Computers & Industrial Engineering 95 (2016) 156–163

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Parallel metaheuristics for the cyclic flow shop scheduling problem

Wojciech Bożejko[a,*], Mariusz Uchroński[b], Mieczysław Wodecki[c]

[a] *Department of Control Systems and Mechatronics, Faculty of Electronics, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland*
[b] *Wrocław Centre of Networking and Supercomputing, Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland*
[c] *Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland*

ARTICLE INFO

Article history:
Received 11 November 2015
Received in revised form 5 March 2016
Accepted 7 March 2016
Available online 16 March 2016

Keywords:
Scheduling
Parallel algorithm
Cyclic flow shop problem
Block properties

ABSTRACT

In the paper there was proposed a new method of detection of block properties for cyclic flow shop problem with machine setups that uses patterns designated for each machine by solving the adequate traveling salesman problem. The proposed method is intended to be run in the environment of shared memory in concurrent computations, such as coprocessors, GPU, or machines with multi-core CPUs. The proposed method of accelerating the review of the neighborhood through the use of the blocks was tested on two parallel metaheuristics: tabu search and simulated annealing.

© 2016 Elsevier Ltd. All rights reserved.

1. Introduction

Recently, there has been observed a growing interest in cyclic problems of tasks scheduling in both the environment of theorists dealing with discrete optimization problems and in the environment of practitioners in the industry. Cyclic production is, in fact, a very effective method in modern flexible manufacturing system as it significantly simplifies control, i.e. a fixed schedule is repeated in many periods. The most important benefit of the method is its ability to produce, in predetermined intervals, multi-assortment product mix resulting from customer demand. This process provides not only systematic replenishment of usually relatively small inventory of customers but also generates a systematic demand for both semi-finished or raw materials and materials obtained from suppliers. This method simplifies the management of supply chain. Another very important advantage of cyclic production is relatively easy detection of anomalies during manufacturing process which may indicate a deterioration of either the quality parameters of the production system or manufactured products themselves.

In the world literature there are many studies concerning various aspects of cyclic control in enterprises which manufacture products on a mass scale. There are examples of application of cyclic scheduling in various spheres of industry, transport and logistics (e.g. Gertsbakh & Serafini, 1991; Kats & Levner, 2010; Mendez, Cerda, Grossmann, Harjunkoski, & Fahl, 2006; Pinedo, 2005, 2008; Pinto, Barbosa-Póvoa, & Novais, 2005). Unfortunately, the existing models and calculation tools enable determination of the optimal (minimizing cycle time) control for production systems executing only a small number of tasks. In the work, there is considered a cyclic flow shop problem with setup times. Strong NP-hardness of many simple versions of cyclic scheduling problems, in particular, of the considered problem, limits the scope of applications of exact algorithms to instances with a small number of tasks, nevertheless, in this context of minimizing the cycle time, both the design and the use of exact algorithms seems to be fully justified (Brucker, Burke, & Groenemeyer, 2012). However, due to the NP-hardness, in determination of satisfactory solutions there are commonly fast approximate algorithms used based on local search techniques, for example: simulated annealing (in parallel version – Bożejko, Pempera, & Wodecki, 2015) or tabu search (Bożejko, Uchroński, & Wodecki, 2015, 2014). Methods of this type are usually based on a two-level decomposition of the problem: the first – determining the optimal sequence of tasks (upper level) and the second – multiple determining the minimum value of criteria for a given sequence of tasks (bottom level). In case of the conventional, non-cyclic scheduling problem, solution to the lower level can be obtained in a time-efficient manner by analyzing a specific graph. However, in case of the defined, in this paper, problem obtaining the solution to the lower level is a relatively time-consuming process since, in general, it requires the solution of a certain linear programming problem. Therefore, any special properties, including those that enable obtaining more efficient calculation of the cycle time, the search schedule and reduction of the

* Corresponding author.
E-mail addresses: wojciech.bozejko@pwr.edu.pl (W. Bożejko), mariusz.uchronski@pwr.edu.pl (M. Uchroński), mwd@ii.uni.wroc.pl (M. Wodecki).

http://dx.doi.org/10.1016/j.cie.2016.03.008
0360-8352/© 2016 Elsevier Ltd. All rights reserved.

multiplicity of the locally browsed neighborhood or acceleration of its browse, are very desirable.

In this paper, we propose the use of new properties, the so-called blocks which reduce the number of solutions viewed while generating the neighborhood executed by local search algorithms, such as tabu search or simulated annealing. Determination of the blocks can be performed both sequentially and simultaneously, using multiprocessor calculations environment. Appropriate methods of construction are shown in this work with the use of model PRAM machine equipment, which for many years, has not only been the standard for the theoretical verification of the computational complexity of the parallel algorithms but also close to practice approximation of contemporary parallel architectures.

2. Problem description

The cyclic manufacturing process, considered in the work, can be formulated as follows: there is a set of n tasks given $\mathcal{J} = \{1, 2, \ldots, n\}$, which are to be performed in cycles (repeatedly) on machines from the set $\mathcal{M} = \{1, 2, \ldots, m\}$. A given task should be executed in a sequence, on each of the m machines $1, 2, \ldots, m$, in a technological order. The task $j \in \mathcal{J}$ is a sequence of m operations $O_{1,j}, O_{2,j}, \ldots, O_{m,j}$. The operation $O_{k,j}$ corresponds to execution of task j on machine k, in time $p_{k,j}$ $(k = 1, 2, \ldots, m, j = 1, 2, \ldots, n)$. After the completion of a certain operation and before the start of the next operation there should be setups of a machine performed. Let $s^k_{i,j}$ $(k \in \mathcal{M},\ i \neq j\ i,j \in \mathcal{J})$ be the setup time between the operation $O_{k,i}$ and $O_{k,j}$.

A set of tasks executed in a single cycle is called PMS (*minimal part set*). MPSs are processed cyclically one after another. The order of tasks is to be determined (the same on each machine) in such a way which minimizes the cycle time, i.e. the time of commencement of the tasks from the set $\mathcal{J}$ in the next cycle. The following restrictions must be fulfilled:

(a) each operation can be executed only by one machine,
(b) none of the machines can execute more than one operation at the same time,
(c) the technological order of operations must be preserved,
(d) the execution of any operation cannot be interrupted before its completion,
(e) each machine, between successively performed operations, requires setup,
(f) each operation is executed sequentially (in successive MPSs) after the completion of cycle time.

The considered problem relies in determination of the moments of starting of the tasks' execution on machines that meet the limitations (a)–(f), so that the cycle time (the time at which the task is executed in the next MPS) was minimal. Let us assume that in each of the MPSs, on each machine, the tasks are executed in the same order. Thus, in the cyclic schedule the order of tasks' execution on the machines can be represented by a permutation of the tasks in the first MPS. In fact, on its basis, we can determine the starting moments of tasks' execution on the machines in the first MPS. Increasing them by a multiplication of the cycle time, one gets the starting moments of tasks' execution in any of the MPS (the starting moment of execution of any operations in the next MPS should be increased by cycle time). Let Φ be the set of all permutations of the elements from the set of tasks $\mathcal{J}$. Therefore, the considered in the work problem boils down to determining of permutations of tasks (elements of the set Φ) which minimizes the length of the cycle time. In short, this problem will be denoted by **CFS**.

3. Mathematical model

Let $[S^k]_{m\times n}$ be the matrix of starting moments of tasks' execution of kth MPS (for the established order $\pi \in \Phi$), where $S^k_{i,j}$ denotes the starting moment of execution of task j on the machine i. We assume that tasks in the next MPS-s are carried out cyclically. This indicates that there is a constant $T(\pi)$ (period) such that

$$S^{k+1}_{i,\pi(j)} = S^k_{i,\pi(j)} + T(\pi), \quad i = 1, \ldots, m,\ j = 1, \ldots, n,\ k = 1, 2, \ldots \tag{1}$$

Period $T(\pi)$ undoubtedly depends on permutation π and is called *time period* of the system. The minimum value $T(\pi)$, for a fixed π, will be called *minimum cycle time* and will be denoted by $T^*(\pi)$. Since the order of tasks within the given MPS is the same, it is sufficient just to designate the order of tasks π for a single (*the first*) MPS and move it by the quantity $k \cdot T(\pi),\ k = 1, 2, \ldots$ on the time-line. For the established order of execution of tasks $\pi \in \Phi$, optimum value of cycle time $T^*(\pi)$ can be determined by solving the following optimization task:

$$T^*(\pi) = \min\{T : T \in \mathbb{R}\}, \tag{2}$$

$$S_{i,\pi(j)} + p_{i,\pi(j)} \leqslant S_{i+1,\pi(j)}, \quad i = 1, \ldots, m-1,\ j = 1, \ldots, n, \tag{3}$$

$$S_{i,\pi(j)} + p_{i,\pi(j)} + s_{\pi(j,\pi(j+1))} \leqslant S_{i,\pi(j+1)}, \quad i = 1, \ldots, m,\ j = 1, \ldots, n-1, \tag{4}$$

$$S_{i,\pi(n)} + p_{i,\pi(n)} + s_{\pi(n,\pi(1))} \leqslant S_{i,\pi(1)} + T, \quad i = 1, \ldots, m, \tag{5}$$

$$S_{i+1,\pi(n)} \leqslant S_{i,\pi(1)} + T, \quad i = 1, \ldots, m-1. \tag{6}$$

Without loss of generality, it is possible to assume that the starting moment of the first task's execution on the first machine is $S_{1,\pi(1)} = 0$. For any order of tasks in the first MPS and solving the above linear programming task, it is possible to determine the minimum cycle time in polynomial time. In case of an exact algorithm (complete overview) the solution to **CFS** problem should therefore be done for each of $n!$ permutation – element of a set Φ. The next chapter includes a presentation of an approximate solutions method to the considered in this work problem.

4. Solution method

In many heuristic algorithms solutions to NP-hard problems are constituted by reviewed neighborhoods, i.e. subsets of solution space. In case where solutions to problems are permutations neighborhoods are usually generated by insert or swap type moves and their combinations (Bożejko & Wodecki, 2007). They consist in changing positions of elements in the permutation. The number of elements of such a neighborhood is at least $n(n-1)/2$, where n is the size of the data. In practical applications (with large n), viewing the neighborhood is the most time consuming element of the algorithm. The description of computational experiments presented in the literature shows that the number of iterations of the algorithm has a direct impact on the quality of designated solutions. Hence, there is observed the search for methods accelerating the work of a single iteration of the algorithm. One of them is the reduction of the number of the neighborhood elements, their parallel generation and viewing. In case of tasks scheduling problems on multiple machines with the minimization of time of tasks' execution (C_{max}) there are 'blocks eliminating properties' (Grabowski & Wodecki, 2004) successfully used. Similar properties are implemented in the algorithm solving the problem of determining minimum cycle time, more specifically – a minimum time of a single machine run. The properties enable elimination of elements from the neighborhood that do not directly provide the improvement of the best solution found so far. In the work (Bożejko et al., 2015) there was a method to solve the **CFS** problem and tabu search sequential algorithm presented. The further part of the chapter outlines the main elements of the above mentioned method. For a fixed permutation $\pi \in \Phi$ and machine $k \in \Phi$

$$T_k(\pi) = \sum_{i=1}^{n-1} (p_{k,\pi(i)} + s^k_{\pi(i),\pi(i+1)}) + p_{k,\pi(n)} + s^k_{\pi(n),\pi(1)} \tag{7}$$

is the time of tasks' execution in the order π, with a setup time between the task $\pi(n)$ and $\pi(1)$ (i.e. The last task in a given MPS and the first in the next one). Thus, it can be easily proven that the minimum cycle time is

$$T^*(\pi) = \min\{T_i(\pi) : i = 1, 2, \ldots, m\}. \tag{8}$$

Ideas of the algorithm solving the problem of determining the optimal cycle time (solution to **CFS** problem) can be summarized as follows:

```
MinCyc algorithm
  $\pi$ – start permutation;
  $T^\circ$ – the best solution found so far;
  $T^\circ \leftarrow T^*(\pi)$;
  repeat
    Step 1: Generate from $\pi$ a New permutation $\beta$;
    Step 2: Compute minimum cycle time $T^*(\beta)$;
      if $T^*(\beta) < T^\circ$ then
        $T^\circ \leftarrow T^*(\beta)$;
  until (end condition).
```

In Step 1 permutation β will be generated from the neighborhood π. In Step 2, using (8) it is possible to compute minimum cycle time $T^*(\beta)$. Let us assume that the maximum in (8) was achieved for kth machine, i.e. $T^*(\beta) = T_k(\beta)$. Hence, the following remark.

Remark 1 (Bożejko et al., 2015). The necessary condition for reducing the minimum cycle time $T^*(\beta)$ is shortening the work time of kth machine, i.e. reducing of $T_k(\beta)$.

Designating of minimum work time of kth machine, i.e. the value $\min\{T_k(\delta) : \delta \in \Phi\}$ can be reduced to the following traveling salesman problem. Let $H_k = (\mathcal{V}, \mathcal{E}; p, s)$ be the full graph, where

- set of vertices: $\mathcal{V} = \mathcal{J}$,
- set of edges: $\mathcal{E} = \{(v, u) : v \neq u,\ v, u \in \mathcal{V}\}$,
- weights of vertices: $p(v) = p_{k,v},\ v \in \mathcal{V}$,
- weights of edges: $s(e) = s^k_e,\ e \in \mathcal{E}$.

Remark 2 (Bożejko et al., 2015). The work time of kth machine $T_k(\pi)$ equals to the length (i.e. the sum of weights of vertices and edges) of Hamiltonian cycle $(\pi(1), \pi(2), \ldots, \pi(n), \pi(1))$ in graph H_k.

Remark 3 (Bożejko et al., 2015). Minimum work time of kth machine equals to the length of the traveling salesman path in graph H_k, i.e. minimum (due to the length) of Hamiltonian cycle.

Let π^*_k be optimum Hamiltonian cycle (traveling salesman path) in graph $H_k(\pi)$ $(k = 1, 2, \ldots, m)$. This is the optimal (i.e. minimal due to execution time) order of the tasks from the set $\mathcal{J}$ on kth machine. Such permutation will be called *pattern* for kth machine.

In order to reduce the work time of kth machine $T_k(\pi)$ one must generate permutations from π taking into account the elements of the pattern. The patterns also enable eliminating of the elements from the neighborhood which do not improve the current value of the cycle time T° (Step 2 of **MinCyc** algorithm).

4.1. Blocks of tasks

Let

$$B = (\pi(a), \pi(a+1), \ldots, \pi(b)), \tag{9}$$

be a sequence of occurring immediately one after another tasks of the permutation $\pi \in \Phi$, π^*_k *pattern* for the kth machine and u, v $(u \neq v,\ 1 \leqslant u, v \leqslant n)$ pair of numbers such that:

W1: $\pi(a) = \pi^*(u), \pi(a+1) = \pi^*(u+1), \ldots, \pi(b-1) = \pi^*(v-1)$, $\pi(b) = \pi^*(v)$, or
W2: $\pi(b) = \pi^*(u),\ \pi(b-1) = \pi^*(u+1), \ldots, \pi(a+1) = \pi^*(v-1)$, $\pi(a) = \pi^*(v)$
W3: B is the maximum subsequence due to the inclusion, i.e. it is not possible to enlarge it either by an element $\pi(a-1)$, or by $\pi(b+1)$, fulfilling the constraints **W1** or **W2**),

If the sequence of tasks (9) meets the conditions **W1** and **W3** or **W2** and **W3**, then it is called a *block* on kth machine $(k \in \mathcal{M})$.

A sequential algorithm determining all of the blocks in the permutation is presented below.

```
SeqBlock algorithm
  $\pi = (\pi(1), \pi(1), \ldots, \pi(n))$ – permutation;
  $\pi^* = (\pi^*(1), \pi^*(1), \ldots, \pi^*(n))$ – pattern of permutation $\pi$;
  $t$ – number of blocks;
  $(b_1, b_2, \ldots, b_t)$ – vector of positions of initial blocks in $\pi$;
  $t \leftarrow 1; i \leftarrow 1$;
  while $(i \leqslant n)$ do
  begin
    $b_t \leftarrow i; q \leftarrow (\pi^*)^{-1}(\pi(i))$;
    while $(\pi(i) = \pi^*(i))$ do
    begin
      $q \leftarrow q + 1$;
      $i \leftarrow i + 1$;
    end;
      $i \leftarrow i + 1$;
  end.
```

The computational complexity of the algorithm is $O(n)$.

Determination of the pattern (optimal traveling salesman path in the graph H_k) is a NP-hard problem. Therefore, the use of approximation algorithms, e.g. 2-opt is justified. For each machine model the pattern should be determined once before running of **MinCyc** algorithm.

In the further part of this chapter we will describe permutations generated from π (Step 1 of **MinCyc** algorithm), where the cycle time is not less than $T^*(\pi)$, therefore do not give the improvement of the best current cycle time T°.

Theorem 1 (Bożejko et al., 2015). *If permutation β was generated from permutation π by swapping the order elements in certain internal block on machine $k \in \mathcal{M}$, then $T_k(\beta) \geqslant T_k(\pi)$.*

Therefore, generation of the neighborhood of permutation π (from which the best element is chosen – Step 1 of **MinCyc** algorithm) will take place in two phases:

1. Designation of blocks in π.

2. Swap before the first or the last element of the block a task that exists on this position in a pattern. This will increase one of the blocks.

5. Parallel determination of the blocks

In order to accelerate the algorithm for determining the minimum cycle time we present a method of parallelization of the most time-consuming, performed in each iteration, procedure for determining blocks.

Theorem 2. *The designation of blocks for the cyclic flow shop problem with setups can be done in* $O(\log n)$ *time on mn – processor of CREW PRAM machine.*

Proof. The design of the algorithm for block designation on n-processor CREW PRAM machine will be presented, based on methodology appearing in the monograph (Bożejko, 2010). Let the number of processors be $p = n$.

```
ParBlock algorithm
  $\pi = (\pi(1), \pi(1), \ldots, \pi(n))$ – permutation;
  $\pi^* = (\pi^*(1), \pi^*(1), \ldots, \pi^*(n))$ – pattern of permutation $\pi$;
  $t$ – number of blocks;
  $(b_1, b_2, \ldots, b_t)$ – vector of positions of initial blocks in $\pi$;
  $t \leftarrow 1; i \leftarrow 1$;
  Step 1: parfor $r \in \{1, 2, \ldots, p\}$ do
    $\pi(0) \leftarrow \pi(n+1) \leftarrow \pi^*(0) \leftarrow \pi^*(n+1) \leftarrow -1$;
    $(\pi^*)^{-1}(0) \leftarrow (\pi^*)^{-1}(n+1) \leftarrow -1$;
  if (previous position in $\pi$ is identical as in $\pi^*$, i.e.
    $\pi(r-1) = \pi^*((\pi^*)^{-1}(\pi(r)) - 1)$) then
      $B[r] \leftarrow 1$;
        else
      $B[r] \leftarrow 0$;
  Step 2: Determine prefix sum of $P$ elements from the table
   $B$, i.e.

$\forall_{r \in \{1,2,\ldots,p\}} P[r] = \sum_{i=1}^{r} B[i]$.

  Step 3: $P[n+1] \leftarrow -1$;
    parfor $r \in \{1, 2, \ldots, p\}$ do
      if $((P[r+1] = P[r])$ and $(P[r-1] < P[r]))$ then
          $B[r] \leftarrow 1$;
            else
          $B[r] \leftarrow 0$;
  Step 4: parfor $r \in \{1, 2, \ldots, p\}$ do
    if $(B[r] > 0)$ then
      $P'[r] \leftarrow 1$;
        else
      $P'[r] \leftarrow 0$;
  Step 5: Determine prefix sums of $P'$ elements from the table
   and place them
  again in table $P'$.
  Step 6: $b_0 := 0$;
    parfor $r \in \{1, 2, \ldots, p\}$ do
    if $(B[r] > 0)$ then
    begin
      $b_{P'[r]} := P[r]$;
      Synch_Barrier;
      $b_{P'[r]} \leftarrow b_{P'[r]} - b_{P'[r]-1}$;
      Synch_Barrier;
      $b_{P'[r]} \leftarrow b_{P'[r]} + 1$;
    end.
```

Table 1
The speedup values for different numbers of processors – Intel Xeon Phi 3120A, part I.

λ[a]	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$
1	0.003	0.001	0.001	0.001	0.001
2	0.008	0.003	0.003	0.003	0.003
5	0.018	0.008	0.008	0.007	0.007
10	0.037	0.016	0.017	0.017	0.015
20	0.078	0.033	0.036	0.043	0.033
50	0.308	0.154	0.165	0.143	0.147
100	0.632	0.441	0.458	0.443	0.400
200	0.961	0.915	0.986	1.047	0.980
500	1.229	1.578	1.889	2.061	2.171
1000	1.309	1.943	2.498	2.773	3.024
2000	1.302	2.255	2.994	3.490	3.810
5000	1.366	2.478	3.432	4.160	4.880
10,000	1.398	2.584	3.612	4.462	5.266

[a] The number of elements of permutation (tasks) is $\lambda = n \cdot 10^3$.

The above presented algorithm enables the determination of the blocks on exactly one machine. In order to designate blocks on each m machine, the number of $p = mn$ processors is required. It will not change the logarithmic time of calculations, because they are independent of each other. □

Prefix sums of the sequence of p-elements (Step 2 of **ParBlock** algorithm) can be determined on the CREW PRAM machine in time $O(\log p)$.

Proposition 1. *The above presented method of determining the blocks in time* $O(\log n)$ *can be implemented also on a smaller number of processors, i.e.* $p = O\left(\frac{mn}{\log n}\right)$.

The neighborhoods generated with the use of blocks determined parallelly were implemented in two canonical metaheuristic algorithms. In the first, Tabu Search (TS, Bożejko et al., 2015), the whole neighborhood is (deterministically) searched. In the second, Simulated Annealing (SA, Bożejko et al., 2015), an element of the neighborhood is determined randomly (in accordance with an established probability distribution).

6. Computational experiments

Computational experiments were performed with two main objectives:

1. Determination of speedup of parallel procedures of determining blocks.
2. Examining of the impact of parallel calculating of blocks on time and quality of solutions of the cyclic flow shop scheduling problem.

6.1. Acceleration of parallel procedures for blocks determining

A parallel process for the blocks designation has been implemented in C++ using OpenMP library. Data for computational experiments (permutations of tasks) were generated randomly. The number of permutations of elements changed in the range from 10^3 to 10^7.

Computational experiments were carried out in three computing environments with shared memory:

1. CPU – multicore processor Intel Core i7 X-980 (3.38 GHz) processor, enabling the use of 12 processors,
2. CPU – multicore Intel Xeon processor E5-2670 (2.30 GHz), enabling the use of 48 processors,
3. MIC – coprocessor Xeon Phi 3120 (6 GB, 1.1 GHz), enabling the use of 228 processors.

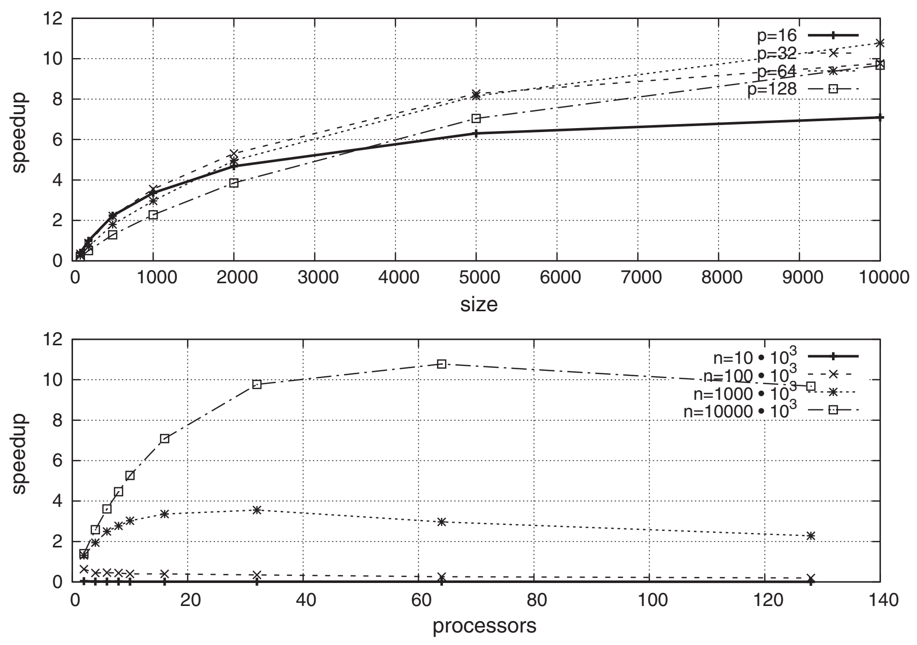

Fig. 1. The dependence of the speedup on the number processors for different number of tasks in the permutation and for different numbers of processors – Intel Xeon Phi 3120A.

The obtained results in the form of acceleration of parallel procedure for designating the blocks were presented in Tables 1–4. The first column indicates the number of elements of permutations (tasks), whereas the following columns show speedup values for different numbers of processors (threads) p. The symbol λ is a coefficient of expression denoting the number of elements in a permutation, which is $\lambda \cdot 10^3$. The dependence of speedup of the number of processors for different numbers of tasks in the permutation was presented in the form of graphs (Figs. 1 and 2). In Fig. 1 there was presented dependence of the speedup on the number of tasks in the permutations for a co-processor Intel Xeon Phi 3120A.

The results of computational experiments for co-processor Xeon Phi 3120A were presented in Tables 1 and 2 and in Fig. 1. The obtained results show that for a different number of tasks in permutation the speedup initially increases fairly rapidly, reaches a maximum and then decreases slowly. The number of processors, for which the maximum speedup is reached, depends on the size of the problem. For example, for $5000 \cdot 10^3$ tasks in permutation the maximum speedup is achieved for the number of $p = 32$ processors, whereas for $10,000 \cdot 10^3$ tasks it is $p = 64$ processors. Having in mind the notion of *scalability* of parallel algorithms it is possible to state that for $5000 \cdot 10^3$ tasks in permutation the parallel method for determining the blocks for $p = 1 \ldots 32$ is characterized with a *strong scalability* because with the growing number of processors the speedup increases. However, for $p = 64 \ldots 128$ the method is characterized with *weak scalability* since in order to obtain the increase of speed up, the size of the problem must be increased. On the basis of carried out computational experiments it is possible to state that the co-processor Intel Xeon Phi 3120A is scalable because increasing the size of the problem leads to greater speedup with a fixed number of processors (Fig. 1).

The results of computational experiments for Intel Core i7 X-980 processor are published in Table 4 and Fig. 2. Parallel method of blocks determination blocks in this case behaves similarly to the co-processor Intel Xeon Phi 3120A, however, the smaller the number of processors does not allow us to fully observe the scalability of hardware and the algorithm. It is worth mentioning that the values of speedup for any number of processors for Intel Core i7 X-980 is greater than 1 already for $10 \cdot 10^3$ tasks in permutations. In contrast, for the co-processor Xeon Phi 3120A exceeding the threshold of speedup $s = 1$ occurs for $500 \cdot 10^3$ task in permutations.

Table 2
The speedup values for different numbers of processors – Intel Xeon Phi 3120A, part II.

λ[a]	$p = 16$	$p = 32$	$p = 64$	$p = 128$
1	0.001	0.001	0.001	0.001
2	0.003	0.002	0.002	0.001
5	0.007	0.006	0.006	0.004
10	0.015	0.013	0.012	0.008
20	0.033	0.029	0.023	0.019
50	0.129	0.129	0.100	0.073
100	0.400	0.347	0.261	0.195
200	0.988	0.884	0.697	0.507
500	2.234	2.222	1.799	1.285
1000	3.360	3.552	2.967	2.278
2000	4.679	5.307	4.961	3.853
5000	6.306	8.270	8.154	7.043
10,000	7.091	9.766	10.775	9.679

[a] The number of elements of permutation (tasks) is $\lambda = n \cdot 10^3$.

Table 3
The speedup values for different numbers of processors – Intel Xeon E5-2670.

λ[a]	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$	$p = 16$	$p = 32$
1	0.247	0.232	0.211	0.192	0.171	0.128	0.032
2	0.447	0.453	0.339	0.353	0.292	0.231	0.081
5	0.578	0.815	0.677	0.614	0.593	0.453	0.167
10	0.786	0.839	0.941	0.816	0.835	0.631	0.219
20	0.885	1.057	1.351	1.242	1.357	1.164	0.400
50	1.005	1.365	1.726	1.464	1.914	1.994	0.886
100	1.101	1.581	1.897	2.122	2.326	2.474	1.358
200	1.103	1.602	2.012	2.371	2.491	2.900	1.757
500	1.125	1.583	1.931	2.255	2.345	2.606	2.331
1000	1.102	1.604	2.015	2.312	2.429	2.913	2.330
2000	1.119	1.761	2.094	2.572	2.821	3.287	3.245
5000	1.185	2.015	2.666	3.340	3.409	4.092	5.228
10,000	1.192	2.121	2.833	3.384	3.724	4.672	5.281

[a] The number of elements of permutation (tasks) is $\lambda = n \cdot 10^3$.

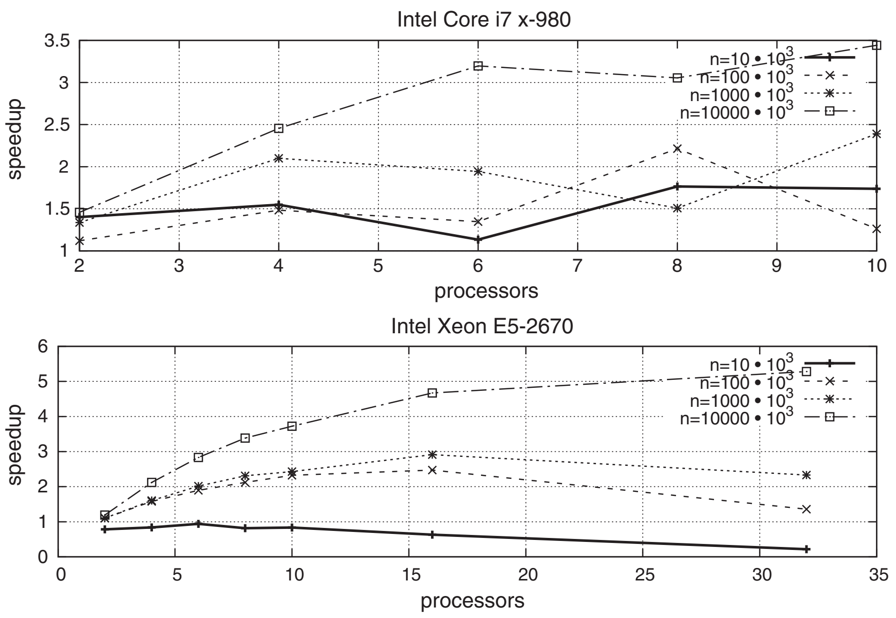

Fig. 2. The dependence of the speedup on the number processors for different number of tasks in the permutation – Intel Core i7 x-980 and Intel Xeon E5-2670.

6.2. Parallel metaheuristics

Parallel tabu search and simulated annealing algorithms for the cyclic job shop problem with setups of machines were implemented in C++ language using MPI library. In the implementation process there was MPSS (*Multiple Initial Point Single Strategy*) used according to Voß (1993) classification, in which processors start calculations with different initial solutions using the same search strategy. Computational experiments were performed on a server operating under control of a 64-bit operating system Linux Ubuntu 12.04 equipped with Intel Core i7 CPU X980. The algorithms were run for different numbers of threads $p = 1, 2, 4, 6, 8, 10$.

The calculations were performed on examples for the hybrid flow-shop problem included in the work (Ruiz & Stützle, 2008). Gathered data was divided into 25 groups. Each consists of

Table 4
The speedup values for different numbers of processors – Intel Core i7 x-980.

λ[a]	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$
1	0.495	0.214	0.214	0.199	0.217
2	0.551	0.667	0.520	0.453	0.587
5	0.981	1.188	0.615	1.253	0.890
10	1.401	1.549	1.133	1.765	1.737
20	1.243	1.709	1.952	2.144	2.399
50	1.142	2.045	2.407	1.854	2.575
100	1.119	1.483	1.347	2.214	1.262
200	1.349	2.090	2.614	2.118	2.486
500	1.382	2.126	1.907	2.181	2.510
1000	1.337	2.100	1.944	1.508	2.391
2000	1.267	2.207	2.746	2.544	3.058
5000	1.387	2.304	3.008	2.930	3.322
10,000	1.458	2.456	3.198	3.054	3.443

[a] The number of elements of permutation (tasks) is $\lambda = n \cdot 10^3$.

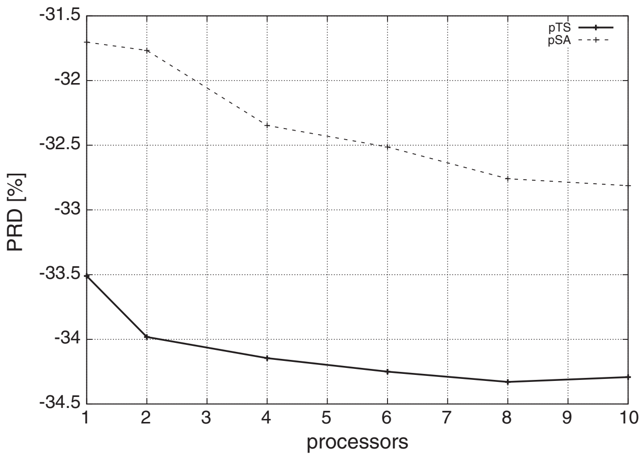

Fig. 3. Dependence of PRD from the number of processors.

Table 5
The mean relative improvement of PRD [%]. Tabu search algorithm.

$n \times m$	$p = 1$	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$	LB
20 × 5	−29.52	−33.24	−33.57	−33.74	−33.77	−33.74	−45.67
20 × 10	−31.61	−31.89	−32.24	−32.38	−32.52	−32.47	−43.77
20 × 20	−31.07	−31.01	−31.28	−31.38	−31.52	−31.44	−44.62
50 × 5	−38.03	−38.18	−38.07	−38.31	−38.44	−38.34	−48.92
50 × 10	−34.84	−34.82	−35.095	−35.06	−35.14	−35.09	−48.01
50 × 20	−31.93	−32.04	−32.13	−32.16	−32.28	−32.21	−46.32
100 × 5	−37.71	−37.86	−38.10	−38.16	−38.19	−38.21	−49.27
100 × 10	−34.30	−34.43	−34.49	−34.60	−34.74	−34.69	−48.01
Average	−33.51	−33.98	−34.14	−34.25	−34.33	−34.29	−46.82

Table 6
The mean relative improvement of PRD [%]. Simulated annealing algorithm.

$n \times m$	$p = 1$	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$	LB
20 × 5	−31.05	−30.92	−32.11	−32.16	−32.31	−32.82	−45.67
20 × 10	−30.05	−29.83	−30.44	−30.72	−31.36	−31.20	−43.77
20 × 20	−29.13	−29.50	−30.33	−30.36	−30.79	−30.60	−44.62
50 × 5	−35.38	−35.27	−36.08	−36.23	−36.39	−36.45	−48.92
50 × 10	−32.75	−33.00	−33.14	−33.50	−33.72	−33.78	−48.01
50 × 20	−30.34	−30.86	−31.00	−31.28	−31.52	−31.50	−46.32
100 × 5	−34.86	−34.93	−35.10	−35.30	−35.58	−35.63	−49.27
100 × 10	−31.94	−31.61	−32.43	−32.78	−32.65	−32.95	−48.01
100 × 20	−30.15	−30.31	−30.83	−30.86	−31.07	−31.17	−47.10
Average	−31.70	−31.77	−32.35	−32.51	−32.76	−32.81	−46.85

examples of the same number of tasks and machines. Individual groups differ from one another in the way of generation and/or the number of machines or tasks.

Percentage relative deviation of solutions (abbreviated to PRD) were determined as follows:

$$\mathrm{PRD} = \frac{F_{ref} - F_{alg}}{F_{ref}} \cdot 100\% \qquad (10)$$

where F_{ref} and F_{alg} are, respectively, the values of the objective function, the reference solution (determined by NEH construction algorithm Nawaz, Enscore, & Ham, 1983) and the solution determined by the tested algorithm. As the element of algorithms accuracy analysis, column LB shows the relative percentage deviation of the NEH solutions (i.e. upper bounds) to the lower bounds determined by the literature 1-tree method (Lawler, Lenstra, Rinnoy Kan, & Shmoys, 1985).

Parallel Tabu Search Algorithm. Differentiation of the initial solutions was obtained by running a certain local search algorithm with a small number of iterations $10r$, $r = 1, 2, \ldots, p$, where p denotes the number of processors. The tabu search algorithm was run for a fixed number of iterations $it = 1000$. As a final solution there was selected the best solution (in terms of objective function value) out of the solutions determined by individual processors. In Table 5 there were the average values of percentage relative deviation (PRD) presented in reference to the reference solutions.

On the basis of the obtained results it is possible to state that the parallel algorithm determines much better solutions. Fig. 3 shows the relationship between the PRD and the number of processors. It is easy to notice that the PRD value decreases with an increasing number of processors. In this way we get better and better solutions.

Parallel Simulated Annealing Algorithm. The process of implementation of parallel simulated annealing algorithm can be achieved in several ways. To solve the presented in the work problem there was a strategy used in which individual processors execute calculations independently.

Differentiation of the initial solutions was obtained by changing the number of iterations in the process of determination of the initial temperature – $rn^2/4$, where $r = 1, 2, \ldots, p$, and p the number of processors. Simulated annealing algorithm was run for a fixed number of iterations – $it = 1000$. As a final solution there was the best solution (in terms of the objective function value) chosen out of the solutions determined by each of the processors. The average percentage relative deviations (improved solutions determined by the NEH) are provided in Table 6.

On the basis of the obtained results it can be concluded that, like in case of TS, the application of parallel simulated annealing algorithm for the cyclic flow shop problem allows one to achieve better results (in terms of objective function value) than for sequential algorithm. The dependence of PRD on the number of processors is shown in Fig. 3. The presented dependency shows that increasing the number of processors allows for a reduction in the value of the percentage relative deviation, i.e. bigger improvement of the reference solution.

7. Conclusions

In the work there was parallel algorithm determining blocks for cyclic flow shop problem demonstrated. The performed computational experiments showed a significant, even almost 10-fold acceleration of the procedures for designating blocks. Next, the TS and SA algorithms solving a cyclic flow shop problem with machine setups were presented.

Parallel implementations of algorithms executed MPSS (*Multiple Initial Point Single Strategy*) were proposed, in which the processors start operation from different starting solutions using the same search strategy. The performed computational experiments (for different numbers of processors) showed significant improvement of reference solutions determined by NEH algorithm. Increasing the number of processors directly improved the values of determined solutions.

Block properties proposed here and applied the tabu search and simulated annealing methods can be used in designing of any efficient metaheuristic algorithms which uses neighborhoods. Additionally, parallel block determination can be used for the neighborhood searching acceleration in local optima determination of such metaheuristics as memetic algorithms, scatter search, etc.

References

Bożejko, W. (2010). *A new class of parallel scheduling algorithms*. Wrocław University of Technology Publishing House. 1–280. <http://www.dbc.wroc.pl/publication/11299>.

Bożejko, W., Gniewkowski, Ł., Pempera, J., & Wodecki, M. (2014). Cyclic hybrid flow-shop scheduling problem with machine setups. *Procedia Computer Science, 29*, 2127–2136.

Bożejko, W., Pempera, J., & Wodecki, M. (2015). Parallel simulated annealing algorithm for cyclic flexible job shop scheduling problem. In *Lecture notes in artificial intelligence no. 9120* (pp. 603–612). Springer.

Bożejko, W., Uchroński, M., & Wodecki, M. (2015). Block approach to the cyclic flow shop scheduling. *Computers & Industrial Engineering, 81*, 158–166.

Bożejko, W., & Wodecki, M. (2007). On the theoretical properties of swap multimoves. *Operations Research Letters, 35*(2), 227–231.

Brucker, P., Burke, E. K., & Groenemeyer, S. (2012). A branch and bound algorithm for the cyclic job-shop problem with transportation. *Computers & Operations Research, 39*(12), 3200–3214.

Gertsbakh, I., & Serafini, P. (1991). Periodic transportation schedules with flexible departure times. *European Journal of Operational Research, 50*, 298–309.

Grabowski, J., & Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers & Operations Research, 31*, 1891–1909.

Kats, V., & Levner, E. (2010). A fast algorithm for a cyclic scheduling problem with interval data. In *Proceedings of the annual operations research society of Israel (ORSIS-2010) conference, February 2010, Nir Etzion, Israel.*

Lawler, E. L., Lenstra, J. K., Rinnoy Kan, A. H. G., & Shmoys, D. B. (1985). *The traveling salesman problem. Wiley-Interscience series in discrete mathematics and optimization*.New York: John Wiley & Sons.

Mendez, C. A., Cerda, J., Grossmann, I. E., Harjunkoski, I., & Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers and Chemical Engineering, 30*, 913–946.

Nawaz, M., Enscore, E. E., Jr, & Ham, I. (1983). A heuristic algorithm for the m–machine, n–job flow–shop sequencing problem. *OMEGA International Journal of Management Science, 11*, 91–95.

Pinedo, M. (2005). *Planning and scheduling in manufacturing and services*.New York: Springer.

Pinedo, M. (2008). *Scheduling: Theory, algorithms and systems*.New York: Springer.

Pinto, T., Barbosa-Póvoa, A. P. F. D., & Novais, A. Q. (2005). Optimal design and retrofit of batch plants with a periodic mode of operation. *Computers and Chemical Engineering, 29*, 1293–1303.

Ruiz, R., & Stützle, T. (2008). An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research, 187*(3), 1143–1159.

Voß, S. (1993). Tabu search: Applications and prospects. In D. Z. Du & P. Pardalos (Eds.), *Network optimization problems* (pp. 333–353). Singapore: World Scientific Publishing Co.