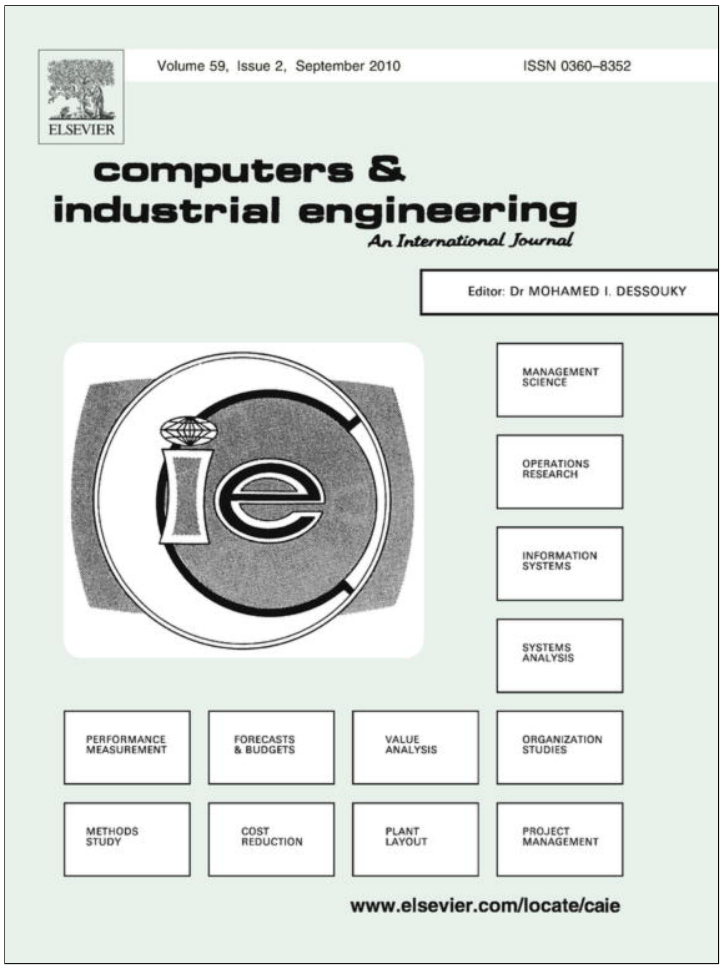

Computers & Industrial Engineering 59 (2010) 323–333

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Parallel hybrid metaheuristics for the flexible job shop problem ☆

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

[a] Wrocław University of Technology, Institute of Computer Engineering, Control and Robotics, Janiszewskiego 11-17, 50-372 Wrocław, Poland
[b] University of Wrocław, Institute of Computer Science, Joliot-Curie 15, 50-383 Wrocław, Poland

ARTICLE INFO

Article history:
Received 16 October 2009
Received in revised form 22 April 2010
Accepted 6 May 2010
Available online 10 May 2010

Keywords:
Scheduling
Flexible job shop problem
Hybrid metaheuristics
Tabu search
Population-based algorithm

ABSTRACT

A parallel approach to flexible job shop scheduling problem is presented in this paper. We propose two double-level parallel metaheuristic algorithms based on the new method of the neighborhood determination. Algorithms proposed here include two major modules: the machine selection module refer to executed sequentially, and the operation scheduling module executed in parallel. On each level a metaheuristic algorithm is used, therefore we call this method meta^{2}heuristics. We carry out a computational experiment using Graphics Processing Units (GPU). It was possible to obtain the new best known solutions for the benchmark instances from the literature.

1. Introduction

The flexible job shop problem constitutes a generalization of the classic job shop problem where operations have to be executed on one machine from a set of dedicated machines. Then, as a job shop problem it also belongs to the strongly NP-hard class. Although exact algorithms based on a disjunctive graph representation of the solution have been developed (see Pinedo (2002)), they are not effective for instances with more than 20 jobs and 10 machines. Many approximate algorithms, mainly metaheuristics, have been proposed. Hurink, Jurisch, and Thole (1994) developed the tabu search method for this problem. Also Dauzère-Pérès and Pauli (1997) used the tabu search approach extending the disjunctive graph representation for the classic job shop problem taking into consideration the assignment of operations to machines. Mastrolilli and Gambardella (2000) proposed a tabu search procedure with effective neighborhood functions for the flexible job shop problem. Many authors have proposed a method of assigning operations to machines and then determining sequence of operations on each machines. Such an approach is followed by Brandimarte (1993) and Pauli (1995). These authors solved the assignment problem (i.e. using dispatching rules) and next used metaheuristics to solve the job shop problem. Also genetic approaches have been adopted to solve the flexible job shop problem. Recent works are these of Jia, Nee, Fuh, and Zhang (2003), Ho and Tay (2004), Kacem, Hammadi, and Borne (2002) Pezzella, Morganti, and Ciaschetti (2008). Gao, Sun, and Gen (2008) proposed the hybrid genetic and variable neighborhood descent algorithm for this problem. There are only a few papers considering parallel algorithms for the FJSP. Yazdani, Amiri, and Zandieh (2010) propose a parallel variable neighborhood search (VNS) algorithm for the FJSP based on independent VNS runs. Defersha and Chen (2009) describe a coarse-grain version of the parallel genetic algorithm for the considered FJSP basing on island-model of parallelization focusing on genetic operators used and scalability of the parallel algorithm. Both papers are focused on parallelization side of the programming methodology and they do not use any special properties of the FJSP.

In this paper we propose a parallel double-level metaheuristic approach for the flexible job shop problem based on two methods implemented on the higher level: tabu search and population-based approach. Additionally we have implemented the new method of the neighborhood determination (so-called *t-moves*) for the considered problem. We apply INSertion Algorithm INSA, (Nowicki & Smutnicki, 1996) and Tabu Search Algorithm with Backtracking TSAB, (Nowicki & Smutnicki, 1996) on the second level of parallelism. Using this method we have obtained the new best known solutions for the benchmark instances of Barnes and Chambers (1996) from the literature.

2. Hybrid metaheuristics

A hybrid approach to solving difficult optimization problems by using several metaheuristics simultaneously makes possible to

☆ This manuscript was processed by Area Editor Maged M. Dessouky.

* Corresponding author. Tel.: +48603672142.
E-mail addresses: wojciech.bozejko@pwr.wroc.pl (W. Bożejko), mariusz.uchronski@pwr.wroc.pl (M. Uchroński), mwd@ii.uni.wroc.pl (M. Wodecki).

0360-8352/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2010.05.004

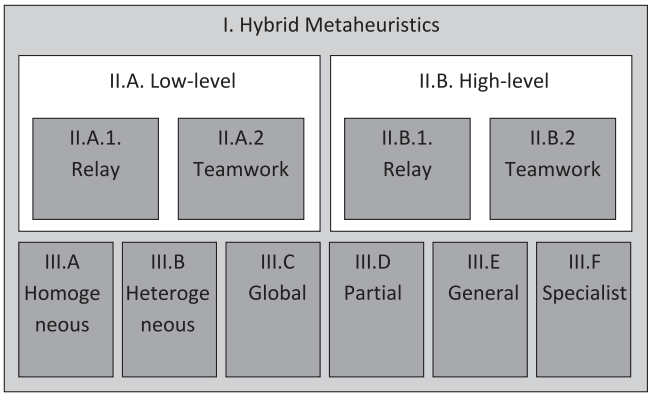

Fig. 1. Classification of hybrid metaheuristics proposed by Talbi (2002).

take advantages of all of them. Talbi (2002) provides a systematic characterization of parallel hybrid metaheuristics, which is visualized on the Fig. 1.

The upper part of the figure presents the hierarchical structure of the hybridization. In *high-level* hybrid algorithms the different metaheuristics are self-contained – there is no direct relationship to the internal workings of a metaheuristic. The *low-level* hybridization addresses the functional composition of a single optimization method – a given function of a metaheuristics is replaced by another metaheuristics. A *teamwork* hybridization represents cooperative models of optimization in which many parallel agents cooperate and each agent makes a search in its own part of the solution space. On the other hand, in *relay* hybrids, a number of metaheuristics is applied one after another; each one uses the output of the previous one as its own input, as in pipeline.

The lower part of the Fig. 1 (so-called a flat part) specifies the features of algorithms involved in the hybrid. In *homogeneous* hybrids all the combined algorithms use to same metaheuristic methods. On the contrary, in *heterogeneous* algorithms, different metaheuristics are used. In *global* hybrids, all the algorithms search in the whole solution space – the goal is to explore the space more thoroughly. All the algorithms solve the whole optimization problem. On the other hand, in *partial* hybrids, the considered problem is decomposed into subproblems; each one having its own solution space. Each algorithm is dedicated to search in one of these subspaces. Subproblems are linked with each other involving constraints between optima which are found by each algorithm. Algorithms establish communication to respect these constraints and create a solution of the main problem. In *general* hybrids, all the algorithms solve the same target optimization problem. *Specialist* hybrids combine algorithms which solve different problems, i.e. by solving another optimization problem into which the main problem is transformed.

Both algorithms proposed in this paper belong to the *high-level teamwork homogeneous general* hybrid metaheuristics. These algorithms possess two major modules: the machine selection module which is executed sequentially, and the operation scheduling module executed in parallel. Metaheuristics connected with each module are executed one after another, acting in a pipeline fashion. The proposed algorithms belong to the *partial* hybrids, because the problem to be sold is decomposed into subproblems connected with operation-machine assignments.

3. Flexible job shop problem

The flexible job shop problem (FJSP), also called the general job shop problem with parallel machines, can be formulated as follows. Let $\mathscr{J} = \{1, 2, \ldots, n\}$ be a set of jobs which have to be executed on machines from the set $\mathscr{M} = \{1, 2, \ldots, m\}$. There exists a partition of the set of machines into types, i.e. subsets of machines with the same functional properties. A job constitutes a sequence of some operations. Each operation has to be executed on an adequate type of machine (nest) within a fixed time. The problem consists in the allocation of jobs to machines from the adequate type and the schedule of jobs execution determination on each machine to minimize the total job finishing time. The following constrains have to be fulfilled:

(i) each job has to be executed on only one machine of a determined type in each moment of time,
(ii) machines can not execute more than one job in each moment of time,
(iii) there are no idle times (i.e. the job execution must not be broken),
(iv) the technological order has to be obeyed.

Let $\mathscr{O} = \{1, 2, \ldots, o\}$ be the set of all operations. This set can be partitioned into sequences corresponding to jobs where the job $j \in \mathscr{J}$ is a sequence of o_j operations which have to be executed in an order on dedicated machines (i.e. in the so-called technological order). Operations are indexed by numbers $(l_{j-1} + 1, \ldots, l_{j-1} + o_j)$ where $l_j = \sum_{i=1}^{j} o_i$ is the number of operations of the first j jobs, $j = 1, 2, \ldots, n$, where $l_0 = 0$ and $o = \sum_{i=1}^{n} o_i$.

The set of machines $\mathscr{M} = \{1, 2, \ldots, m\}$ can be partitioned into q subsets of the same type (*nests*) where ith ($i = 1, 2, \ldots, q$) type $\mathscr{M}^i$ includes m_i machines which are indexed by numbers $(t_{i-1} + 1, \ldots, t_{i-1} + m_i)$, where $t_i = \sum_{j=1}^{i} m_j$ is the number of machines in the first i types, $i = 1, 2, \ldots, q$, where $t_0 = 0$ and $m = \sum_{j=1}^{q} m_j$.

An operation $v \in \mathscr{O}$ has to be executed on the machines of the type $\mu(v)$, i.e. on one of the machines from the set (nest) $\mathscr{M}^{\mu(v)}$ in the time p_{vj} where $j \in \mathscr{M}^{\mu(v)}$.

Let

$$\mathscr{O}^k = \{v \in \mathscr{O} : \mu(v) = k\}$$

be a set of operations executed in the kth nest ($k = 1, 2, \ldots, q$). A sequence of operations sets

$$\mathscr{Q} = [\mathscr{Q}^1, \mathscr{Q}^2, \ldots, \mathscr{Q}^m],$$

such that for each $k = 1, 2, \ldots, q$

$$\mathscr{O}^k = \bigcup_{i=t_{k-1}+1}^{t_{k-1}+m_k} \mathscr{Q}^i \quad \text{and} \quad \mathscr{Q}^i \cap \mathscr{Q}^j = \emptyset, \quad i \neq j, \ i, \ j = 1, 2, \ldots, m,$$

we call an *assignment of operations from the set $\mathscr{O}$ to machines from the set $\mathscr{M}$* (or shortly, operation-machine assignment).

A sequence $[\mathscr{Q}^{t_{k-1}+1}, \mathscr{Q}^{t_{k-1}+2}, \ldots, \mathscr{Q}^{t_{k-1}+m_k}]$ is an *assignment* of operations to machines in the ith nest (shortly, assignment in the ith nest). In a special case a machine can execute no operations and then a set of operations assigned to execute by this machine is an empty set.

If the assignment of operations to machines has been completed, then the optimal schedule of operations execution determination (including a sequence of operations execution on machines) leads to the classic scheduling problem solving, i.e. the job shop problem (Grabowski & Wodecki (2005)).

Let $K = [K_1, K_2, \ldots, K_m]$ be a sequence of sets where $K_i \in 2^{\mathscr{O}^i}$, $i = 1, 2, \ldots, m$ (in particular case elements of this sequence can constitute empty sets). By $\mathscr{K}$ we denote the set of all such sequences. The number of elements of the set $\mathscr{K}$ is $2^{|\mathscr{O}^1|} \cdot 2^{|\mathscr{O}^2|} \cdot \ldots \cdot 2^{|\mathscr{O}^m|}$.

If $\mathscr{Q}$ is an assignment of operations to machines then $\mathscr{Q} \in \mathscr{K}$ (of course, the set $\mathscr{K}$ includes also sequences which are not feasible; that is such sequences do not constitute assignments of operations to machines).

For any sequence of sets $K = [K_1, K_2, \ldots, K_m]$ $(K \in \mathcal{K})$ by $\Pi_i(K)$ we denote the set of all permutations of elements from K_i. Thereafter, let

$$\pi(K) = (\pi_1(K), \pi_2(K), \ldots, \pi_m(K))$$

be a concatenation m sequences (permutations), where $\pi_i(K) \in \Pi_i(K)$. Therefore

$$\pi(K) \in \Pi(K) = \Pi_1(K) \times \Pi_2(K) \times, \ldots, \Pi_m(K).$$

It is easy to observe that if $K = [K_1, K_2, \ldots, K_m]$ is an assignment of operations to machines then the set $\pi_i(K)$ $(i = 1,2,\ldots,m)$ includes all permutations (possible sequences of execution) of operations from the set K_i on the machine i. Further, let

$$\Phi = \{(K, \pi(K)) : K \in \mathcal{K} \wedge \pi(K) \in \Pi(K)\}$$

be a set of pairs, where the first element is a sequence set and the second – a concatenation of permutations of elements of these sets. Any feasible solution of the FJSP is a pair $(\mathcal{Q}, \pi(\mathcal{Q})) \in \Phi$ where $\mathcal{Q}$ is an assignment of operations to machines and $\pi(Q)$ is a permutations concatenation determining the operation execution sequence which are assigned to each machine fulfilling constrains (i–iv).

By Φ° we denote a set of feasible solutions for the FJSP. Of course $\Phi^\circ \subset \Phi$.

3.1. Graph representation

Any feasible solution $\Theta = (\mathcal{Q}, \pi(\mathcal{Q})) \in \Phi^\circ$ (where $\mathcal{Q}$ is an assignment of operations to machines and $\pi(\mathcal{Q})$ determines the operation execution sequence on each machine) of the FJSP can be presented as a directed graph with weighted vertexes $G(\Theta) = (\mathcal{V}, \mathcal{R} \cup \mathcal{E}(\Theta))$ where $\mathcal{V}$ is a set of vertexes and a $\mathcal{R} \cup \mathcal{E}(\Theta)$ is a set of arcs, whereas:

(1) $\mathcal{V} = \mathcal{O} \cup \{s, c\}$, where s and c are additional (fictitious) operations which represents 'start' and 'finish', respectively. A vertex $v \in \mathcal{V} \setminus \{s, c\}$ possesses two attributes:

- $\lambda(v)$ – a number of machines on which an operation $v \in \mathcal{O}$ has to be executed,
- $p_{v,\lambda(v)}$ – a weight of the vertex which equals to the time of operation $v \in \mathcal{O}$ execution on the assigned machine $\lambda(v)$.

Weights of additional vertexes ($p_s = p_c = 0$).

(2) $\mathcal{R} = \bigcup_{j=1}^{n} [\bigcup_{i=1}^{o_j - 1} \{(l_{j-1} + i, l_{j-1} + i + 1)\} \cup \{(s, l_{j-1} + 1)\} \cup \{(l_{j-1} + o_j, c)\}]$.

A set $\mathcal{R}$ includes arcs which connect successive operations of the job, arcs from the vertex s to the first operation of each job and arcs from the last operation of each job to the vertex c.

(3) $\mathcal{E}(\Theta) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{|\mathcal{O}^k| - 1} \{(\pi_k(i), \pi_k(i+1))\}$.

It is easy to notice, that arcs from the set $\mathcal{E}(\Theta)$ connect operations executed on the same machine (π_k is a permutation of operations executed on the machine M_k, that is operations from the set $\mathcal{O}^k$).

Arcs from the set $\mathcal{R}$ determine the operation execution sequence inside jobs (technological order) and arcs from the set $\mathcal{E}(\pi)$ the operation execution sequence on each machine.

Remark 1. A pair $\Theta = (\mathcal{Q}, \pi(\mathcal{Q})) \in \Phi$ is a feasible solution for the FJSP if and only if $G(\Theta)$ does not include cycles.

A sequence of vertexes $(v_1, v_2, \ldots, v_k)$ of the graph $G(\Theta)$ such that $(v_i, v_{i+1}) \in \mathcal{R} \cup \mathcal{E}(\Theta)$ for $i = 1,2,\ldots,k-1$, we call a *path* from the vertex v_1 to v_k. By $C(v,u)$ we denote the longest path (called a *critical path*) in the graph $G(\Theta)$ from the vertex v to u $(v, u \in \mathcal{V})$ and by $L(v,u)$ we denote a *length* (sum of vertexes weights) of this path.

It is easy to notice that the time of all operations execution $C_{\max}(\Theta)$ related with the assignment of operations $\mathcal{Q}$ and schedule $\pi(\mathcal{Q})$ equals the length $L(s,c)$ of the critical path $C(s,c)$ in the graph $G(\Theta)$. A solution of the FJSP amounts to determine a feasible solution $\Theta = (\mathcal{Q}, \pi(\mathcal{Q})) \in \Phi^\circ$ for which the graph connected with this solution $G(\Theta)$ has the shortest critical path, that is it minimizes $L(s,c)$.

Let $C(s,c) = (s, v_1, v_2, \ldots, v_w, c)$, $v_i \in \mathcal{O}$ $(1 \leqslant i \leqslant w)$ be a critical path in the graph $G(\Theta)$ from the starting vertex s to the final vertex c. This path can be divided into subsequences of vertexes

$$\mathcal{B} = [B^1, B^2, \ldots, B^r],$$

called *blocks* in the permutations on the critical path $C(s,c)$ (Grabowski (1979), Grabowski & Wodecki (2005)) where

(a) a block is a subsequence of vertexes from the critical path including successive operations executed directly one after another,
(b) a block includes operations executed on the same machine,
(c) a product of any two blocks is an empty set,
(d) a block is a maximal (due to the inclusion) subset of operations from the critical path fulfilling constrains (a)–(c).

In the further part only these blocks for which $|B^k| > 1$ will be considered, i.e. non-empty blocks.

If $B^k (k = 1,2,\ldots,r)$ is a block on the machine M_i $(i = 1,2,\ldots,m)$ from the nest $t (t = 1,2,\ldots,q)$ then we will denote it as follows:

$$B^k = (\pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k - 1), \pi_i(b^k)),$$

where $1 \leqslant a^k < b^k \leqslant |Q^i|$. Operations $\pi(a^k)$ and $\pi(b^k)$ in the block B^k are called *the first* and *the last*, respectively. In turn a block without the first and the last operation we call an *internal block*. The definitions given are presented on the Fig. 2.

In the work of Grabowski (1979) there are theorems called *elimination criteria* of blocks in the job shop problem.

Theorem 1. (Grabowski (1979)) *Let $\mathcal{B} = [B^1, B^2, \ldots, B^r]$ be a sequence of blocks of the critical path in the acyclic graph $G(\Theta)$, $\Theta \in \Phi^\circ$. If the graph $G(\Omega)$ is feasible (i.e. it represents a feasible solution) and if it was generated from $G(\Theta)$ by the change of the order of operations execution on some machine and $C_{max}(\Omega) < C_{max}(\Theta)$ then in the $G(\Omega)$*

(i) at least one operation from a block B^k, $k \in \{1,2,\ldots,r\}$ precedes the first element $\pi(a^k)$ of this block, or
(ii) at least one operation from a block B^k, $k \in \{1,2,\ldots,r\}$ occurs after the last element $\pi(b^k)$ of this block.

The change of operations order in any block does not generate the solution with less value of the cost function. At least one operation from any block should be moved before the first or after the last operation of this block to generate the solution (graph) with smaller weight of the critical path. We use this property to reduce the neighborhood size, i.e. do not generate solutions with greater values (comparing with the current solution) of the cost function.

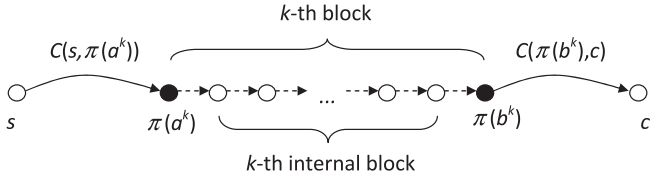

Fig. 2. Blocks on the critical path.

3.2. Solution method

There is an exponential number of possible jobs to machines assignments, due to the number of operations. Each feasible assignment generates a NP-hard problem (job shop) whose solution consists in determining an optimal jobs execution order on machines. One has to solve exponential number of NP-hard problems to solve the flexible job shop problem. Therefore, we will apply approximate algorithms consisting in execution of these following two steps:

Step 1: Jobs to machines assignment determination;
Step 2: Solving a job shop problem for the assignment determined in the Step 1.

We will apply the tabu search algorithm in the Step 1. The neighborhood of the current solution (assignment) is generated by jobs moving between machines of the same types. The best element of this neighborhood generates a job shop problem, which is solved in the Step 2. For comparison, we will also present an algorithm which uses population based method in the Step 1., without jobs moves between machines of the same type.

3.3. Jobs to machines assignment problem

The problem of a 'good' (suboptimal) operations to machines determination is considered in this section. Each operation is assigned exactly to one nest. It is necessary to make a partition of assigned operations to machines in each nest. The method of partitioning has an influence on execution finish time of all jobs, that is the value of the job shop problem solution. Generally, we are looking for such a partition whose the cost function value (of the adequate job shop problem) is minimal.

Let $\Theta = (\mathscr{Q}, \pi(\mathscr{Q})) \in \Phi^{\circ}$ is a feasible solution of the FJSP where $\mathscr{Q} = [\mathscr{Q}^1, \mathscr{Q}^2, \ldots, \mathscr{Q}^m]$ is operation-machine assignment, ϱ_i is a number of operation executed on the machine M_i (i.e. $\varrho_i = |\mathscr{Q}^i|$) and

$$\pi(\mathscr{Q}) = (\pi_1(\mathscr{Q}), \pi_2(\mathscr{Q}), \ldots, \pi_m(\mathscr{Q}))$$

is a concatenation of m permutations. A permutation $\pi_i(\mathscr{Q})$ determines a sequence of operations from the set $\mathscr{Q}^i$ which has to be executed on the machine M_i ($i = 1,2,\ldots,m$).

In the further part of this section, in case in which it does not evoke ambiguity, we omit operations assignment $\mathscr{Q}$ which appears as a permutation parameter. Therefore, the concatenation $\pi(\mathscr{Q}) = (\pi_1(\mathscr{Q}), \pi_2(\mathscr{Q}), \ldots, \pi_m(\mathscr{Q}))$ will be presented as $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$.

By $t^i_j(k,l)$ we denote a *transfer* type move (in brief *t-move*) which consists in moving an operation from the position k in the permutation π_i (i.e. the operation $\pi_i(k)$) to the position l in the permutation π_j (moving operations from positions $l, l+1, \ldots$ one position to the right). The execution of the move $t^i_j(k,l)$ generates from $\Theta = (\mathscr{Q}, \pi) \in \Phi^{\circ}$ a new solution $\Theta' = (\mathscr{Q}', \pi')$ such that

$$\pi'_v = \pi_v, \quad v \neq i,j, \quad v = 1,2,\ldots,m \tag{1}$$

and

$$\pi'_i = (\pi_i(1), \pi_i(2), \ldots, \pi_i(\mathbf{k}-\mathbf{1}), \pi_i(\mathbf{k}+\mathbf{1}), \ldots, \pi_i(\varrho_i - 1)), \tag{2}$$

$$\pi'_j = (\pi_j(1), \pi_j(2), \ldots, \pi_j(\mathbf{l}-\mathbf{1}), \underline{\pi_i(\mathbf{k})}, \pi_j(\mathbf{l}), \ldots, \pi_j(\varrho_j + 1)). \tag{3}$$

The execution of this move causes a movement of the operation $\pi_i(k)$ from the set $\mathscr{Q}^i$ (i.e. from the machine M_i) to the set $\mathscr{Q}^j$ (i.e. to the machine M_j). Therefore,

$$\mathscr{Q}'^v = \mathscr{Q}^v, \quad v \neq i,j, \quad v = 1,2,\ldots,m \tag{4}$$

and

$$\mathscr{Q}'^i = \mathscr{Q}^i \setminus \{\pi_i(k)\}, \quad \mathscr{Q}'^j = \mathscr{Q}^j \cup \{\pi_i(k)\}. \tag{5}$$

A graph $G(\Theta')$ generated by a *t-move* execution can have a cycle and then the solution $\Theta' = (\mathscr{Q}', \pi')$ is not feasible.

Remark 2. An upper bound of the number of *t-moves* is $O(qm^2o^2)$.

The execution of *t-move* causes an operation transferring from a machine into another, i.e. a new operation-machine assignment in a nest. Therefore, from any solution (operation-machine assignment), executing *t-moves*, it is possible to obtain any other operation-machine assignment, i.e. operations sets partition in particular sets.

If τ is a *t-move*, by $\tau(\Theta)$ we denote a solution generated from Θ by executing a move τ (see (1)–(5)).

For a fixed feasible solution Θ, let $\mathscr{T}(\Theta)$ be a set of all *t-moves*. A *neighborhood* Θ is a set

$$\mathscr{N}(\Theta) = \{\tau(\Theta) \in \Phi^{\circ} : \tau \in \mathscr{T}\}, \tag{6}$$

where Φ° is a set of feasible solutions. Feasibility $\tau(\Theta)$ corresponds to the acyclicity of the graph $G(\tau(\Theta))$.

It was mentioned at the beginning of this chapter that the solution of the FJSP consists of two steps. The first – determination of operation-machine assignment and the second – operation execution order determination, i.e. a job shop problem solving.

Let $\Theta = (\mathscr{Q}, \pi)$ be a feasible solution of the FJSP. The new operation-machine assignment $\mathscr{Q}'$ will be generated from the assignment $\mathscr{Q}$ as follows:

- determine a neighborhood $\mathscr{N}(\Theta)$,
- select from the neighborhood a solution $\Theta' = (\mathscr{Q}', \pi')$ with the lowest goal function value – the new operation-machine assignment $\mathscr{Q}'$.

The number of all *t-moves* can be huge, so we will omit some of them and we will considered only these which can provide us with a goal function value improvement. Moreover, we will not determine an exact goal function value of solutions generated by *t-moves*, but approximate them only. As computational experiments proved it causes significant algorithm work acceleration with little results aggravation. In the further part we will describe precisely methods of elimination from the neighborhood (6) superfluous moves.

3.4. Neighborhood determination

Execution a *t-move* can conduct to generate a non-feasible solution, i.e. a graph connected with this solution can have a cycle. Therefore, checking feasibility equals checking if a graph has a cycle. The adequate algorithm has a computational complexity $O(o)$ where o is a number of all operations. In the further part we will prove theorems which make it possible checking feasibility of solutions (i.e. acyclicity of adequate graphs) generated by *t-moves* in the constant time.

Let $\Theta = (\mathscr{Q}, \pi)$ be a feasible solution where $\mathscr{Q} = [\mathscr{Q}^1, \mathscr{Q}^2, \ldots, \mathscr{Q}^m]$ is an operation-machine assignment and $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ is a concatenation of m permutations. A permutation π_i determines an execution order of operations from the set $\mathscr{Q}^i$ on the machine M_i ($i = 1,2,\ldots,m$).

We consider two machines M_i and M_j from the same nest. A permutation π_i determines an execution order of operations from the set $\mathscr{Q}^i$ on the machine M_i and π_j – an execution order of operations from the set $\mathscr{Q}^j$ on the machine M_j. For an operation $\pi_i(k) \in \mathscr{Q}^i$ we define two parameters connected with paths in the graph $G(\Theta)$.

The first

$$\eta_j(k)=\begin{cases}1, & \text{if } \forall\, v=1,2,\ldots,\varrho_j\\ & \text{there does not exist a path } C(\pi_j(v),\pi_i(k)),\\ 1+\max\limits_{1\leqslant v\leqslant\varrho_j}\{\text{there exists a path } C(\pi_j(v),\pi_i(k))\}, & \\ & \text{in other case}\end{cases} \tag{7}$$

Therefore, none of operations placed in the permutation π_j on positions $\eta_j(k),\eta_j(k)+1,\ldots,\varrho_j$ (where $\varrho_j=|\mathscr{Q}^j|$) there does not exist a path to the operation (vertex) $\pi_i(k)$ in the graph $G(\Theta)$. This situation is showed on the Fig. 3.

The second parameter

$$\rho_j(k)=\begin{cases}1+\varrho_j, & \text{if } \forall\, v=1,2,\ldots,\varrho_j\\ & \text{there does not exist a path } C(\pi_j(v),\pi_i(k)),\\ 1+\min\limits_{\eta_j(k)\leqslant v\leqslant\varrho_j}\{\text{there exists a path } C(\pi_i(k),\pi_j(v))\}, & \\ & \text{in other case}\end{cases} \tag{8}$$

From the definition formulated above it follows that in the graph $G(\Theta)$ there is not a path from a vertex $\pi_i(k)$ to any operation placed in positions $\eta_j(k),\eta_j(k)+1,\ldots,\rho_j(k)$ in the permutation π_j (see Fig. 3).

Now we will prove two theorems characterizing a *t-move* whose execution generates an unfeasible solution. These theorems constitute the construction base of the very efficient neighborhoods. The assumptions structure allow to easy implementation in the parallel computations environment, such as GPUs.

Theorem 2. *Let $\Theta=(\mathscr{Q},\pi)$ be a feasible solution for the* FJSP *and let π_i, π_j be operations permutations executed on machines M_i, M_j. If machines M_i, M_j belong to the same nest then executing a t-move $t_j^i(k,l)$ $(\pi_i(k)\in\mathscr{Q}^i,\ l=1,2,\ldots,\eta_j(k)-1)$ generates a solution which is not feasible.*

Proof. Let $\Theta=(\mathscr{Q},\pi)$ be a feasible solution and let $G(\Theta)$ be an adequate graph connected with it. The permutation π_i determines operations executing order on the machines M_i and π_j – an order on the machine M_j. We consider a *t-move* $t_j^i(k,l)$ consisting in an operation $\pi_i(k)$ transfer from a machine M_i to a position l $(1\leqslant l\leqslant\eta_j(k)-1)$ in the permutation π_j, i.e. to the machine M_j. This move generates a new solution $\Theta'=(\mathscr{Q}',\pi')$ (see (1)–(5) and an adequate graph $G(\Theta')$). Now we will prove that this graph includes a cycle.

From the definition (7) of the parameter $\eta_j(k)$ it follows that in the graph $G(\Theta)$ there exists a path from the vertex $\pi_j(\eta_j(k)-1)$ to $\pi_i(k)$, i.e. the path $C(\pi_j(\eta_j(k)-1),\pi_i(k))$. It is showed on the Fig. 4.

Moreover, there is also a path $C(\pi_j(l))=(\pi_i(k),\pi_j(\eta_j(k)-1))$ in this graph. Executing a move $t_j^i(k,l)$ causes an insertion of the operation $\pi_i(k)$ in the position l to the permutation π_j. It results in an insertion to the graph $G(\Theta')$, among others, an arc $(\pi_i(k),\pi_j(l))$. It creates a cycle $((\pi_i(k),\pi_j(l)),C(\pi_j(l),\pi_j(\eta_j(k)-1)),C(\pi_j(\eta_j(k)-1),\pi_i(k)))$, which finishes the proof of the theorem. □

Similar theorem can be proved for the $\rho_j(k)$ parameter.

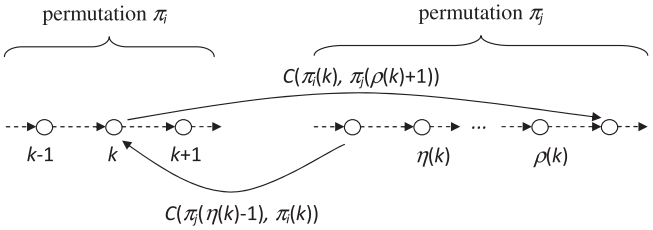

Fig. 3. Parameters $\eta_j(k)$ and $\rho_j(k)$ visualization for an operation $\pi_i(k)$.

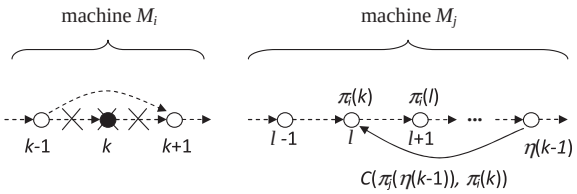

Fig. 4. Directed graph $G(\Theta')=G(t_j^i(k,l)(\Theta))=G(\mathscr{Q}',\pi')$.

Theorem 3. *Let $\Theta=(\mathscr{Q},\pi)$ be a feasible solution for the* FJSP *and π_i, π_j be operations permutations executing on machines M_i, M_j. If machines M_i, M_j belong to the same nest then executing t-move $t_j^i(k,l)$ $(\pi_j(k)\in\mathscr{Q}^i,\ l=\rho_j(k)+1,\rho_j(k)+2,\ldots,\varrho_j)$ generates a solution which is not feasible.*

Proof. Similarly as in the proof of the Theorem 2 one can show that after executing a move $t_j^i(k,l)$ $(l=\rho_j(k)+1,\rho_j(k)+2,\ldots,\varrho_j)$ there is a cycle $(C(\pi_i(k),\pi_j(\rho_j(k)-l)),C(\pi_j(\rho_j(k)-l),\pi_j(l-1)),(\pi_j(l-1),\pi_i(k)))$ in the generated graph $G(\Theta')$. □

Let us denote a set of these *t-moves* z $\mathscr{T}(\Theta)$ which fulfill assumptions of Theorems 2 or 3 by $\mathscr{T}^{noacc}$. Therefore, the moves generates non-feasible solutions from Θ.

Theorem 4. *Let $\Theta=(\mathscr{Q},\pi)$ be a feasible solution of the* FJSP *and π_i, π_j be operations permutations executed on machines M_i, M_j. If machines M_i, M_j belong to the same nest then executing a t-move $t_j^i(k,l)$ $(l=\eta_j(k),\eta_j(k)+1,\ldots,\rho_j(k))$ generates a feasible solution.*

Proof. The proof of this theorem follows directly from the definition of parameters $\eta_j(k)$, $\rho_j(k)(\pi_i(k)\in\mathscr{Q}^i)$ and Theorems 2 and 3. □

Property 1. *For each operation $\pi_i(k)$ executed on the machine M_i there exists a position l in the permutation π_j (i.e. on the machine M_j from the same nest) such that executing a move $t_j^i(k,l)$ generates a feasible solution from a solution Θ.*

Proof. It is enough to observe that if Θ is a feasible solution therefore for an operation $\pi_i(k)$ there is $\rho_j(k)\geqslant\eta_j(k)$. □

Now we will prove a theorem which constitutes a base of an elimination of some *t-moves* during a process of neighborhood generation. Its function is similar to the function of the Theorem 1.

Let us assume that for an assignment $\mathscr{Q}$ a concatenation π is an optimal operations schedule on machines, $G(\Theta)$ is a graph connected with a solution $\Theta=(\mathscr{Q},\pi)$ and $C_{\max}(\Theta)$ is a cost function value, i.e. a critical path length in the graph $G(\Theta)$.

Theorem 5. *Let $\Theta=(\mathscr{Q},\pi)$ be a feasible solution for the* FJSP *and let $\mathscr{B}=[B^1,B^2,\ldots,B^r]$ be a sequence of critical path blocks in the graph $G(\Theta)$. If $\Theta'=(\mathscr{Q}',\pi')$ is a feasible solution which was generated from Θ by operation-machine assignment changing in a nest and $C_{max}(\Theta')<C_{max}(\Theta)$ therefore in the Θ' at least one operation from some block was moved to other machine (in the same nest).*

Proof. Let $\mathscr{B}=[B^1,B^2,\ldots,B^r]$ be a critical path blocks sequence in the graph $G(\Theta)$. Each block is an operation sequence

$$B^i=(\pi(a^i),\pi(a^i+1),\ldots,\pi(b^i)),$$

for $i=1,2,\ldots,r$ where $1\leqslant a^1\leqslant b^1<a^2\leqslant b^2<\cdots<a^k\leqslant b^k$.

For a notion simplification we assume (in the proof of this theorem) that each operation from a critical path belongs to some block. Therefore a block can consist of the single operation.

By

$$\mathscr{Y}^i(\pi) = \{\pi(a^i), \pi(a^i+1), \ldots, \pi(b^i)\},$$

we denote a jobs set from a block B^i.

The critical path $C(s,c)$ in the graph $G(\Theta)$ includes all vertexes (operations) of a set $\bigcup_{i=1}^{r}\mathscr{Y}^i$ and its length $L(s,c) = C_{\max}(\Theta) = \sum_{i=1}^{r}\sum_{v\in\mathscr{Y}^i} p_v$.

Let $\Theta' = (\mathscr{Q}', \pi')$ be a feasible solution such that $C_{\max}(\Theta') < C_{\max}(\Theta)$. Let us assume that no operations from no block $B^1, B^2, \ldots, B^r$ in the operation-machine assignment $\mathscr{Q}'$ is not to be moved to another machine from the same nest. Thus

$$\mathscr{Y}^i(\pi) = \mathscr{Y}^i(\pi'), \quad i = 1,2,\ldots,r.$$

Therefore, jobs sequence $(\pi(a^i),\pi(a^i+1),\ldots,\pi(b^i))$ in the permutation π and $(\pi'(a^i),\pi'(a^i+1),\ldots,\pi'(b^i))$ in π' are permutations of the same jobs subsequence $\mathscr{Y}^i = \{\pi(a^i), \pi(a^i+1), \ldots, \pi(b^i)\}$. We consider a path $C'(s,c)$ in the graph $G(\Theta')$. Vertexes of this path belong to a set $\mathscr{A} = \bigcup_i^r \mathscr{Y}^i(\pi')$. The length of this path $L'(s,c) = \sum_{v\in\mathscr{A}} p_v$ so it equals the length $L(s,c) = C_{\max}(\Theta)$ of the critical path $C(s,c)$ in the graph $G(\Theta)$. Therefore $C_{\max}(\Theta') \geqslant C_{\max}(\Theta)$, which contradicts the assumption. □

Let Θ be a feasible solution, $\mathscr{B}$ – a critical path blocks sequence in the graph $G(\Theta)$ and $\mathscr{T}$ – a set of *t-moves* defined for the Θ.

We denote by $\mathscr{T}^{out}(\Theta)$ a set of these moves from $\mathscr{T}(\Theta)$ which moves operations which do not belong to any block (i.e. apart from the block) to another machine.

Directly from the Theorem 5 it follows a property which constitutes a base to elimination of superfluous moves.

Property 2. *If a feasible solution Θ' was generated from Θ by execution a t-move belonging to the set $\mathscr{T}^{out}(\Theta)$ then*

$$C_{\max}(\Theta') \geqslant C_{\max}(\Theta).$$

Proof. The proof results from the Theorem 5 directly. □

Therefore, executing a *t-move* consisted in moving an operations which does not lie on the critical path to another machine which does not generate a solution with lower cost function value.

Theorem 6. *Let $\Theta = (\mathscr{Q}, \pi)$ be a feasible solution for the* FJSP. *If B^u is a block on the machine M_i and $B^v$is a block on M_j and both machines belong to the same nest then a transfer type move consisting in moving an operation from an internal block B^u to the internal block B^vdoes not generate a solution with lower cost function value.*

Proof. Let $\Theta = (\mathscr{Q}, \pi)$ be a feasible solution, $G(\Theta)$ – a graph connected with it and $\mathscr{B} = [B^1, B^2, \ldots, B^r]$ – block from the critical path sequence. We assume that

$$B^u = (\pi(a^u), \pi(a^u+1), \ldots, \pi(b^u)),$$
$$B^v = (\pi(a^v), \pi(a^v+1), \ldots, \pi(b^v)),$$

are blocks $(1 \leqslant u < v \leqslant r)$ on machine M_i and M_j, respectively. We consider a *t-move* $t_j^i(k,l)$ where $a^u < k < b^u$ and $a^v < l < b^v$ moving the operation from the block B^u to the block B^v. This move generates from Θ a new solution $\Theta' = (\mathscr{Q}', \pi')$. The operation-machine assignment $\mathscr{Q}'$ and the permutation π' are defined in (1)–(5). The critical path $C(s,c)$ in the graph $G(\Theta)$ can be partitioned as follows:

$$C(s,c) = (C(s,\pi(a^u)), C(\pi(a^u),\pi(b^u)), C(\pi(b^u),\pi(a^v)), C(\pi(a^v),$$
$$\pi(b^v)), C(\pi(b^v),c)).$$

This path is showed on the Fig. 5

There is a path

$$C'(s,c) = (C'(s,\pi'(a^u)), C'(\pi'(a^u),\pi'(b^u)),$$
$$C'(\pi'(b^u),\pi'(a^v)), C'(\pi'(a^v),\pi'(b^v)), C'(\pi'(b^v),c))$$

in the graph $G(\Theta')$. It is easy to see that the following paths are equal:

$$C'(s,\pi'(a^u)) = C(s,\pi(a^u)), \quad C'(\pi'(b^u),\pi'(a^v)) = C(\pi(b^u),\pi(a^v))$$

and

$$C'(\pi'(b^v),s) = C(\pi(b^v),s),$$

so they lengths are also equal.

We consider paths $C(\pi(a^u),\pi(b^u))$ and $C(\pi(a^v),\pi(b^v))$ in the graph $G(\Theta)$, and $C'(\pi(a^u),\pi(b^u))$ and $C'(\pi(a^v),\pi(b^v))$ in the graph $G(\Theta')$. Because the path $C'(\pi'(a^u),\pi'(b^u))$ includes all vertexes of the path $C(\pi(a^u),\pi(b^u))$ excluding the vertex $\pi(k)$ which was moved by the *t-move* from the block B^u, therefore the path length

$$L'(\pi'(a^u),\pi'(b^u)) = L(\pi(a^u),\pi(b^u)) - p_{\pi(k)}.$$

Similarly, considering paths $C(\pi(a^v),\pi(b^v))$ i $C'(\pi(a^v),\pi'(b^v))$ one can show that

$$L'(\pi'(a^v),\pi'(b^v)) = L(\pi(a^v),\pi(b^v)) + p_{\pi(k)}.$$

From this it follows that $L'(c,s) = L(c,s)$.

Summing up, the length of some path $C'(c,s)$ in the graph $G(\Theta')$ equals $C(s,t) = C_{\max}(\Theta)$. In accordance with it $C_{\max}(\Theta') \geqslant C_{\max}(\Theta)$ finishing the proof of the theorem. □

Therefore, to generate a better solution by executing a *t-move* one should move the first of the last operation of the block before the first or after the last operation of another block.

Theorems 2 and 3 proved in this section concern feasibility of solutions generated by *t-moves*. If paths between any pair of vertexes in the graph are known, then this check is executed in the constant time. Moreover, Theorems 5 and 6 defined so-called blocks elimination criteria. That is why they make it possible to omit in the generation procedure these moves which do not generate better solutions than the current ones. From the set of moves $\mathscr{T}(\Theta)$ generating the neighborhood of the solution Θ we omit all the moves which fulfill assumptions of Theorems 2, 3, 5 and 6. Therefore, *t-moves* from the

$$\mathscr{T}^{acc}(\Theta) = \mathscr{T}(\Theta) \setminus (\mathscr{T}^{noacc} \cup \mathscr{T}^{out})$$

will be used to the neighborhood Θ creation, where $\mathscr{T}^{noacc}$ is the set of moves generating non-feasible solutions (Theorems 2 and 3) and $\mathscr{T}^{out}$ is the set of moves generating solutions with cost function value lower than $C_{\max}(\Theta)$ (Theorems 5 and 6).

If $\mathscr{B} = [B^1, B^2, \ldots, B^r]$ is a block from the critical path sequence in the graph $G(\Theta)$ then the set $\mathscr{T}^{acc}(\Theta)$ includes moves which transfer the first (or the last) operation of each block from the machine M_i

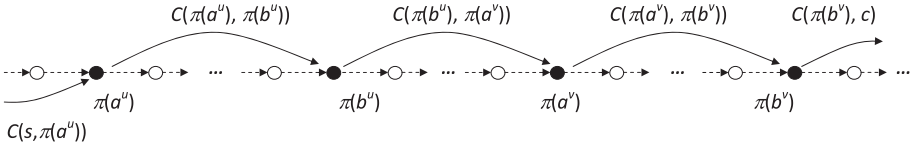

Fig. 5. Critical path in the graph $G(\mathscr{Q}, \pi)$.

to another machine (from the same nest). If $\pi(v)$ is the first (or the last) operation of a block and M_j is the machine from the same nest then the set $\mathscr{T}^{acc}(\Theta)$ includes moves which transfer $\pi(v)$ to the following positions: $\eta_j(v),\eta_j(v+1),\ldots,\rho_j(v)$. Such a neighborhood has a big size, so we have limited moves which transfer the first (or the last) operation $\pi(v)$ of a block *only* in the position $\eta_j(v)$ or $\rho_j(v)$. So, ultimately

$$\mathscr{T}^{subm}(\Theta) = \{t_j^i(v,w) \in \mathscr{T}^{acc} : v \in \{a^k, b^k\},$$

$$w \in \{\eta_j(v), \rho_j(v)\},\ k = 1,2,\ldots,r\}. \quad (9)$$

The neighborhood Θ constitutes the feasible solutions set

$$\mathscr{N}(\Theta) = \{\tau(\Theta) : \tau \in \mathscr{T}^{subm}(\Theta)\}. \quad (10)$$

This neighborhood has the size $O(r \cdot m)$, where r is a number of the critical path blocks.

4. Proposed FJSP solving algorithms

Flexible job shop problem solving algorithms proposed here include two major modules: the machine selection module and the operation scheduling module.

Machine selection module. This module is based on the tabu search (1st approach) and the population-based metaheuristics (2nd approach) methods and it works sequentially. It helps an operation to select one of the parallel machine from the set of machine types to process it.

Operation scheduling module. This module is used to schedule the sequence and the timing of all operations assigned to each machine from the center. It has to solve classic job shop problems after having assigned operations to machines. Two approaches: constructive INSA (Nowicki & Smutnicki, 1996) and TSAB (Nowicki & Smutnicki, 1996) (tabu search) were used on this level.

On each level a metaheuristic algorithm is used, so we call this method meta^{2}heuristics (*meta-square-heuristics*).

4.1. Parallel tabu search based meta^{2}heuristics

The tabu search method was used here as a machine selection module. The algorithm operates on solutions which constitute jobs to machines assignments. The general idea of the tabu search method applied for scheduling problems can be found in (Grabowski & Wodecki, 2005 & Nowicki & Smutnicki, 1996). The tabu list T stores couples (v,k) where v is the position in the assignment vector and k is the machine to which v is assigned before the move. The first assignment $\mathscr{Q}^0$ is generated by the search for the global minimum in the processing time table taken from (Pezzella et al., 2008). The skeleton of the double-layer algorithm including both machine selection module and operation scheduling module is presented below.

Algorithm 1. Parallel Tabu Search Based Meta^{2}heuristics (TSBM^{2}h)

$\Theta^* = (\mathscr{Q}^*, \pi(\mathscr{Q}^*))$ – the best known solution, where
$\mathscr{Q}^*$ – the best known assignment and
$\pi(\mathscr{Q}^*)$ – the operation sequence corresponding to $\mathscr{Q}^*$;
Step 0: Find the starting solution $\Theta^0 = (\mathscr{Q}^0, \pi(\mathscr{Q}^0))$; $\Theta^* \leftarrow \Theta^0$;
$\Theta = (\mathscr{Q}, \pi(\mathscr{Q}))$ – current solution; $\Theta \leftarrow \Theta^0$;
Step 1: Generate the neighborhood $\mathscr{N}(\mathscr{Q})$ of the current assignment $\mathscr{Q}$.
Exclude from $\mathscr{N}(\mathscr{Q})$ elements from tabu list T;
Step 2: Divide $\mathscr{N}(\mathscr{Q})$ into $k = \lceil \frac{|\mathscr{N}(\mathscr{Q})|}{p} \rceil$ groups;
Each group consists of at most p elements;
Step 3: For each group k find (using p processors)
an operation sequence $\pi(\mathscr{X})$
corresponding to the assignment $\mathscr{X} \in \mathscr{N}(\mathscr{Q})$ and value of the makespan $C_{max}(\mathscr{X}, \pi(\mathscr{X}))$;
Step 4: Find an assignment $Z \in \mathscr{N}(\mathscr{Q})$ such that
$C_{max}(\mathscr{Z}, \pi(\mathscr{Z})) = min\{C_{max}(\mathscr{X}, \pi(\mathscr{X})) : \mathscr{X} \in \mathscr{N}(\mathscr{Q})\}$;
Step 5: if $C_{max}(\mathscr{Z}, \pi(\mathscr{Z})) < C_{max}(\mathscr{Q}^*, \pi(\mathscr{Q}^*))$ **then**
$\pi(\mathscr{Q}^*) \leftarrow \pi(\mathscr{Z})$;
$\mathscr{Q}^* \leftarrow \mathscr{Z}$;
Include Z to the list T;
$\mathscr{Q} \leftarrow \mathscr{Z}$;
$\pi(\mathscr{Q}) \leftarrow \pi(\mathscr{Z})$;
Step 6: if (*Stop condition* is true) **then** Stop;
else go to Step 1.;

In the second step of the algorithm a neighborhood $\mathscr{N}(\mathscr{Q})$ is divided into disjoint sets

$$\bigcup_{i=1}^{k} \mathscr{N}_i(\mathscr{Q}) = \mathscr{N}(\mathscr{Q}), \quad \bigcap_{i=1}^{k} \mathscr{N}_i(\mathscr{Q}) = \emptyset.$$

For each group k values of the makespan are calculated using p GPU processors. The number of processors used in the third step depends on the neighborhood size. In the Step 3 the value of makespan corresponding to the assignment is calculated by means of INSA or TSAB algorithms. The general scheme of the TSBM^{2}h execution on GPU for the CUDA programming environment is presented on the Fig. 6 as a case of heterogeneous programming model (i.e. with using both CPU and GPUs).

We use following parameters for the main (high-level) tabu search algorithm: 200 iteration (stop criterion), length of the tabu list: 7. For the TSAB algorithm used as low-level metaheuristics of the main TS algorithm the following parameters values have been used: 20,000 iterations; 4000 iterations without the best solution improving causes a backtrack jump, the length of the tabu list: 8, the length of the long-term memory used for backtracking: 5.

4.2. Parallel population-based meta^{2}heuristics

The basic idea of this approach is to start with an initial population (any subset of the solution space) – jobs to machines assignments. Next, for each element of the population, a local optimization algorithm is applied to determine a local minimum. In this way we obtain a set of solutions – local minima. If there is an element which is in the same position in several solutions, then it is fixed in this position in the solution, and other positions and elements of solutions are still free. A new population (a set of assignments) is generated by drawing free elements in free positions (because there are fixed elements in fixed positions). After having determined a set of local minima (for the new population) we can increase the number of fixed elements. To prevent from finishing the algorithm's work after executing some number of iterations (when all positions are fixed and there is nothing left to draw), in each iteration 'the oldest' fixed elements are set as free.

The method mentioned above was proposed in the paper (Bożejko & Wodecki, 2009) for the permutational scheduling problems. Here we have adopted this approach to flexible job shop scheduling problem, where solution is a pair of jobs to machines assignment and sequence of jobs permutations on each machine.

Algorithm 2. Parallel Population-Based Meta^{2}heuristics (PBM^{2}h)

$\Theta^* = (\mathscr{Q}^*, \pi(\mathscr{Q}^*))$ – the best known solution, where
$\mathscr{Q}^*$ – the best known assignment and
$\pi(\mathscr{Q}^*)$ – the operation sequence corresponding to $\mathscr{Q}^*$;
Step 0: Generate an initial population P_0 of assignments;
$i \leftarrow 1$; $P_i \leftarrow P_0$ – the first generation of the population;

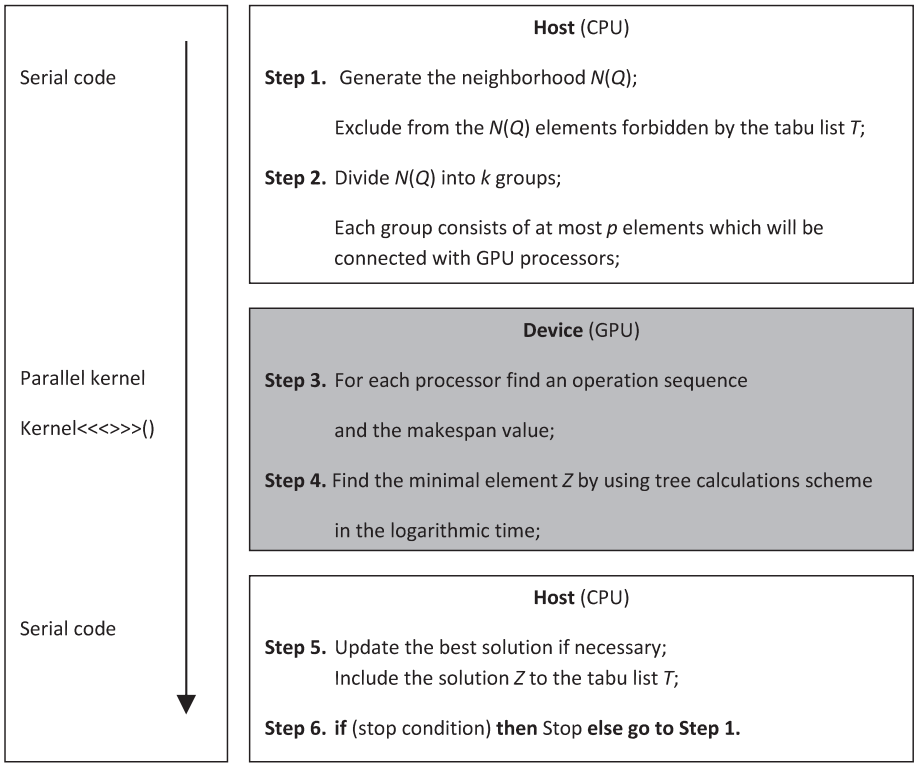

Fig. 6. The general scheme of the TSBM^{2}h execution on the host (CPU) and the computational device (GPU) for the CUDA environment.

Step 1: Divide P_i into $k = \lceil \frac{|P_i|}{p} \rceil$ groups;
Each group consist of at most p elements;
Step 2: For each element $\mathscr{X}$ of each group find (using p processors)
an operation sequence $\pi(\mathscr{X})$
corresponding to the assignment $\mathscr{X} \in P_i$ and value
of the makespan $C_{max}(\mathscr{X}, \pi(\mathscr{X}))$ using tabu search algorithm;
Step 3: Find assignment $\mathscr{Z} \in P_i$ such that
$C_{max}(\mathscr{Z}, \pi(\mathscr{Z})) = min\{C_{max}(\mathscr{X}, \pi(\mathscr{X})) : \mathscr{X} \in P_i\}$;
Step 4: **if** $C_{max}(\mathscr{Z}, \pi(\mathscr{Z})) < C_{max}(\mathscr{Z}^*, \pi(\mathscr{Z}^*))$, **then**
$\pi(\mathscr{Z}^*) = \pi(\mathscr{Z})$;
$\mathscr{Z}^* = \mathscr{Z}$;
Step 5: For each position in population P_i find a number of assignments υ in which the machine m is in the position l;
Step 6: Fix a position in population P_i for which $\frac{\upsilon}{|P_i|} > \alpha$;
Step 7: Insert randomly drawn free elements (machines) in free positions;
$P_{i+1} \leftarrow P_i$;
Step 8: **if** (*Stop condition* is true) **then** Stop;
else go to Step 1.;

In the population-based algorithm the initial population is generated randomly. The size of generated population equals *pop_size* = 100. In first step of algorithm population P_i in the ith generation is divided into disjoint sets

$$\bigcup_{j=1}^{pop\_size} P_i^j = P_i, \qquad \bigcap_{j=1}^{pop\_size} P_i^j = \emptyset.$$

Each element in the randomly generated initial population P_0 starts an assignment for the tabu search algorithm. The tabu search algorithm find a new assignment corresponding to this assignment operation sequence and the value of makespan. The general scheme of the PBM^{2}h execution on GPU for the CUDA programming environment is showed on the Fig. 7.

The parallel population algorithm and parallel tabu search algorithm were coded in C++ language with MPI library and tested on the cluster in the Wrocław Center of Networking and Supercomputing. Algorithms were executed under the OpenPBS batching system.

5. Computational results

Parallel meta^{2}heuristics (TSBM^{2}h and PBM^{2}h) for the flexible job shop problem were coded in C (CUDA) for GPU, ran on the nVidia Tesla C870 GPU (512 GFLOPS) with 128 streaming processors cores and tested on the benchmark problems from literature. The GPU was installed on the Hewlett–Packard server based on 2 Dual-Core AMD 1 GHz Opteron processors with 1 MB cache memory and 8 GB RAM working under 64-bit Linux Debian 5.0 operating system. We compare our results with results obtained by other authors:

1. the set of 10 problem instances taken from Brandimarte (1993),
2. the set of 21 problem instances taken from Barnes and Chambers (1996),
3. the set of 15 problem instances taken from Hurink et al. (1994).

The first phase of computational experiments was devoted to parallelization efficiency determination by estimating experimental speedup values. The sequential algorithm using one GPU processor was coded with the aim of determining the speedup value of the parallel algorithm. Such an approach is called orthodox speedup (see Alba (2005)) and it compares times of algorithm execution on machines with the same processors (1 versus p processors). Tables 1 and 2 show computational times for the sequential and the parallel algorithm as well as speedup values. The obtained orthodox speedup measure value is visualized on the Fig. 8. Flex. denotes the average number of equivalent machines per operation. As we can notice the highest speedup values

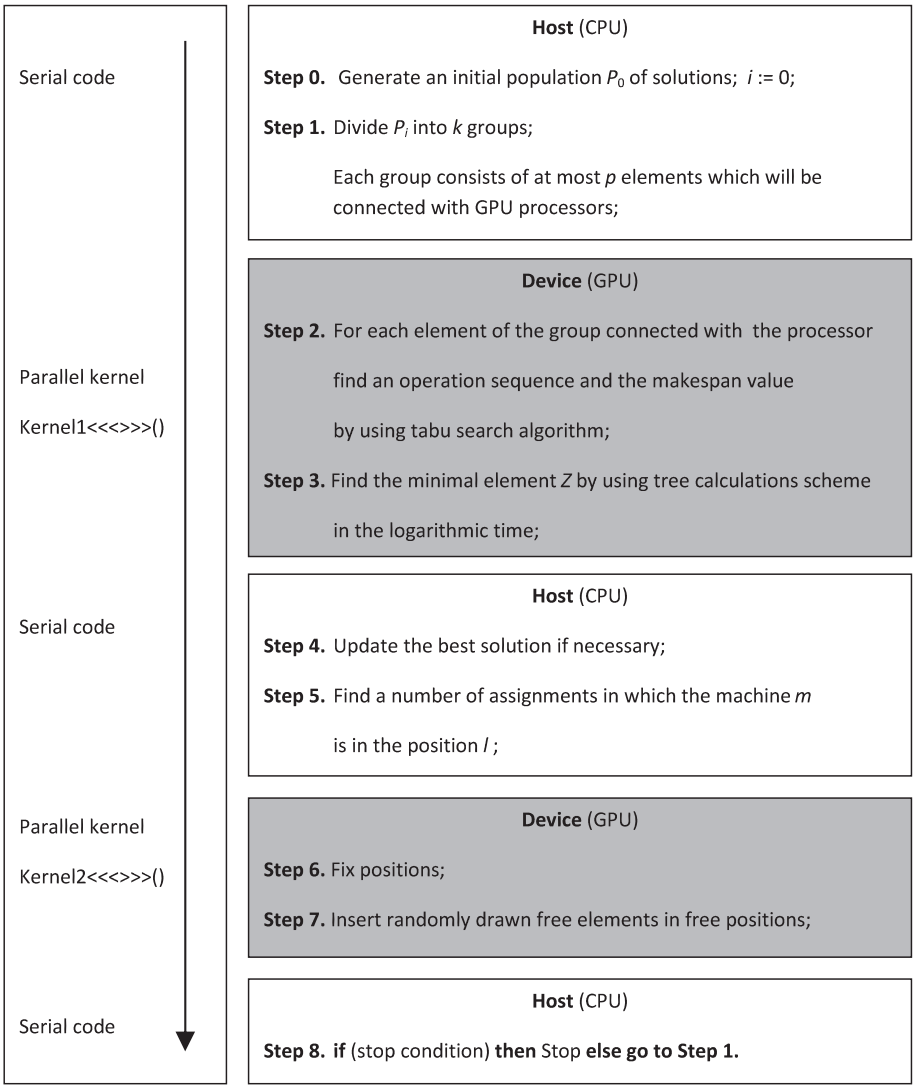

Fig. 7. The general scheme of the PBM^{2}h execution on the host (CPU) and the computational device (GPU) for the CUDA environment.

Table 1
Experimental results of the TSBM^{2}h for Brandimarte (1993) instances. The INSA algorithm was used in the operation scheduling module.

Problem	$n \times m$	Flex.	o	t_s (s)	t_p (s)	Speedup s
Mk01	10 × 6	2.09	55	133.61	10.79	12.38
Mk02	10 × 6	4.10	58	218.02	10.55	20.67
Mk03	15 × 8	3.01	150	6495.35	136.19	47.69
Mk04	15 × 8	1.91	90	620.69	29.59	20.98
Mk05	15 × 4	1.71	106	1449.80	74.55	19.45
Mk06	10 × 15	3.27	150	8094.39	147.83	54.75
Mk07	20 × 5	2.83	100	1939.33	57.92	33.48
Mk08	20 × 10	1.43	225	8950.91	643.39	13.91
Mk09	20 × 10	2.53	240	24586.00	641.88	38.30
Mk10	20 × 15	2.98	240	31990.55	593.49	53.90

were obtained for the problem instances with a bigger number of jobs n and the number of operations o. In this phase the simple INSA algorithm was applied in the operation scheduling module of the parallel meta^{2}heuristics.

For Barnes and Chambers (1996) test instances an average number of equivalent machines per operation is between 1.07 and 1.30; for Hurink et al. (1994) test instances it equals 2 for each instance from set. Test instances with greater number of equivalent ma-

Table 2
Experimental results of the TSBM^{2}h for Barnes and Chambers (1996) instances. The INSA algorithm was used in the operation scheduling module.

Problem	$n \times m$	Flex.	o	t_s (s)	t_p (s)	Speedup s
mt10c1	10 × 11	1.10	100	69.27	28.18	2.46
mt10cc	10 × 12	1.20	100	130.47	27.28	4.78
mt10x	10 × 11	1.10	100	69.26	28.04	2.47
mt10xx	10 × 12	1.20	100	134.78	28.01	4.81
mt10xxx	10 × 13	1.30	100	133.99	28.03	4.78
mt10xy	10 × 12	1.20	100	130.93	27.43	4.77
mt10xyz	10 × 13	1.30	100	187.19	26.37	7.10
setb4c9	15 × 11	1.10	150	333.71	96.52	3.46
setb4cc	15 × 12	1.20	150	631.39	92.71	6.81
setb4x	15 × 11	1.10	150	332.64	96.90	3.43
setb4xx	15 × 12	1.20	150	654.09	95.27	6.87
setb4xxx	15 × 13	1.30	150	648.42	94.43	6.87
setb4xy	15 × 12	1.20	150	632.55	92.95	6.81
setb4xyz	15 × 13	1.30	150	896.66	88.54	10.13
seti5c12	15 × 16	1.07	225	747.64	340.46	2.20
seti5cc	15 × 17	1.13	225	1458.94	335.48	4.35
seti5x	15 × 16	1.07	225	753.07	342.35	2.20
seti5xx	15 × 17	1.13	225	1493.45	341.35	3.38
seti5xxx	15 × 18	1.20	225	1481.39	339.62	4.36
seti5xy	15 × 17	1.13	225	1459.27	335.42	4.35
seti5xyz	15 × 18	1.20	225	2123.35	325.94	6.51

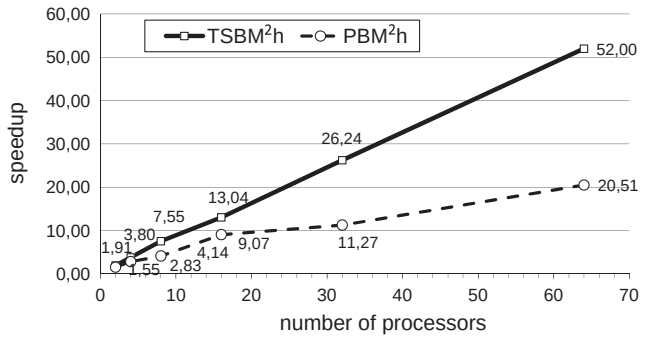

Fig. 8. Comparision of parallel tabu search TSBM^{2}h and population based PBM^{2}h algorithms speedups.

Table 3
Values of the obtaining solutions for Barnes and Chambers (1996) instances. The TSAB algorithm was used in the operation scheduling module of the TSBM^{2}h. The new best known solutions are marked out by a bold font.

Problem	$n \times m$	(LB,UB)	MG (Mastrolilli & Gambardella, 2000)	hGA (Gao et al., 2008)	TSBM^{2}h
mt10c1	10 × 11	(655,927)	928	927	927
mt10cc	10 × 12	(655,914)	910	910	**908**
mt10x	10 × 11	(655,929)	918	918	922
mt10xx	10 × 12	(655,929)	918	918	918
mt10xxx	10 × 13	(655,936)	918	918	918
mt10xy	10 × 12	(655,913)	906	905	905
mt10xyz	10 × 13	(655,849)	847	849	849
setb4c9	15 × 11	(857,924)	919	914	914
setb4cc	15 × 12	(857,909)	909	914	**907**
setb4x	15 × 11	(846,937)	925	925	925
setb4xx	15 × 12	(846,930)	925	925	925
setb4xxx	15 × 13	(846,925)	925	925	925
setb4xy	15 × 12	(845,924)	916	916	**910**
setb4xyz	15 × 13	(838,914)	905	905	**903**
seti5c12	15 × 16	(1027,1185)	1174	1175	1174
seti5cc	15 × 17	(955,1136)	1136	1138	1136
seti5x	15 × 16	(955,1218)	1201	1204	**1198**
seti5xx	15 × 17	(955,1204)	1199	1202	**1197**
seti5xxx	15 × 18	(955,1213)	1197	1204	1197
seti5xy	15 × 17	(955,1148)	1136	1136	1136
seti5xyz	15 × 18	(955,1127)	1125	1126	1128

chines per operation are more difficult to solve because there is more possible assignments of operations to machines. PBM^{2}h gives better result for Hurink et al. (Table 5) instances because the size of the search space in this algorithm is bigger than in TSBM^{2}h. Increasing the exploration measure (i.e. executing local optimization procedure for a longer time) gives better results for PBM^{2}h in comparison with TSBM^{2}h. PBM^{2}h gives better results than TSBM^{2}h but computation time is longer. A number of equivalent machines per operation can be used for the proposed algorithms parameters determination. For example if a number of equivalent machines per operation is low in a particular test instance, there should be executed more iterations in the operation scheduling module.

The second phase of the tests refers to obtaining as good results of the cost function as possible. In this phase a specialized TSAB algorithm of Nowicki and Smutnicki (1996) was used in the operation scheduling module of the parallel meta^{2}heuristics. Despite of being more time-consuming the quality of the obtained results is much better than in the case of using INSA. Results obtained by the TSBM^{2}h and PBM^{2}h for Barnes and Chambers (1996) and Hurink et al. (1994) test instances are showed in Tables 3 and 5 (results), Tables 4 and 6 (computation times). These results are compared to the results of MG (Mastrolilli & Gambardella, 2000)

Table 4
Computation times for Barnes and Chambers (1996) instances.

Problem	$n \times m$	MG (Mastrolilli & Gambardella, 2000)	hGA (Gao et al., 2008)	TSBM^{2}h
mt10c1	10 × 11	2.33	12.87	44.50
mt10cc	10 × 12	10.04	12.24	65.74
mt10x	10 × 11	4.31	12.69	55.11
mt10xx	10 × 12	1.73	11.70	50.18
mt10xxx	10 × 13	1.10	11.52	47.57
mt10xy	10 × 12	4.02	12.24	76.26
mt10xyz	10 × 13	5.50	10.71	110.13
setb4c9	15 × 11	14.02	45.99	111.40
setb4cc	15 × 12	12.95	38.79	151.19
setb4x	15 × 11	7.45	43.20	93.76
setb4xx	15 × 12	14.87	42.57	92.28
setb4xxx	15 × 13	7.99	36.09	89.405
setb4xy	15 × 12	3.15	41.49	150.83
setb4xyz	15 × 13	7.35	39.15	152.67
seti5c12	15 × 16	19.49	72.27	351.32
seti5cc	15 × 17	11.91	63.00	670.35
seti5x	15 × 16	15.85	66.78	257.75
seti5xx	15 × 17	23.64	63.72	264.58
seti5xxx	15 × 18	23.51	63.36	226.29
seti5xy	15 × 17	11.91	63.18	675.40
seti5xyz	15 × 18	17.13	57.96	717.60

Table 5
Comparison of results obtained by MG (Mastrolilli & Gambardella, 2000) and proposed tabu search based TSBM^{2}h and population-based PBM^{2}h algorithms for test instances taken from Hurink et al. (1994).

Problem	$n \times m$	(LB,UB)	MG (Mastrolilli & Gambardella, 2000)	PBM^{2}h	TSBM^{2}h
la01	10 × 5	(570,574)	571	572	574
la02	10 × 5	(529,532)	530	530	532
la03	10 × 5	(477,479)	478	478	482
la04	10 × 5	(502,504)	502	502	509
la05	10 × 5	(457,458)	457	458	462
la06	15 × 5	(799,800)	799	799	801
la07	15 × 5	(749,750)	750	750	751
la08	15 × 5	(765,767)	765	765	767
la09	15 × 5	(853,854)	853	853	856
la10	15 × 5	(804,805)	804	805	807
la11	20 × 5	(1071,1072)	1071	1071	1072
la12	20 × 5	936	936	936	937
la13	20 × 5	1038	1038	1038	1039
la14	20 × 5	1070	1070	1070	1071
la15	20 × 5	(1089,1090)	1090	1090	1090

Table 6
Computation times for Hurink et al. (1994) instances.

Problem	$n \times m$	MG (Mastrolilli & Gambardella, 2000)	TSBM^{2}h
la01.fjs	10 × 5	1.97	16.81
la02.fjs	10 × 5	1.31	14.67
la03.fjs	10 × 5	1.36	15.52
la04.fjs	10 × 5	0.62	12.65
la05.fjs	10 × 5	1.78	5.767
la06.fjs	15 × 5	2.99	5.327
la07.fjs	15 × 5	1.13	12.22
la08.fjs	15 × 5	0.35	1.780
la09.fjs	15 × 5	2.29	9.585
la10.fjs	15 × 5	1.32	17.36
la11.fjs	20 × 5	2.56	19.87
la12.fjs	20 × 5	0.08	13.31
la13.fjs	20 × 5	0.90	47.34
la14.fjs	20 × 5	0.28	3.670
la15.fjs	20 × 5	1.76	19.07

and hGA (Gao et al., 2008) algorithms. The proposed parallel TSBM^{2}h algorithm managed to obtain the average relative percentage deviation to the best known solution of the Barnes and Chambers benchmark instances on the level of 0.014% versus 0.036% of the MG (Mastrolilli & Gambardella, 2000) algorithm of Mastrolilli and Gambardella and 0.106% of the hGA (Gao et al., 2008) algorithm of Gao et al. Due to this approach it was possible to obtain 6 the new best known solutions for the benchmarks of Barnes and Chambers (1996), for instances `mtl0cc` (the new value 908), `setb4c9` (907), `setb4xy` (910), `setb4xyz` (903), `seti5x` (1198) and `seti5xx` (1197).

Authors of parallel VNS algorithm for the FJSP (Yazdani et al., 2010) as well as authors of island-model parallel genetic algorithm (Defersha & Chen, 2009) do not report computations time which makes a comparison to they algorithms impossible in the area of efficiency. In the area of solutions quality our tabu search based meta^{2}heuristics *TSBM^{2}h* allow us to find the new best known solutions for Barnes and Chambers (1996) instances so we can say *TSBM^{2}h* approach is more effective.

6. Conclusions

We have discussed a new approach to the scheduling problems with parallel machines, where the assignment of operations to machines defines a classical problem without parallel machines. We propose double-level parallel metaheuristics, where each solution of the higher level, i.e. jobs assignment to machines, defines an NP-hard job shop problem, which we solve by the second metaheuristics – we call such an approach meta^{2}heuristics. On the Machine Selection Module (higher level), we apply two metaheuristics: the tabu search and the population-based approach to determine an assignment of operations to machines. The distribute tabu search threads are used as Operations Scheduling Module (lower level). Using the exact algorithms on both levels (i.e. branch and bound) makes possible to obtain an optimal solution of the problem.

Acknowledgement

The work was partially supported by the Polish Ministry of Science and Higher Education, Grant No. N N514 232237.

References

Alba, E. (2005). *Parallel metaheuristics. A new class of algorithms*. Wiley & Sons Inc.
Barnes, J. W., Chambers, J. B. (1996). Flexible job shop scheduling by tabu search. Graduate program in operations research and industrial engineering, The University of Texas at Austin, Technical Report Series: ORP96-09.
Bożejko, W., & Wodecki, M. (2009). Solving permutational routing problems by population-based metaheuristics. *Computers & Industrial Engineering, 57*, 269–276.
Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research, 41*, 157–183.
Dauzère-Pérès, S., & Pauli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job shop scheduling problem using tabu search. *Annals of Operations Research, 70*(3), 281–306.
Defersha, F. M., Chen, M. (2009). A coarse-grain parallel genetic algorithm for flexible job-shop scheduling with lot streaming. In *Proceedings of the international conference on computational science and engineering* (Vol. 1, pp. 201–208).
Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research, 35*, 2892–2907.
Grabowski, J. (1979). Generalized problems of operations sequencing in the discrete production systems (in Polish). *Monographs*, 9, Scientific Papers of the Institute of Technical Cybernetics of Wrocław Technical Univercity.
Grabowski, J., & Wodecki, M. (2005). A very fast tabu search algorithm for the job shop problem. In C. Rego & B. Alideaee (Eds.), *Adaptive memory and evolution; tabu search and scatter search* (pp. 117–144). Dordrecht: Kluwer Academic Publishers.
Ho, N. B., Tay, J. C. (2004). GENACE: An efficient cultural algorithm for solving the flexible job-shop problem. In *IEEE international conference on robotics and automation* (pp. 1759–1766).
Hurink, E., Jurisch, B., & Thole, M. (1994). Tabu search for the job shop scheduling problem with multi-purpose machine. *Operations Research Spektrum, 15*, 205–215.
Jia, H. Z., Nee, A. Y. C., Fuh, J. Y. H., & Zhang, Y. F. (2003). A modified genetic algorithm for distributed scheduling problems. *International Journal of Intelligent Manufacturing, 14*, 351–362.
Kacem, I., Hammadi, S., & Borne, P. (2002). 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.
Mastrolilli, M., & Gambardella, L. M. (2000). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling, 3*(1), 3–20.
Nowicki, E., & Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research, 91*, 160–175.
Pauli, J. (1995). A hierarchical approach for the FMS schduling problem. *European Journal of Operational Research, 86*(1), 32–42.
Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research, 35*, 3202–3212.
Pinedo, M. (2002). *Scheduling: Theory, algorithms and systems*. Englewood cliffs, NJ: Prentice-Hall.
Talbi, E.-G. (2002). A taxonomy of hybrid metaheuristics. *Journal of Heuristics, 8*(5), 541–564.
Yazdani, M., Amiri, M., & Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications: An International Journal, 37*(1), 678–687.