 351–362 (2003)
11. Kacem, I., Hammadi, S., Borne, P.: Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Transactions on Systems, Man, and Cybernetics Part C 32(1), 1–13 (2002)
12. Kampmeyer, T.: Cyclic Scheduling Problems, Ph. D. Thesis, University Osnabrück (2006)
13. Kechadi, M., Low, K.S., Goncalves, G.: Recurrent neural network approach for cyclic job shop scheduling problem. Journal of Manufacturing Systems 32, 689–699 (2013)
14. Mastrolilli, M., Gambardella, L.M.: Effective neighborhood functions for the flexible job shop problem. Journal of Scheduling 3(1), 3–20 (2000)
15. Pempera, J., Smutnicki, C., Żelazny, D.: Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm. Procedia Computer Science 18, 936–945 (2013)
16. Xia, W., Wu, Z.: An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problem. Computers and Industrial Engineering 48, 409–425 (2005)