10.1515/acsc-2017-0011

Archives of Control Sciences
Volume 27(LXIII), 2017
No. 2*, pages 183–195*

Parallel patterns determination in solving cyclic flow shop problem with setups

WOJCIECH BOŻEJKO, ZENON CHACZKO, MARIUSZ UCHROŃSKI and MIECZYSŁAW WODECKI

The subject of this work is the new idea of blocks for the cyclic flow shop problem with setup times, using multiple patterns with different sizes determined for each machine constituting optimal schedule of cities for the traveling salesman problem (TSP). We propose to take advantage of the Intel Xeon Phi parallel computing environment during so-called 'blocks' determination basing on patterns, in effect significantly improving the quality of obtained results.

Key words: cyclic scheduling, parallel algorithm, metaheuristics

1. Introduction

In recent years there has been an observed growth in interest in cyclic problems of tasks scheduling, both in the circle of theorists dealing with discrete optimization problems and in the population of industry practitioners. Cyclic production is, in fact, a very effective mode of production in modern flexible manufacturing systems. In the available literature there are many studies on different aspects of cyclic control in companies that produce goods on a mass scale. There are examples of the application of cyclic scheduling in various spheres of industry, transport and logistics (e.g. Pinto et al. [12], Pinedo [11], Mendez et al. [9], Gertsbakh and Serafini [6], Kats and Levner [8]). Unfortunately, the existing models and calculation tools allow one to determine the optimal (minimizing cycle time) control of production systems executing only a small number of tasks.

In this work we considered a cyclic flow shop problem with setup times. Strong NP-hardness of many simplest versions of the cyclic scheduling problem (Smutnicki [14]), in particular of the considered problem, limits the scope of application of exact algorithms

W. Bożejko (corresponding author), and M. Uchroński are with Department of Automatics, Mechatronics and Control Systems, Faculty of Electronics, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland. E-mails: {wojciech.bozejko, mariusz.uchronski}@pwr.edu.pl. Z. Chaczko is with Faculty of Engineering and Information Technology (FEIT), University of Technology, Sydney(UTS), Australia, NSW, Sydney, e-mail: zenon.chaczko@uts.edu.au and M. Wodecki is with Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland, e-mail: mieczyslaw.wodecki@uwr.edu.pl

Received 14.12.2016. Revised 30.04.2017.

in instances with low number of tasks, but in the context of minimizing the design cycle time and the use of exact algorithms, it seems to be fully justified (Brucker et al. [5]). However, due to the NP-hardness, to determine satisfactory solutions there are common fast approximate algorithms used, based on local search techniques , such as: simulated annealing (in parallel version Bożejko et al. [2]) or tabu search (Bożejko et al. [3]). Methods of this type are usually based on a two-level decomposition of the problem: determination of the optimum order of tasks (upper level) and multiple determination of the minimum criteria for a given sequence of tasks (lower level).

While for classic, non-cyclic scheduling problems, a solution to the lower level problem can be obtained in a time-efficient manner by analyzing a specific graph, in case of such a posed problem the solution to lower level is relatively time-consuming because it generally requires solving oflinear programming problem. Therefore, any special properties, including these allowing for more efficient calculation of the cycle time, the search of schedule and limiting the cardinality of locally searched neighborhood or acceleration of the speed of the viewing are very desirable.

In the presented paper, we are proposing the use of new eliminating properties, including the so-called patterns to reduce the number of solutions viewed when generating neighborhood with local search algorithms, such as the tabu search with prohibitions or simulated annealing. Determination of the patterns may be performed either sequentially, or in parallel, using a multiprocessing computations environment. Relevant properties were formulated using the PRAM model machine which is standard for the theoretical verification of the computational complexity of parallel algorithms.

The reminder of hte paper is organized as follows. Section 2 presents a problem description. Its mathematical model is proposed in the section 3. Section 4 contains proposed solution methid in which we define adn use blocks of tasks. Results of computational experiments are discussed in section 5. Conclusions and comments are presented in the Section 6.

2. Problem description

The problem of cyclic production considered in this work may be formulated as follows: there is a set of n tasks given $\mathcal{J} = \{1,2,\ldots,n\}$, to be performed cyclically (repeatedly) on machines from the set $\mathcal{M} = \{1,2,\ldots,m\}$. Any task should be performed in sequence on each of m machines $1,2,\ldots,m$ (technological line). Task $j \in \mathcal{J}$ is a sequence m operations $O_{1,j}, O_{2,j}, \ldots,\ O_{m,j}$. Operation $O_{k,j}$ corresponds to the activity of 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, and before the start of the next operation a machine setup is performed. Let $s^k_{i,j}$ $(k \in \mathcal{M},\ i \neq j\ i,j \in \mathcal{J})$ be the setup time between operation $O_{k,i}$ and $O_{k,j}$.

A set of tasks in a single cycle is called a MPS (*minimal part set*). MPSs are processed cyclically, one by one.

One should determine the order of tasks (the same for each machine), which minimizes cycle time, i.e. the moment of commencement of tasks execution from the set $\mathcal{J}$ in the next cycle. In applying this, the following constraints must be met:

(a) each operation can only be performed by one machine,

(b) no machine can perform more than one operation at the same time,

(c) the order of the technological line of operations execution must be preserved,

(d) execution of any operation cannot be interrupted before its completion,

(e) each machine, between sequentially performed operations, requires setups,

(f) each operation is executed sequentially (in successive MPSs) after the cycle time is completed.

The considered problem boils down to determining the starting moments of the tasks execution on machines that meet the constraints (a) - (f), so that the cycle time (the time after which the task is performed in a subsequent MPS) is minimised.

We assume that in each of the MPS, on each machine, the tasks are executed in the same order. Therefore, in a cyclic schedule, the order of tasks execution on machines can be represented by a permutation of the tasks in the first MPS. On this basis we can determine the beginning moments of tasks execution on machines in the first MPS. Increasing them by multiples of the cycle time, we obtain the beginning moments of tasks execution in any of MPSs (commencement of execution of any of the operation in the consecutive MPS should be increased by the cycle time). Let Φ be the set of all permutations of the elements from a set of tasks $\mathcal{J}$. Therefore, the considered in the work problem comes down to the determination of permutation of tasks (element from the set Φ) minimizing the length of cycle time. In short the problem will be denoted by CFS (*Cyclic Flow Shop*).

3. Mathematical model

Let $[S^k]_{m\times n}$ be a matrix of beginning moments of tasks execution of k-th MPS (for the established order $\pi \in \Phi$), where $S^k_{i,j}$ denotes the beginning time moment of execution of task j on i machine. We assume that tasks in the following MPSs are performed cyclically. This means that there is a fixed $T(\pi)$ (period) such that

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

Period $T(\pi)$ depends of course on permutation π and is called *cycle time* of the system. The minimum value $T(\pi)$, for a fixed π, will be called *minimum cycle time* and denoted by $T^*(\pi)$. Since the order of tasks execution in any MPS is the same, it is enough to designate a sequence of tasks π for one (*the first*) MPS and make its shift by $k \cdot T(\pi)$, k=1,2,... on timeline. For a fixed order of execution of tasks $\pi \in \Phi$, optimum value of cycle time $T^*(\pi)$ can be determined by solving the corresponding linear programming task (see Bożejko et al. [1]). For any order of tasks execution in the first MPS, solving the above linear programming task, there can be the minimum time cycle in polynomial time determined. In case of an exact algorithm's (exhaustive search) solution to CFS problem should be therefore done for each of $n!$ permutations – elements of the set Φ. For real-life instances sizes it will be impossible due to computations time, so in the next chapter we propose an approximate method for solving the CFS problem .

4. Solution method

In many heuristic algorithms solving NP-hard problems, there are neighborhoods viewed, i.e. subsets of solution space. In case where solutions to the problem are permutations, usually the neighborhoods are generated by insert- or swap-type moves and their compositions [4]. They consist of changing positions of elements in the permutation. The number of elements of such neighborhood is at least $n(n-1)/2$, where n is the number of tasks. In practical applications (with large n), neighborhood viewing is the most time consuming part of the algorithm. It follows from the descriptions in the literature concerning computational experiments that the number of iterations of the algorithm has a direct impact on the quality of designated solutions. Hence, there is the search of methods accelerating action of a single iteration of the algorithm. One of them relies in the reduction of the number of elements of the neighborhood, their parallel generation and viewing. In the case of tasks scheduling problems on multiple machines with the minimization of tasks execution time (C_{max}) 'block eliminating properties' are successfully used [7]. Similar properties are to be applied in the algorithm solving the problem of determining minimum cycle time, more specifically - a minimum time of running of a single machine. They allow its users to eliminate elements from the neighborhood that do not directly provide animprovement on the best solution found so far.

The work [3] describes a method solving the CFS problem and a sequential algorithm of search with prohibitions. In a further part of the chapter there is a summary of the main elements of the method presented.

For a set permutation $\pi \in \Phi$ and machine $k \in \mathcal{M}$

$$T_k(\pi) = \sum_{i=1}^{n-1} (p_{k,\pi(i)} + s^k_{\pi(i),\pi(i+1)}) + p_{k,\pi(n)} + s^k_{\pi(n),\pi(1)} \tag{2}$$

is the time of execution of tasks in the order π, with setups performed between task $\pi(n)$, and $\pi(1)$ (i.e. the last task in the given MPS, and the first in the next). It is easy to prove that the minimum cycle time

$$T^*(\pi) = \max\{T_i(\pi) : i = 1, 2, \ldots, m\}. \tag{3}$$

Property 1 ([3]). *The necessary condition reducing the value of the minimum cycle time $T^*(\beta)$ is shortening of the running time of k-th machine, i.e. reducing of $T_k(\beta)$, where $T^*(\beta) = T_k(\beta)$.*

Designation of the minimum running time of k-th machine, i.e. the value $\min\{T_k(\delta) : \delta \in \Phi\}$ can be reduced to following traveling salesman problem.

Let $H_k = (\mathcal{V}, \mathcal{E}; p, s)$ be a complete graph, where

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

Property 2 ([3]) *The running time of k-th machine $T_k(\pi)$ is equal to the length (i.e. sum of the weights of vertices and edges) of Hamiltonian cycle $(\pi(1), \pi(2), \ldots, \pi(n), \pi(1))$ in graph H_k.*

Property 3 ([3]) *Minimum running time of k-th machine is equal to the length of the salesman path in the graph H_k, i.e. the minimum (due to the length) Hamiltonian cycle.*

Let π^*_k be the optimal salesman path way in a graph H_k $(k = 1, 2, \ldots, m)$, which is also an optimal (i.e. minimal due to the execution time) order of the tasks from the set $\mathcal{J}$ on k-th machine. This permutation will be called *a pattern* for k-th machine.

In order to reduce the running time of k-th machine $T_k(\pi)$ from π there will be permutations generated, taking into account the existence of the individual elements in the pattern. Patterns also allow to eliminate the elements of the neighborhood which do not give an improvement of the current value of cycle time in local search algorithms.

4.1. Tasks blocks

Let
$$B = (\pi(a), \pi(a+1), \ldots, \pi(b)), \tag{4}$$
be a sequence of occurring immediately after another tasks in permutation $\pi \in \Phi$, π^*_k *pattern* for k-th machine and u, v $(u \neq v, \; 1 \leqslant u, v \leqslant n)$ a 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 can be enlarged neither by an element $\pi(a-1)$, nor by $\pi(b+1)$, satisfying the constraints **W1** or **W2**).

If the sequence of tasks (4) satisfies the conditions **W1** and **W3** or **W2** and **W3**, then it is called a *block* on k-th machine ($k \in \mathcal{M}$).

Below there is presented a sequential algorithm for determining all of the blocks in the permutation.

Algorithm 1. ***Alg_SeqBlock***

```
$\pi = (\pi(1), \pi(1), \ldots, \pi(n))$ - 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 blocks starting positions in $\pi$;
$t \leftarrow 1; i \leftarrow 1;$
while $(i \leqslant n)$ do
   $b_t \leftarrow i; q \leftarrow (\pi^*)^{-1}(\pi(i));$
   while $(\pi(i) = \pi^*(q))$ do
      $q \leftarrow q+1; i \leftarrow i+1;$
      $i \leftarrow i+1;$
```

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

Determination of the pattern (the optimal salesman path in the graph H_k) is an NP-hard problem. Therefore, there will be approximate algorithms used, e.g. 2-opt. For each machine the pattern will be determined before starting the proper algorithm.

4.2. Parallel determination of blocks

To speed up the running of the algorithm for determining the minimum cycle time, we present a method of parallelization of the most time-consuming procedures of determining blocks performed in each iteration.

Property 4 *Determination of blocks for cyclic flow shop problem with setups can be done in time $O(\log n)$ on mn-processor CREW PRAM machine.*

The method of determining the blocks is presented in the Algorithm 2.

Algorithm 2. ***Alg_pblocks***
Input: permutations: $\pi = (\pi(1), \pi(1), \ldots, \pi(n))$
and $\pi^* = (\pi^*(1), \pi^*(1), \ldots, \pi^*(n))$
Output: vector of blocks starting positions $(b_1, b_2, \ldots, b_k)$, k – number of blocks and vector of blocks ending positions $(e_1, e_2, \ldots, e_k)$

Sample input:

$$\pi = (8, 10, 7, 4, 5, 6, 3, 1, 9, 2)$$

$$\pi^* = (9, 3, 1, 6, 10, 7, 4, 5, 2, 8)$$

$$k = 2,\ b = (2, 7),\ e = (5, 8)$$

$$B_1 = (10,7,4,5),\ B_2 = (3,1)$$

```
**Step 1.** **parfor** $r \in \{1,2,\ldots,p\}$ **do**
  $\pi(0) := \pi(n+1) := \pi^*(0) := \pi^*(n+1) := -1$;
  $(\pi^*)^{-1}(0) := (\pi^*)^{-1}(n+1) := -1$;
  **if** (preceding position in $\pi$ is identical as in $\pi^*$, i.e.
    $\pi(r-1) = \pi^*((\pi^*)^{-1}(\pi(r))-1))$ **then**
      $B_b[r] := 1$;
        **else**
      $B_b[r] := 0$;
  **if** (next position in $\pi$ is identical as in $\pi^*$, i.e.
    $\pi(r+1) = \pi^*((\pi^*)^{-1}(\pi(r))+1))$ **then**
      $B_e[r] := 1$;
        **else**
      $B_e[r] := 0$;
**Step 2.** **parfor** $r \in \{1,2,\ldots,p\}$ **do**
    **if** $((B_b[r] = 0)$ and $(B_b[r+1] = 1))$ **then**
        $B_b[r] := 1$;
          **else**
        $B_b[r] := 0$;
```

for sample input:

$$B_b = (0,1,0,0,0,0,1,0,0,0)$$

```
**Step 3.** **parfor** $r \in \{1,2,\ldots,p\}$ **do**
    **if** $((B_e[r] = 0)$ and $(B_e[r-1] = 1))$ **then**
        $B_e[r] := 1$;
          **else**
        $B_e[r] := 0$;
```

for sample input:

$$B_e = (0,0,0,0,1,0,0,1,0,0)$$

Step 4. Determine the prefix sum of P elements from table B_b, i.e.

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

for sample input:

$$B_b = (0,1,0,0,0,0,1,0,0,0)$$

$$P = (0,1,1,1,1,1,2,2,2,2)$$

```
**Step 5.** **parfor** $r \in \{1,2,\ldots,p\}$ **do**
    **if** $(B_b[r] = 1)$ **then**
        $b[P[r]] := r$;
    **if** $(B_e[r] = 1)$ **then**
        $e[P[r]] := r$;
```

for sample input:

$$B_e = (0,0,0,0,1,0,0,1,0,0)$$

$$B_b = (0,1,0,0,0,0,1,0,0,0)$$

$$P = (0,1,1,1,1,1,2,2,2,2)$$

$$b = (2,7),\ e = (5,8)$$

Fig. 1 shows the implementation of Algorithm 2 in the form of `pblocks` procedure of block determination with n processors using OpenMP parallelization library. This algorithm was then run on the Xeon Phi coprocessor. The input data are: the size of permutations in which blocks `n`, blocks ttn, a permutation `pi` and permutation – a pattern `pi_ptr` are determined. Tables `b_b` and `b_e`, after the completion of the procedure, contain the beginning and ending positions of the following blocks.

```
int pblocks(int n, int *pi, int *pi_ptr,
  int *b_b, int *b_e) {
 int *pi_ptr_ = new int[n+2];
 int *b = new int[n+1];
 int *b_ = new int[n+1];
 int *bk = new int[n+1];
 int *p = new int[n+2];
 int *p_ = new int[n+2];
 pi_ptr[0] = pi_ptr[n+1] = -1;
 zeros_pi(n, pi_ptr_);
 pi_ptr_[0] = pi_ptr_[n+1] = -1;
 #pragma omp parallel for
 for(int i=1; i<=n; ++i)
  pi_ptr_[pi_ptr[i]] = i;
 #pragma omp parallel for
 for(int i=1; i<=n; ++i)
 {
  if(pi[i-1]==pi_ptr[pi_ptr_[pi[i]] - 1])
   b[i] = 1; else b[i] = 0;
  if(pi[i+1]==pi_ptr[pi_ptr_[pi[i]] + 1])
   bk[i] = 1; else bk[i] = 0;
  p[i] = p_[i] = 0;
 }
 p[1] = b[1]; p[0] = 0;
 #pragma omp parallel for
 for(int i=1; i<=n; ++i)
  if(b[i] == 0 and b[i+1] == 1)
   b[i] = 1; else b[i] = 0;
 #pragma omp parallel for
 for(int i=n; i>=1; --i)
  if(bk[i] == 0 and bk[i-1] == 1)
   bk[i] = 1; else bk[i] = 0;
 for(int i=2; i<=n; ++i)
  p[i] = p[i-1] + b[i];
 int nblocks = 0;
 #pragma omp parallel for
 for(int i=1; i<=n; ++i)
  if(b[i] == 1)
   b_b[p[i]] = i;
  if(bk[i] == 1)
   b_e[p[i]] = i;
 #pragma omp parallel for
 for(int i=1; i<=n; ++i)
 {
  #pragma omp critical
  if(b_b[i] != 0 or b_e[i] != 0)
   nblocks++;
 }
 #pragma omp parallel for
 for(int i=1; i<=nblocks; ++i)
  if(b_b[i] == 0 and b_e[i] != 0)
   b_b[i] = 1;
  if(b_e[i] == 0 and b_b[i] != 0)
   b_e[i] = n;

 delete[] pi_ptr_; delete[] b; delete[] b_;
 delete[] p; delete[] p_;
 return nblocks;
}
```

Figure 1: Function of the parallel block determination.

5. Computational experiments

A parallel procedure for the blocks determination has been implemented in C++ using OpenMP library. The data for computational experiments (permutations of tasks) were generated randomly. The number of elements of permutations changed in range from 10^3, to 10^7. Computational experiments were carried out in a computing environment with shared memory - the coprocessor Intel Xeon Phi 3120 (space 6GB, 1.1 GHz) enabling the use of 228 cores.

The aim of the first phase of the experiments was to demonstrate the effectiveness of the (sequential) mechanism of blocks based on patterns. For this purpose, a procedure for blocks determining was placed in the classic *tabu search* (TS), algorithm with prohibitions with tabu list determined of length 7. There were two versions of the algorithm run - with and without blocks, for a predefined number of 1000 iterations. There was *Percentage Relative Deviation* (PRD), studied to solve the reference solution obtained with the use of NEH algorithm (Nawaz i in. [10]) for test data from work [13]. The results reported in Table 1, indicate that with nearly a twofold shorter time of the algorithm running, the obtained results were much better (32.8% improvement over the NEH) as compared to the version of the algorithm running without the block mechanism (29.0% improvement over the NEH).

Table 1: Comparison of PRD to NEH for TS algorithm with and without the blocks - 1,000 iterations.

$n \times m$	$t[s]$	$t_B[s]$	PRD	PRD_B
20×5	2.2	0.8	-30.0	-32.7
20×10	3.0	0.9	-29.5	-31.6
20×20	4.7	1.2	-29.8	-30.7
50×5	35.7	17.7	-32.5	-34.1
50×10	48.4	22.0	-29.6	-34.5
50×20	73.9	28.3	-28.3	-31.8
100×5	292.1	181.8	-30.7	-36.7
100×10	391.9	203.8	-28.1	-33.6
100×20	604.7	266.4	-27.9	-31.6
200×10	3212.2	1791.1	-26.7	-32.9
200×20	5255.5	2497.7	-26.2	-31.1
Average	902.2	455.6	-29.0	-32.8

The second phase of the experiment consisted of measuring the acceleration of the procedure for the parallel determining of the blocks. The results of computational experiments for the co-processor Intel Xeon Phi 3120 were shown in Tables 2 and 3 and presented in Fig. 2 and 3. For a different number of tasks in permutation the acceleration initially increases quite rapidly, reaches a maximum, and then decreases slowly. The number of processors for which the maximum acceleration is reached depends on the size of the problem. Using the notion of *scalability* of parallel algorithms one can say that for $5000 \cdot 10^3$ tasks in a permutation parallel method for block determination is characterized for $p = 1, 2, \ldots, 32$ with a strong scalability, since with an increase in the number of processors the speedup increases (Fig. 3).

Table 2: Speedup of `pblocks` procedure.

$\lambda^{(1)}$	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$
1	0.003	0.001	0.001	0.001	0.001
2	0.008	0.004	0.003	0.004	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	3.431	4.160	4.880
10000	1.398	2.584	3.612	4.461	5.266

(1) $\lambda = n \cdot 10^3$, coprocessor Intel Xeon Phi 3120A

Table 3: Speedup of `pblocks` procedure.

$\lambda^{(1)}$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
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
10000	7.091	9.766	10.775	9.679

(1) $\lambda = n \cdot 10^3$, coprocessor Intel Xeon Phi 3120A

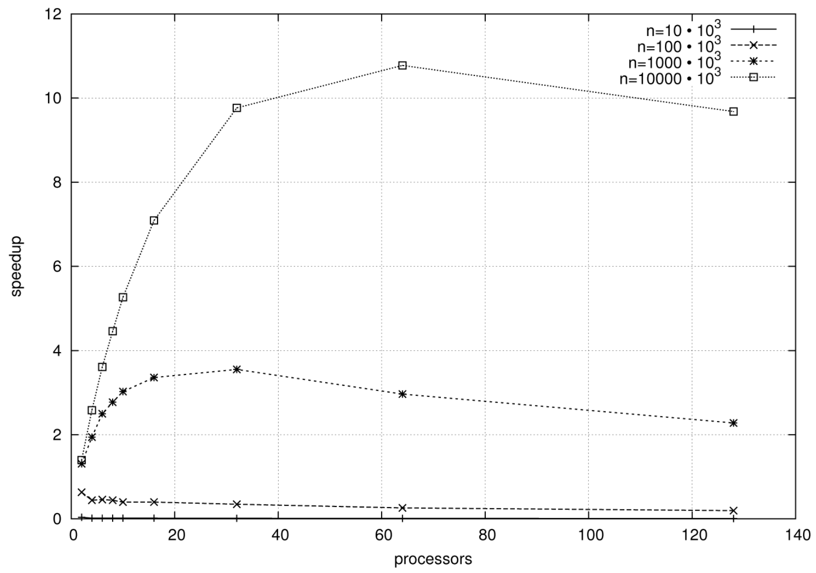

Figure 2: Dependency of speedup on the number of processors – Intel Xeon Phi 3120A.

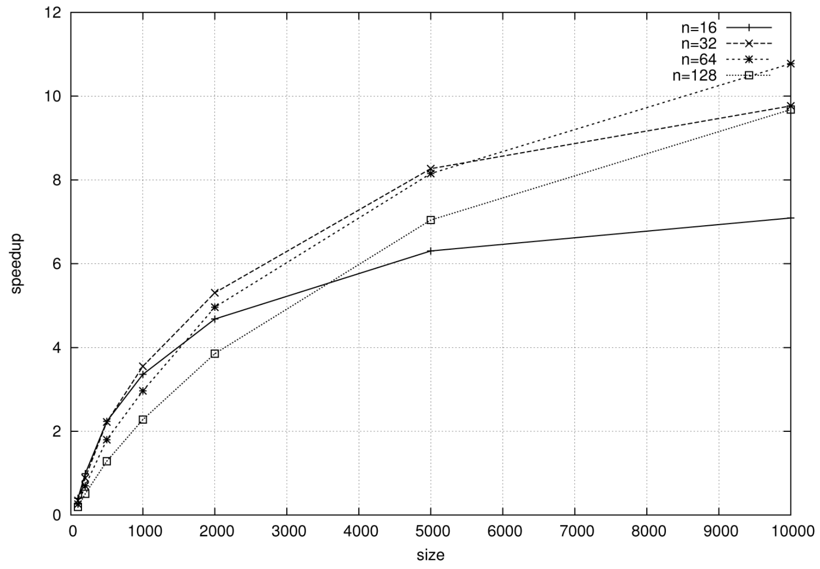

Figure 3: Dependency of speedup on the number of tasks – Intel Xeon Phi 3120A.

6. Conclusions and comments

This paper presents a new concept of blocks for a cyclic flow shop problem with machine setups, using multiple patterns of different sizes. Patterns represent the optimal order of visiting cities in a traveling salesman problem, ensuring block properties in the problem. The use of the block properties enables for a significant reduction in the number of viewed neighbors in metaheuristic algorithms based on viewing the neighborhoods. Future work in the work's field could be focused on the extension of the pattern approach (which creates blocks in a solution) by researching a distance measure based on the pattern-based neighborhood.

References

[1] Bożejko W., Uchroñski M., Wodecki M. (2016). Parallel metaheuristics for the cyclic flow shop scheduling problem. Computers & Industrial Engineering, 95, 156–163.

[2] Bożejko, W., Pempera, J., Wodecki, M. (2015). Parallel Simulated Annealing Algorithm for Cyclic Flexible Job Shop Scheduling Problem. Lecture Notes in Artificial Intelligence No. 9120, Springer, 603–612.

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

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

[5] 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.

[6] Gertsbakh, I., Serafini, P. (1991). Periodic transportation schedules with flexible departure times. European Journal of Operational Research, 50, 298-309.

[7] 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.

[8] 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.

[9] Mendez, C. A., Cerda, J., Grossmann, I. E., Harjunkoski, I., Fahl, M. (2006). Stateofthe-art review of optimization methods for short-term scheduling of batch processes. Computers and Chemical Engineering, 30, 913-946.

[10] Nawaz, M., Enscore, Jr, E.E., 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.

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

[12] Pinnto, T., Barbosa-Povoa, 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.

[13] 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.

[14] Smutnicki, C., New features of the cyclic job shop scheduling problem. In Proceedings of 20th International Conference on Methods and Models in Automation and Robotics MMAR 2015, IEEE Press, 1000–1005.