Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 2236–2245

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

Parallel estimation of the cost function for the flexible scheduling problem☆

Wojciech Bożejko[a], Mariusz Uchroński[a], Mieczysław Wodecki[b,1,*]

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

Abstract

The aim of this paper is to show how to determine the neighborhood of the complex discrete optimization problem and how to search it in the parallel environment, this being illustrated by an example of the hybrid scheduling, more precisely a flexible job shop problem. We present a parallel single-walk approach in this respect. A theoretical analysis based on PRAM model of parallel computing has been made. We propose a cost-optimal method of neighborhood generation parallelization.

Keywords: parallel algorithm, scheduling, flexible job shop, metaheuristics

1. Introduction

Flexible job shop problems constitute a generalization (hybridization) of the classic job shop problem. In this section, we discuss a flexible job shop problem in which 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 [12]), they are not effective for instances with more than 20 jobs and 10 machines.

Many approximate algorithms, chiefly metaheuristic, have been proposed. Nowicki and Smutnicki [10] proposed a tabu search approach using block properties for the special case of the problem considered (flow shop problem with parallel machines). Hurink [8] developed the tabu search method for this problem. Also Dauzère-Pérès and Pauli [5] 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 [9] proposed a tabu search procedure with effective neighborhood functions for the flexible job shop problem.

☆The work was supported by MNiSW Poland, within the grant No. N N514 232237.

*

Email address: mwd@ii.uni.wroc.pl (Mieczysław Wodecki)
[1]Corresponding author

doi:10.1016/j.procs.2011.04.244

Many authors have proposed a method of assigning operations to machines and then determining sequence of operations on each machine. This approach was followed by Brandimarte [4] and Pauli [11]. These authors solved the assignment problem (i.e., using dispatching rules) and next applied metaheuristics to solve the job shop problem. Genetic approaches have been adopted to solve the flexible job shop problem, too. Recent works are those of Ho and Tay [7] and Bożejko et al. [3]. Gao et al. [6] proposed the hybrid genetic and variable neighborhood descent algorithm for this problem.

This paper constitutes a continuation of researches presented in the paper Bożejko et al. [2].

2. Problem formulation

The flexible job shop problem (FJSP), also called the general job shop problem with parallel machines, can be formulated as follows. Let $\mathcal{J} = \{1, 2, \ldots, n\}$ be a set of jobs which have to be executed on machines from the set $\mathcal{M} = \{1, 2, \ldots, m\}$. 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 a dedicated type of machine (from the nest) within a fixed time. The problem consists in the allocation of jobs to machines of dedicated type and in determining the schedule of jobs execution on each machine to minimize the total jobs finishing time. The following constraints have to be fulfilled:

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

Each operation is assigned to one nest only. It is necessary to make a partition of operations assigned to machines in each nest. The method of partitioning has an influence on all jobs completion time, that is the value of the job shop problem solution. Generally, we are looking for such a partition whose cost function value (of the corresponding job shop problem) is minimal.

Let $\Theta = (Q, \pi(Q)) \in \Phi^\circ$ be a feasible solution of FJSP where $Q = (Q^1, Q^2, \ldots, Q^m)$ is the machine workload, ϱ_i is the number of operations executed on machine M_i (i.e., $\varrho_i = |Q^i|$) and

$$\pi(Q) = (\pi_1(Q), \pi_2(Q), \ldots, \pi_m(Q)) \tag{1}$$

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

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

By $t^i_j(k, l)$ we denote a *transfer* type move (a *t-move*, for short) 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 $\Theta = (Q, \pi) \in \Phi^\circ$ (by Φ° we denote a set of feasible solutions of the FJSP, see Bożejko et al. [2]), a new solution $\Theta' = (Q', \pi')$ such that

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

$$\pi'_i = (\pi_i(1), \pi_i(2), \ldots, \pi_i(\boldsymbol{k-1}), \pi_i(\boldsymbol{k+1}), \ldots, \pi_i(\varrho_i - 1), \tag{3}$$

$$\pi'_j = (\pi_j(1), \pi_j(2), \ldots, \pi_j(\boldsymbol{l-1}), \pi_i(\boldsymbol{k}), \pi_j(\boldsymbol{l}), \ldots, \pi_j(\varrho_j + 1). \tag{4}$$

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

$$Q'^v = Q^v, \quad v \neq i, j, \quad v = 1, 2, \ldots, m \quad \text{and} \quad Q'^i = Q^i \setminus \{\pi_i(k)\}, \quad Q'^j = Q^j \cup \{\pi_i(k)\}. \tag{5}$$

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

Remark 1. *An upper bound of the number of t-moves is* $O(qm^2o^2)$.

The execution of *t-move* causes an operation to transfer from one machine to another, i.e., a new machine workload in the nest. Therefore, it is possible to obtain any machine workload from any solution (machine workload) by executing *t-moves*. If τ is a *t-move*, we denote by $\tau(\Theta)$ a solution generated from Θ by executing a move τ (see (2)–(5)). For a fixed feasible solution Θ, let $\mathcal{T}(\Theta)$ be a set of all *t-moves*. A *neighborhood* Θ is a set

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

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

It was mentioned at the beginning of this section that the solution of the FJSP consists of two steps. The first, determination of machine workload and the second, determination of processing order of operations, i.e., a job shop problem solving. Let $\Theta = (Q, \pi)$ be a feasible solution of the FJSP. The new machine workload Q' will be generated from the workload Q as follows:

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

The number of all *t-moves* can be huge, so we omit some of them and consider only those which can offer an improvement of the goal function value. Moreover, we do not determine an exact goal function value of solutions generated by *t-moves*, but approximate them only. As computational experiments proved, this causes a significant algorithm work acceleration with little results aggravation. In the further part, we precisely describe methods of eliminating superfluous moves from the neighborhood (6) as well as methods of estimating a goal function value.

Neighborhood determination. Execution of a *t-move* can lead to 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 corresponding algorithm has a computational complexity $O(o)$ where o is the number of all operations. Further on we prove theorems which make it possible to check feasibility of solutions (i.e., acyclicity of corresponding graphs) generated by *t-moves* in constant time.

Let $\Theta = (Q, \pi)$ be a feasible solution where $Q = (Q^1, Q^2, \ldots, Q^m)$ is the machine workload and $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ is a concatenation of m permutations. A permutation π_i determines a processing order of operations from the set 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 a processing order of operations from the set Q^i on the machine M_i and π_j – a processing order of operations from the set Q^j on the machine M_j. For an operation $\pi_i(k) \in Q^i$ we define two parameters connected with paths in the graph $G(\Theta)$.

The first parameter is

$$\eta_j(k) = \begin{cases} 1, \ \text{if there is no path } C(\pi_j(v), \pi_i(k)) \ \forall\, v = 1, 2, \ldots, \varrho_j, \\ 1 + \max\limits_{1 \le v \le \varrho_j} \{\text{there is a path } C(\pi_j(v), \pi_i(k))\}, \ \text{otherwise.} \end{cases} \tag{7}$$

Thus, there does not exist any path to the operation (vertex) $\pi_i(k)$ from any of the operations placed in the permutation π_j in positions $\eta_j(k), \eta_j(k) + 1, \ldots, \varrho_j$ (where $\varrho_j = |Q^j|$) in the graph $G(\Theta)$.

The second parameter is

$$\rho_j(k) = \begin{cases} 1 + \varrho_j, \ \text{if there is no path } C(\pi_j(v), \pi_i(k)) \ \forall\, v = 1, 2, \ldots, \varrho_j, \\ 1 + \min\limits_{\eta_j(k) \le v \le \varrho_j} \{\text{there is a path } C(\pi_i(k), \pi_j(v))\}, \ \text{otherwise.} \end{cases} \tag{8}$$

From the definition formulated above it follows that in the graph there is no 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. Now we prove two theorems characterizing a *t-move* whose execution generates an unfeasible solution. These theorems constitute a constructional base for very efficient neighborhoods. The structure of assumptions allows an easy implementation in the parallel computing environment, such as GPUs.

Now we show a theorem which constitutes a base for eliminating some *t-moves* during the process of neighborhood generation. Let us assume that for an assignment Q the concatenation π is the optimal operations schedule on machines, $G(\Theta)$ is a graph connected with solution $\Theta = (Q, \pi)$ and $C_{\max}(\Theta)$ is a cost function value, i.e., the critical path length in the graph $G(\Theta)$.

Theorem 1 ([1]). *Let $\Theta = (Q, \pi)$ be a feasible solution for the* FJSP *and let $\mathcal{B} = (B^1, B^2, \ldots, B^r)$ be a sequence of critical path blocks in the graph $G(\Theta)$. If $\Theta' = (Q', \pi')$ is a feasible solution which was generated from Θ by machine workload 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 a different machine (in the same nest).*

Let Θ be a feasible solution, $\mathcal{B}$ – a sequence of critical path blocks in the graph $G(\Theta)$ and $\mathcal{T}$ – a set of *t-moves* defined for the Θ. We denote by $\mathcal{T}^{out}(\Theta)$ a set of those moves from $\mathcal{T}(\Theta)$ which consider operations not belonging to any block. Directly from Theorem 1 there follows a property which constitutes a base for eliminating superfluous moves.

Property 1. *If a feasible solution Θ' is generated from Θ by executing a t-move belonging to the set $\mathcal{T}^{out}(\Theta)$ then*

$$C_{\max}(\Theta') \geq C_{\max}(\Theta). \tag{9}$$

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

$$\mathcal{T}^{subm}(\Theta) = \{t^i_j(v, w) \in \mathcal{T}^{acc} : \ v \in \{a^k, b^k\}, \quad w \in \{\eta_j(v), \rho_j(v)\}, \ k = 1, 2, \ldots, r. \tag{10}$$

The neighborhood Θ constitutes the set of feasible solutions

$$\mathcal{N}(\Theta) = \{\tau(\Theta) : \ \tau \in \mathcal{T}^{subm}(\Theta)\}. \tag{11}$$

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

In practice, we should select the best element of the neighborhood (e.g. inside a metaheuristic). Making use of parallel computing environment we can follow one of the approaches below.

- We get as many processors as there are blocks r. Next, in the loop, each processor checks the cost of the operation insertion to another machine of the same type, concurrently calculating the minimal value of such an insertion.
- We get as many processors as there are pairs of machines of the same type. We calculate the minimal value in logarithmic time using a tree scheme of parallel calculations.

The first approach leads to the cost-optimal method. However, the second approach, using much greater number of processors, leads us to obtaining shorter computing time.

Methods of the cost function value estimation. Each solution $\Theta = (Q, \pi)$ is a pair whose first element is a set sequence – machine workloads. A new assignment will be determined by choosing an element with the lowest cost function value from the neighborhood (11). This requires a critical path to be determined for each element of the neighborhood. To speed this procedure up, we will compute the lower bounds of the cost function value as a choosing criterion. In this section, we will present methods of determining lower bounds.

Let $\Theta = (Q, \pi)$ be a feasible solution where $Q = (Q^1, Q^2, \ldots, Q^m)$ constitutes machine workload and $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ is a concatenation of m permutations. Further, let $\mathcal{B} = (B^1, B^2, \ldots, B^r)$ be a sequence of critical path blocks in the graph $G(\Theta)$. We consider two machines M_i and M_j belonging to the same nest. On machine M_i

there are executed operations from the set Q^i in the order $\pi_i = (\pi_i(1), \pi_i(2), \ldots, \pi_i(\varrho_i))$ and on machine M_j operations from the set Q^j in the order $\pi_j = (\pi_j(1), \pi_j(2), \ldots, \pi_j(\varrho_j))$. Let us assume that the block

$$B^k = (\pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k - 1), \pi_i(b^k)), \tag{12}$$

includes operations executed on machine M_i. For simplicity, we omit index k which denotes the block number. Therefore, $\pi_i(a)$ is the first and $\pi_i(b)$ is the last operation of the block B^k.

According to the strategy of searching neighborhood $\mathcal{N}(\Theta)$ we are looking for such a move $\tau \in \mathcal{T}^{subm}(\Theta)$ which generates a graph $G(\tau(\Theta))$ – a feasible solution with possibly the lowest estimation of the critical path length (i.e., cost function value). For moves from $\mathcal{T}^{subm}(\Theta)$ which transfer the first operation $\pi_i(a^k)$ of the block B^k to position $\eta_j(a^k)$ or $\rho_j(a^k)$ in the permutation π_j we introduce the notions

$$\Delta^{a^k}_{\mathrm{x}(a^k)} = \max\{L_1^{\mathrm{x}(a^k)}, L_2^{\mathrm{x}(a^k)}, L_3^{\mathrm{x}(a^k)}, L_4^{\mathrm{x}(a^k)}\}, \quad \mathrm{x}(a^k) \in \{\eta_j(a^k),\ \rho_j(a^k)\}, \tag{13}$$

where

$$L_1^{\mathrm{x}(a^k)} = L(s, \pi_i(a^k - 1)) - L(s, \pi_i(a^k)), \tag{14}$$

$$L_2^{\mathrm{x}(a^k)} = L(s, \pi_i(a^k + 1)) - L(s, \pi_i(a^k)) - p_{\pi_i(a^k+1)}, \tag{15}$$

$$L_3^{\mathrm{x}(a^k)} = L(s, \pi_j(1)) + \sum_{h=2}^{\varrho_j - 1} p_{\pi_i(h)} + p_{\pi_i(a^k)} + L(\pi_j(\varrho_j), c) + L(s, \pi_i(a^k)) - \sum_{h=a^k+1}^{b^k-1} p_{\pi_i(h)} - L(\pi_i(b^k), c),$$

$$L_4^{\mathrm{x}(a^k)} = \sum_{h=\mathrm{x}(a^k)+1}^{\varrho_j - 1} p_{\pi_j(h)} + L(\pi_j(\varrho_j), c) + \sum_{h=a^k+1}^{b^k-1} p_{\pi_i(h)} - L(\pi_i(b^k), c).$$

Similarly, for moves from $\mathcal{T}^{subm}(\Theta)$ which move the last operation $\pi_i(b^k)$ of the block B^k to position $\eta_j(a^k)$ or $\rho_j(a^k)$ in the permutation π_j we introduce the notions

$$\Delta^{b^k}_{\mathrm{y}(b^k)} = \max\{L_1^{\mathrm{y}(b^k)}, L_2^{\mathrm{y}(b^k)}, L_3^{\mathrm{y}(b^k)}, L_4^{\mathrm{y}(b^k)}\}, \quad \mathrm{y}(b^k) \in \{\eta_j(b^k),\ \varrho_j(b^k)\}, \tag{16}$$

where

$$L_1^{\mathrm{y}(b^k)} = L(\pi_i(b^k - 1), c) - p_{\pi(b^k-1)} - L(\pi_i(b^k), c), \tag{17}$$

$$L_2^{\mathrm{y}(b^k)} = L(\pi_i(b^k + 1), c) - L(\pi_i(b^k), c), \tag{18}$$

$$L_3^{\mathrm{y}(b^k)} = L(s, \pi_j(1)) + \sum_{h=2}^{\varrho_j - 1} p_{\pi_j(h)} + p_{\pi_i(b^k)} + L(\pi_j(\varrho_j), c) + L(s, \pi_i(a^k)) - \sum_{h=a^k+1}^{b^k-1} p_{\pi_i(h)} - L(\pi_i(b^k), c),$$

$$L_4^{\mathrm{y}(b^k)} = L(s, \beta(1)) + \sum_{h=2}^{\mathrm{y}(b^k)-1} p_{\pi_j(h)} - L(s, \pi_i(a^k)) - \sum_{h=a^k+1}^{b^k-1} p_{\pi_i(h)}.$$

Now, we will prove theorems which allow us to estimate a cost function value for a solution generated from Θ by shifting the first operation $\pi_i(a)$ from the block B^k (executing a *t-move*) to position $\eta_j(a)$ or $\rho_j(a)$ on machine M_j.

Theorem 2. *If the solution $\Theta' = (Q', \pi')$ is generated from the $\Theta = (Q, \pi)$ by executing the move $t^i_j(a^k, \mathrm{x}(a^k)) \in \mathcal{T}^{subm}(\Theta)$, $\mathrm{x}(a^k) \in \{\eta_j(a^k), \rho_j(a^k)\}$ then*

$$L'(s, c) \geq L(s, c) + \Delta^{a^k}_{\mathrm{x}(a^k)}. \tag{19}$$

Proof. Let $\pi_i = (\pi_i(1), \pi_i(2), \ldots, \pi_i(\varrho_i))$ and $\pi_j = (\pi_j(1), \pi_j(2), \ldots, \pi_j(\varrho_j))$ be permutation of operations executing on machines M_i and M_j, respectively. The operations sequence $B^k = (\pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k))$ $(1 \leq a^k \leq b^k \leq \varrho_i)$ is a block on the machine M_i, i.e., B^k is a subsequence π_i.

For the notion simplification we assume that $a = a^k,\ b = b^k$ and $\alpha = \pi_i = (\alpha(1), \ldots, \alpha(a), \ldots, \alpha(b), \ldots, \alpha(u))$, $\beta = \pi_j = (\beta(1), \ldots, \beta(w))$ where $u = \varrho_i,\ w = \varrho_j$.

The graph $G(\Theta)$ is acyclic, so there exists a critical path $C(s,c)$ with the length $L(s,c)$. For each vertex $v \in O$ there is

$$C(s,c) = (C(s,v), C(v,c)) \tag{20}$$

and

$$L(s,c) = L(s,v) + C(v,c) - p_v. \tag{21}$$

Vertices of the critical path can be partitioned into subsequences

$$C(s,c) = (C(s,\alpha(a)), C(\alpha(a),\alpha(b)), C(\alpha(b),c))) \tag{22}$$

We consider the following paths: $d_1(s,c)$, $d_2(s,c)$, $d_3(s,c)$ and $d_4(s,c)$ from the vertex s to c in the graph $G(\Theta')$.

$$d_1(s,c) = (C'(s,\alpha(a-1)), C'(\alpha(a-1),\alpha(b)), C'(\alpha(b),c)), \tag{23}$$
$$d_2(s,c) = (C'(s,\alpha(a+1)), C'(\alpha(a+1),\alpha(b)), C'(\alpha(b),c)), \tag{24}$$
$$d_3(s,c) = (C'(s,\beta(1), C'(\beta(1),\beta(w)), C'(\beta(w),c)), \tag{25}$$
$$d_4(s,c) = (C'(s,\alpha(a)), C'(\alpha(a) = \beta(\mathrm{x}(a)),\beta(w)), C'(\beta(w),c). \tag{26}$$

Taking advantage of the fact that

$$C'(s,\alpha(a-1)) = C(s,\alpha(a-1)) \text{ and } C'(\alpha(b),c)) = C(\alpha(b),c)) \tag{27}$$

we obtain

$$d_1(s,c) = (C(s,\alpha(a-1)), C'(\alpha(a-1),\alpha(b)), C(\alpha(b),c)) \tag{28}$$

and similarly

$$d_2(s,c) = (C(s,\alpha(a+1)), C'(\alpha(a+1),\alpha(b)), C(\alpha(b),c)), \tag{29}$$
$$d_3(s,c) = (C(s,\beta(1), C'(\beta(1),\beta(w)), C(\beta(w),c)), \tag{30}$$
$$d_4(s,c) = (C(s,\alpha(a)), C'(\alpha(a) = \beta(\mathrm{x}(a)),\beta(w)), C(\beta(w),c). \tag{31}$$

Therefore the length of these paths (in the graph $G(\Theta')$) can be defined by length of some paths in the graph $G(\Theta)$. These are as follows

$$l^1(s,c) = L(s,\alpha(a-1)) + \sum_{h=a+1}^{b-1} p_{\alpha(h)} + L(\alpha(b),c), \tag{32}$$
$$l^2(s,c) = L(s,\alpha(a+1)) + \sum_{h=a+2}^{b-1} p_{\alpha(h)} + L(\alpha(b),c), \tag{33}$$
$$l^3(s,c) = L(s,\beta(1)) + \sum_{h=2}^{w-1} p_{\beta(h)} + p_{\alpha(a)} + L(\beta(w),c), \tag{34}$$
$$l^4(s,c) = L(s,\alpha(a) + \sum_{h=\mathrm{x}(a)+1}^{w-1} p_{\beta(h)} + L(\beta(w),c). \tag{35}$$

As the graph $G(\Theta')$ is acyclic, there exists a critical path $C'(s,c)$ whose length must not be shorter than the length of any other paths from vertex s to c in $G(\Theta')$. Therefore

$$L'(s,c) \geq l^1(s,c), \tag{36}$$
$$L'(s,c) \geq l^2(s,c), \tag{37}$$
$$L'(s,c) \geq l^3(s,c), \tag{38}$$
$$L'(s,c) \geq l^4(s,c). \tag{39}$$

From this and using (22) we obtain

$$\begin{aligned}
L'(s,c) \geq\ & \max\{l^1(s,c),\ l^2(s,c),\ l^3(s,c),\ l^4(s,c)\} = \max\{L(s,\alpha(a-1)) \\
& + \sum_{h=a+1}^{b-1} p_{\alpha(h)} + L(\alpha(b),c),\ L(s,\alpha(a+1)) + \sum_{h=a+2}^{b-1} p_{\alpha(h)} \\
& +\ L(\alpha(b),c)),\ L(s,\beta(1)) + \sum_{h=2}^{w-1} p_{\alpha(h)} + p_{\alpha(a)} + L(\beta(w),c), \\
& \quad L(s,\alpha(a) + \sum_{h=\mathrm{x}(a)}^{w-1} p_{\alpha(h)} + L(\beta(w),c)\} \\
=\ & \max\{L(s,c) + L(s,\alpha(a-1)) - L(s,\alpha(a)), L(s,c) + L(s,\alpha(a+1)) + \\
& -\ L(s,\alpha(a)) - p_{\alpha(a+1)}, L(s,c) + L(s,\beta(1)) + \sum_{h=2}^{w-1} p_{\beta(h)} + p_{\alpha(a)} + \\
& +\ L(\beta(w),c) - L(s,\pi_i(a^k)) - \sum_{h=a^k+1}^{b^k-1} p_{\pi_i(h)} - L(\pi_i(b^k),c), \\
& \quad L(s,c) + \sum_{h=\mathrm{x}(a)+1}^{w-1} p_{\beta(h)} + L(\beta(w),c) - \sum_{h=a+1}^{b-1} p_{\alpha(h)} - L(\alpha(b),c)\} \\
=\ & L(s,c) + \max\{L_1^{\mathrm{x}(a^k)},\ L_2^{\mathrm{x}(a^k)},\ L_3^{\mathrm{x}(a^k)},\ L_4^{\mathrm{x}(a^k)}\} = L(s,c) + \Delta^{a}_{\mathrm{x}(a)},
\end{aligned}$$

which completes the proof of the theorem. ∎

The next theorem is related with moving the last operation $\pi(b^k)$ from the block B^k to machine M_j.

Theorem 3. *If the solution $\Theta' = (Q', \pi')$ is generated from the $\Theta = (Q, \pi)$ by executing the move $t^i_j(b^k, \mathrm{y}(b^k)) \in \mathcal{T}^{subm}$, $\mathrm{y}(b^k) \in \{\eta_j(b^k),\ \rho_j(b^k)\}$ then*

$$L'(s,c) \geq L(s,c) + \Delta^{b^k}_{\mathrm{y}(b^k)}. \tag{40}$$

The proof is similar to that of the previous theorem.

Remark 2. *Values l^k, $k = 1,2,3,4$, can be determined sequentially in time $O(n) = O(o)$. These calculations can be done in parallel in time $O(\log n) = O(\log o)$ using $O\left(\frac{n}{\log n}\right) = O\left(\frac{o}{\log o}\right)$-processor CREW PRAM.*

Moving the operation $\pi(a^k)$ to the position $\eta_j(a^k)$ or $\rho_j(a^k)$ the graph is generated in which the lower bound of the length of the critical path from vertex s to c is the value of the expression $L(s,c) + \Delta^{a^k}_{\eta_j(a^k)}$ (or $L(s,c) + \Delta^{a^k}_{\rho_j(a^k)}$). That is why the expression $\Delta^{a^k}_{\mathrm{x}(a^k)},\ \mathrm{x}(a^k) \in \{\eta_j(a^k),\ \rho_j(a^k)\}$ can be used to determine the operation (i.e., an element from the neighborhood) that will be moved.

Similarly, $L(s,c) + \Delta^{b^k}_{\eta_j(b^k)}$ (or $L(s,c) + \Delta^{b^k}_{\rho_j(b^k)}$) is a lower bound of the critical path length in the graph generated by moving an operation $\pi(b^k)$ to positions $\eta_j(b^k)$ or $\rho_j(b^k)$ and the expression $\Delta^{b^k}_{\mathrm{y}(b^k)},\ \mathrm{y}(b^k) \in \{\eta_j(b^k),\ \rho_j(b^k)\}$ can be employed to select an element from the neighborhood.

We choose the operation $\pi(v) \in O$ such that

$$\Delta^{v}_{\chi(v)} = \min_{1\leq k\leq r} \min\{\Delta^{z}_{\mu(z)} :\ z \in \{a^k, b^k\},\ \mu(z) \in \{\eta_j(z), \rho_j(z)\}\} \tag{41}$$

The minimal value $\Delta^{v}_{\chi(v)}$ is connected with the best *t-move* which consists in moving the first or the last operation from some block to another machine. From Theorems 2 and 3 it follows that if $\Delta^{v}_{\chi(v)} > 0$, then the critical path length $L'(s,c) > L(s,c)$ in the generated graph $G(\Theta')$.

Summing up, for the solution $\Theta = (Q, \pi)$ (fixed machine workload Q) we propose the following method of the new assignment Q' determination. In the graph $G(\Theta)$ we determine the critical path $C(s, c)$ (if there are more than one, we choose any of them) and we calculate its length $L(s, c) = C_{\max}(\Theta)$. Next, we determine the partition of the path into blocks $\mathcal{B} = (B^1, B^2, \ldots, B^r)$ and in accordance with (10) the set of moves $\mathcal{T}^{subm}(\Theta)$. Using (41) we determine $\Delta^v_{\chi(v)}$ and choose the best *t-move* $t^i_j(v, \chi(v))$. This move generates a solution (the new machine workload) from the neighborhood $\mathcal{N}(\Theta)$ with the lowest value of the lower bound of the cost function.

Machine workload rearrangement. The algorithm proposed here searches the neighborhood generated by *t-moves* transferring the first and the last operations of each block from the critical path to another machine. The computational complexity of the `NewPar` algorithm is $O(o^3)$ because of the complexity of creating a *t-move* neighborhood (Step 3). Determination of the longest paths (Step 1) can be done using Floyd's algorithm in time $O(o^3)$, or applying the recursive method based on topological sorting in time $O(o)$, maintaining the complexity $O(o^3)$ of the whole sequential algorithm.

Parallel determination of the workload. Now, we will show a parallel version of the `NewPar` algorithm designed to be executed on $O(o^2)$-processor CREW PRAM in time $O(o)$. An outline of the algorithm is presented in Figure 1.

Step 1 consists in: (1) sequential determination of the graph $G(\Theta) = (\mathcal{V}, \mathcal{R} \cup \mathcal{E}(\Theta))$ connected with the solution Θ and (2) parallel determination of the longest paths for all pairs of vertices in this graph, which can be done using parallel Floyd's algorithm. Because the graph has at most o vertices, parallel all-pairs the longest paths determination algorithm works in time $\Theta(o)$ using o^2-processor CREW PRAM. It is possible to determine the longest paths faster (using a greater number of processors), but in this case this is useless because Step 3 (neighborhood determination) has linear computational complexity $O(o)$.

Step 2 (block determination) can be executed in constant time $O(o)$ using as many processors as there are vertices on the longest path – at most o. Let us assign each processor to one vertex v lying on the critical path. It is enough to check by each processor if the machine number assigned to its vertex $\lambda(v)$ is the same as the machine number $\lambda(u)$ assigned to the next vertex u lying on the critical path. If it is not the same it means that the next block begins in u. Such a comparison can be made in time $O(1)$ using $O(o)$-processor CREW PRAM.

Step 3 (neighborhood determination) consists of two loops: external and internal one, which can be executed independently in parallel. Inside them each processor needs to determine feasible positions $\eta_j(a^k)$, $\rho_j(a^k)$, $\eta_j(b^k)$ and $\rho_j(b^k)$, which can be done in linear time $O(o)$. Afterwards, values $\Delta^{a^k}_{\eta_j(a^k)}$, $\Delta^{a^k}_{\rho_j(a^k)}$, $\Delta^{b^k}_{\eta_j(b^k)}$ and $\Delta^{b^k}_{\rho_j(b^k)}$ have to be calculated, which also needs time $O(o)$ (the sum of at most o elements has to be determined; see Theorem 3). The entire Step 3 requires $O(o^2)$ processors (to execute two loops in parallel) to be made in time $O(o)$.

Step 4 (the best *t-move* move determination) consists in choosing one move from $O(4r \cdot m_{\max})$ moves, where $m_{\max} = \max_{1 \le i \le q} m_i$ is the maximal number of machines in a nest. Because $O(4r \cdot m_{\max}) = O(rm) = O(o^2)$ therefore we need to use $O(o^2)$ processors to determine the minimal element of $O(o^2)$ elements in time $O(\log o^2) = O(2 \log o) = O(\log o)$. In fact, it is enough to use less processors, namely $O(\frac{o^2}{2 \log o})$ instead of $O(o^2)$ to maintain the same computational complexity $O(\log o)$; though it is not necessary because the other elements of the whole algorithm have linear complexity $O(o)$.

Step 5 consisting in executing the *t-move* selected in the previous step can be made by the single (master) processor in constant time $O(1)$. Thus computational complexity of the whole parallel algorithm is $O(o)$. The algorithm needs to be executed on $O(o^2)$-processor CREW PRAM and it is cost-optimal with the cost $O(o^3)$.

3. Computational results

Proposed method of cost function value estimation was applied to the tabu search algorithm for flexible job shop problem [2]. Algorithms were tested on the set of benchmark problem instances taken from Hurink [8] and ran on the HP xw4600 server with Intel Core 2 Duo 3.16GHz processor with nVidia GeForce GTX480 GPU working under Linux Fedora 12 operating system. In the Table 1 particular columns mean:

- Flex. – average number of equivalent machines per operation,
- TS1 – tabu search algorithm in which exact value of cost function is calculated,
- TS2 – tabu search algorithm in which value of cost function is estimated,

			t [ms]		C_{max}		
problem	$n \times m$	Flex.	TS1	TS2	TS1	TS2	Av. dev. [%]
abz5	10×10	2	0.1125	0.0139	1146	1181	3.05
abz6	10×10	2	0.1581	0.0198	933	952	2.04
abz7	20×15	2	0.4001	0.0201	637	641	0.63
abz8	20×15	2	0.3221	0.0158	635	640	0.79
abz9	20×15	2	0.3743	0.0169	647	656	1.39
average		**2**	**0.2734**	**0.0173**			**1.58**

Table 1: Experimental results of the proposed approach for Hurink [8] instances.

- t – time of cost function calculation (estimation) for all solutions from neighborhood generated with t-moves,
- C_{max} – value of cost function (makespan),
- Av. dev. – average percentage relative deviations of TS2 results to TS1 results.

Obtained results show that using method of cost function value estimation results in shorter calculation time. There is no difference in time of estimate cost function value for different problem size. Computational complexity of cost function value estimation for flexible job shop problem is equal $O(1)$. TS2 algorithm is 15.8 times faster than TS1 algorithm, the quality of obtained solutions is only slightly worse (1.58% for all tested instances). Computational experiments shows that proposed approach is more useful for problems with largest number of machines and larger value of average number of equivalent machines per operation.

4. Remarks and conclusions

A single-walk parallel approach to the flexible job shop scheduling has been presented in this paper. We show the new integrated approach to the neighborhood structure design and to its searching methodology from the point of view of the efficient multi-thread computing environment usage. A theoretical analysis based on PRAM model of parallel computing was also made. We proposed a cost-optimal method of the neighborhood generation parallelization for the CREW PRAM parallel computing model. The workload parallel determination algorithm decreases the computations time from $O(o^3)$ (of the sequential approach) to $O(o)$ time, using $O(o^2)$ processors. Applying PRAM computing model makes it possible to convert the proposed methods to GPU environment easily.

References

[1] Bożejko W., A new class of parallel scheduling algorithms, Wrocław Univeristy of Technology Publishing House (2010), 1–280.
[2] Bożejko W., Uchroński M., Wodecki M., The new golf neighborhood for the flexible job shop problem, Proceedings of the ICCS 2010, Procedia Computer Science 1 (2010), Elsevier, 289–296.
[3] Bożejko W., Uchroński M., Wodecki M., Parallel Meta^{2}2heuristics for the Flexible Job Shop Problem, in: L. Rutkowski et al. (Eds.), Proceedings of the ICAISC 2010, Lecture Notes in Artificial Intellignece No. 6114 (2010), Springer, 395–402.
[4] Brandimarte P., Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research 41 (1993), 157–183.
[5] Dauzère-Pérès S., Pauli J., An integrated approach for modeling and solving the general multiprocessor job shop scheduling problem using tabu search, Annals of Operations Research 70(3), (1997), 281–306.
[6] Gao J., Sun L., Gen M., A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, Computers & Operations Research 35 (2008), 2892–2907.
[7] Ho N.B., Tay J.C., GENACE: an efficient cultural algorithm for solving the Flexible Job-Shop Problem, IEEE International Conference on Robotics and Automation (2004), 1759–1766.
[8] Hurink E., Jurisch B., Thole M., Tabu search for the job shop scheduling problem with multi-purpose machine, OR Spektrum 15 (1994), 205–215.
[9] Mastrolilli M., Gambardella L.M., Effective neighborhood functions for the flexible job shop problem, Journal of Scheduling 3(1), (2000), 3–20.
[10] Nowicki E., Smutnicki C., The flow shop with parallel machines: A tabu search approach, European Journal of Operational Research 106 (1998), 226–253.
[11] Pauli J., A hierarchical approach for the FMS schduling problem, European Journal of Operational Research 86(1), (1995), 32–42.
[12] Pinedo M., Scheduling: theory, algorithms and systems, Englewood Cliffs, NJ: Prentice-Hall (2002).

```
Algorithm 1. ParallelNewPar
Input: Θ = (Q, π) - a feasible solution of the FJSP;
Output: Θ' = (Q', π') - a feasible solution generated by the t-move;
  Step 1: {Graph creation}
      if proc_id = 1 then
        Determine a graph with weighted vertices
            G(Θ) = (V, R ∪ E(Θ))
        connected with the solution Θ;
      parfor proc_id = 1..o^2 do
        Parallel determine the longest paths lengths between vertices
        of the graph G(Θ);
      end parfor;
  Step 2: {Blocks determination}
      if proc_id = 1 then
        Determine the critical path in G(Θ)
            (i.e., vertices sequence C(s, c));
      parfor proc_id = 1..o do
        Parallel determine blocks sequence B = (B^1, B^2, ..., B^r)
        of the critical path C(s, c));
      end parfor;
  Step 3: {Neighborhood determination}
      parfor k:=1 to r do        {consecutive blocks consideration}
        if (block operations B^k = (π(a^k), π(a^k + 1), ..., π(b^k))
            are executed on the machine M_v from the nest M^u)
        then                                 {machines of the nest M^u}
            parfor i := t_{u-1} + 1 to t_{u-1} + m_u do
                if i ≠ v then
                begin
                  determine feasible positions η_j(a^k) and ρ_j(a^k)
                  for the operation a^k on the machine M_i and
                  calculate expressions value Δ^{a^k}_{η_j(a^k)} and Δ^{a^k}_{ρ_j(a^k)};
                  determine feasible positions η_j(b^k) and ρ_j(b^k)
                  for the operation b^k on the machine M_i and
                  calculate expressions value Δ^{b^k}_{η_j(b^k)} and Δ^{b^k}_{ρ_j(b^k)};
                end;
  Step 4: {The best move determination}
      parfor proc_id = 1..o^2 do
        Parallel Determine the minimal value
        Δ^v_{χ(v)} = min_{1≤k≤r} min{Δ^z_{μ(z)} : z ∈ {a^k, b^k},
            μ(z) ∈ {η_j(z), ρ_j(z)}}
        connected with the best t-move t^v_{χ(v)} consisting in
        moving the first or the last operation, respectively,
        from some block to another machine from the same nest;
  Step 5: {The new assignment determination}
      if proc_id = 1 then
        Determine the new machine workload Q'
        connected with the solution Θ' generated by the t-move t^v_{χ(v)}
end.
```

Figure 1: Outline of the ParallelNewPar algorithm.