Parallel Neighborhood Search for the Permutation Flow Shop Scheduling Problem

Wojciech Bożejko, Jarosław Rudy, and Mieczysław Wodecki

Abstract In this chapter we consider the classic flow shop problem of task scheduling, which is a representative problem for a larger group of problems in which the solution is represented by permutation, such as Traveling Salesman Problem, Quadratic Assignment Problem, etc. We consider the most expensive (time-consuming) part of the local search algorithms for this class, which is search of the neighborhood of a given solution. We propose a number of methods to effectively find the best element of the neighborhood using parallel computing for three well-known neighborhoods: Adjacent Pair Interchange, Insert and Non-adjacent Pair Interchange. The methods are formulated as theorems for the PRAM model of parallel computation. Some of the methods are cost-optimal.

Keywords Flow Shop scheduling · Discrete optimization · Parallel computing · Local search

W. Bożejko
Department of Control Systems and Mechatronics, Faculty of Electronics,
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: wojciech.bozejko@pwr.edu.pl

J. Rudy (✉)
Department of Computer Engineering, Faculty of Electronics,
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: jaroslaw.rudy@pwr.edu.pl

M. Wodecki
Telecommunications and Teleinformatics Department, Faculty of Electronics,
Wrocław University of Science and Technology, 27 Wybrzeże Wyspiańskiego St., 50-372 Wrocław, Poland
e-mail: mieczyslaw.wodecki@pwr.edu.p

G. Bocewicz et al. (eds.), *Performance Evaluation Models for Distributed Service Networks*, Studies in Systems, Decision and Control 343,
https://doi.org/10.1007/978-3-030-67063-4_2

1 Introduction

The permutation flowshop problem with makespan criterion (here denoted as $F^*||C_{\max}$) is one of the oldest and most well-known, classic scheduling problems. Scientific papers related to this problem have been appearing for over 50 years now. Despite its simple formula and a finite set of solutions, the problem belongs to one of the hardest combinatorial optimization problem classes: the strongly NP-hard problems. Due to this, it is often used to test new ideas, properties and methods of construction of solving algorithms. Many papers concerning this problem have emerged in the literature, including research on fast, non-exact solving algorithms based on iterative improvement of solutions. A considerable advancement in development of such metaheuristic algorithms was possible thanks to the use of blocks (see e.g. Nowicki and Smutnicki [13]). For more details on research and results for this problem in recent years consider our previous works [8, 11]. Classification of scheduling problems is proposed in the work of Graham et al. [9]. Recently, there is a growing interest in bio-inspired, metaheuristic approaches, see [1, 7, 12, 14, 18].

For over a decade now, the increase of the number of cores in processors and processors in a computer system has become a standard for development of computer architectures. Such increase in computing power of parallel systems yields new possibilities, such as reduction of computation time, improved convergence capabilities and obtaining of better solutions. One of the first parallel algorithms for the flowshop problem was a Simulated Annealing method by Wodecki and Bożejko [16]. Parallel algorithms are popular in solving scheduling problems and all also well-suited for population-based metaheuristics (see e.g. [15]).

The key element of iterative improvement methods (including the best known metaheuristics) in combinatorial optimizations problems is the procedure for generation and evaluation (search) of the neighborhood of a solution. This procedure has crucial effect on computation time and quality of results. Thus, it is desirable to apply parallel computation techniques to this procedure (see e.g. [10]). In this chapter we consider several popular neighborhoods for the $F^*||C_{\max}$ problem and prove properties that can be used in parallel algorithms implemented on the PRAM (Parallel Random Access Machine, see e.g. [6]) parallel computation model. It should be noted that obtained results could be applied to similar permutation-based problems such as the Traveling Salesman Problem and Quadratic Assignment Problem. Further results in this area can be found in the works of Bocewicz [2], Bocewicz et al. [3, 4], and Wójcik and Pempera [17].

2 Permutation Flowshop Problem

Let $\mathscr{J} = \{1, 2, \ldots, n\}$ and $M = \{1, 2, \ldots, m\}$ be sets of n jobs and m machines respectively. Each job j is a sequence of m operations $O_{1j}, O_{2j}, \ldots, O_{mj}$. Operation O_{ij} has to be processed on machine i for time p_{ij} without interruption. Processing

of job j on machine $i > 1$ can only start when that job had finished processing on machine $i - 1$. A solution is a job processing schedule seen as matrices of job starting times $S = (S_1, S_2, \ldots, S_n)$, where $S_j = (S_{1j}, S_{2j}, \ldots, S_{mj})$ and completion times $C = (C_1, C_2, \ldots, C_n)$, where $C_j = (C_{1j}, C_{2j}, \ldots, C_{mj})$. In practice, only one matrix is needed to determine the schedule as $C_{ij} = S_{ij} + p_{ij}$.

For the makespan criterion the optimal schedule is always left-shifted. This allows us to represent the schedule using job processing order, which is an n-element permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ from the set Π of all possible permutations. Each permutation $\pi \in \Pi$ unequivocally determines the processing order of jobs on all machines (the same for each machine). In order to determine C_{ij} using π, we use the following recursive formula:

$$C_{i\pi(j)} = \max\{C_{i-1,\pi(j)}, C_{i,\pi(j-1)}\} + p_{i,\pi(j)},$$

$$i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n, \tag{1}$$

with initial conditions $C_{i,\pi(0)} = 0$, $i = 1, 2, \ldots, m$, $C_{0,\pi(j)} = 0$, $j = 1, 2, \ldots, n$, or a non-recursive one:

$$C_{i,\pi(j)} = \max_{1=j_0 \le j_1 \le \ldots \le j_i = j} \sum_{s=1}^{i} \sum_{j=j_{i-1}}^{j_i} p_{s,\pi(j)}. \tag{2}$$

Our goal is minimization of the makespan $C_{\max}$:

$$C_{\max} = \max_{j \in \pi, i \in M} C_{i,\pi(j)} = \max_{j \in \pi} C_{m,\pi(j)}, \tag{3}$$

thus we need to obtain permutation $\pi^* \in \Pi$ such that:

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi). \tag{4}$$

The values $C_{i\pi(j)}$ can be also determined using the graph model. For a given processing order π we construct a lattice graph $G(\pi) = (M \times N, F^0 \cup F^*)$, where $M = \{1, 2, \ldots, m\}$, $N = \{1, 2, \ldots, n\}$ are vertices,

$$F^0 = \bigcup_{s=1}^{m-1} \bigcup_{t=1}^{n} \{((s, t), (s + 1, t))\} \tag{5}$$

is a set of vertical arcs denoting technological order of processing of operations from a given job and

$$F^* = \bigcup_{s=1}^{m} \bigcup_{t=1}^{n-1} \{((s, t), (s, t + 1))\} \tag{6}$$

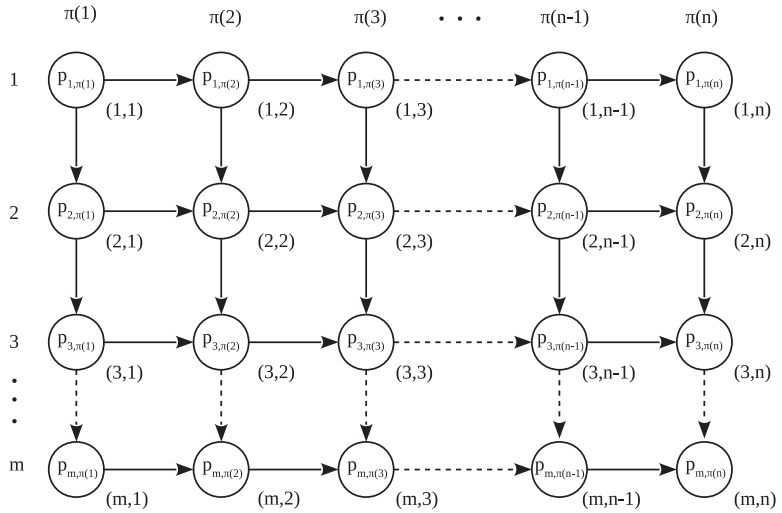

Fig. 1 Structure of a lattice graph $G(\pi)$

is the set of horizontal arcs denoting the job processing order π. The structure of graph $G(\pi)$ is shown in Fig. 1.

Arcs in graph $G(\pi)$ have no weights while the weight of vertex (s, t) is $p_{s,\pi(t)}$. Completion time $C_{i,\pi(j)}$ of job $\pi(j)$, on machine i is equal to the length of the longest path starting at vertex (1,1) and ending at vertex (i, j) including the weights of those vertices. For the $F^*||C_{max}$ problem the makespan $C_{\max}(\pi)$ is equal to the critical (longest) path in graph $G(\pi)$.

3 Adjacent Pair Interchange Neighborhood

The API (Adjacent Pair Interchange, called also "swap") neighborhood is one of the simplest and most commonly used. The speedup for sequential algorithms for the C_{max} goal function is realized by the so-called (sequential) accelerator (see for example [13]). Some of the theorems proven here are based on this accelerator, thus we describe it in details below.

Let π be a permutation from which the API neighborhood is generated and $v = (a, a + 1)$ be a pair of adjacent positions. Swapping those positions in π generates a neighboring solution $\pi_{(v)}$. For π we calculate:

$$r_{s,t} = \max\{r_{s-1,t}, r_{s,t-1} + p_{s,\pi(t)}\} \tag{7}$$

for $t = 1, 2, \ldots, a - 1$, $s = 1, 2, \ldots, m$, and

$$q_{s,t} = \max\{q_{s+1,t}, q_{s,t+1} + p_{s,\pi(t)}\} \tag{8}$$

for $t=a-1, a-2, \ldots, 1, s=m, m-1, \ldots, 1$, where $r_{0,t}=0=q_{j,t}, t=1,2,\ldots,n$, $r_{s,0}=0=q_{s,m+1}$, $s=1,2,\ldots,m$. Value $r_{s,t}$ is the length of the longest path in the lattice graph $G(\pi)$ described in Sect. 2 that ends at vertex (s,t), including the weight of that vertex, while $q_{s,t}$ is the length of the longest path starting at vertex (s,t), including its weight. The weight of vertex (s,t) is $p_{s,\pi(t)}$. With this, each value $C_{max}(\pi_{(v)})$ for an interchange of a single adjacent pair of jobs $v=(a,a+1)$ can be found in time $O(m)$ using the formula:

$$C_{\max}(\pi_{(v)}) = \max_{1\le s\le m}(d'_s + q_{s,a+2}), \tag{9}$$

where

$$d'_s = \max\{d'_{s-1}, d_s\} + p_{s,\pi(a)}, \quad s=1,2,\ldots,m, \tag{10}$$

is the length of the longest path ending at vertex $(s,a+1)$ in $G(\pi)$ and

$$d_s = \max\{d_{s-1}, r_{s,a-1}\} + p_{s\pi(a+1)}, \quad s=1,2,\ldots,m \tag{11}$$

is the length of the longest path ending at vertex (s,a) in graph $G(\pi_{(v)})$. The starting conditions are: $d'_0=d_0=0$, $r_{s0}=0=q_{s,n+2}$, $s=1,2,\ldots,m$. The API neighborhood contains $n-1$ solutions.

The goal of searching the API neighborhood for a permutation π is to find a permutation in $\pi_{(v)}$, $v=(a,a+1)$, $a=1,2,\ldots,n-1$ such that the goal function value is minimized. For the problem $F^*||C_{max}$ the complexity of that process is $O(n^2m)$, but can be reduced to $O(nm)$ using the accelerator described above.

In the proofs of theorems regarding computational complexity we will use the following commonly known parallel algorithms facts (See Cormen et al. [6]):

Fact 1 *Prefix sums of the input sequence can be determined on EREW (Exclusive Read Exclusive Write) PRAM in the time $O(\log n)$ with using $O(n/\log n)$ processors.*

Fact 2 *Minimal and maximal value of the input sequence can be determined on EREW PRAM in the time $O(\log n)$ with using $O(n/\log n)$ processors.*

Fact 3 *Values $y=(y_1,y_2,\ldots,y_n)$ where $y_i=f(x_i)$, $x=(x_1,x_2,\ldots,x_n)$ can be determined on CREW (Concurrent Read Exclusive Write) PRAM in time $O(\log n)$ on $O(n/\log n)$ processors.*

Theorem 1 *The API neighborhood for the $F^*||C_{max}$ problem can be searched in time $O(n+m)$ on the CREW PRAM using $O(\frac{n^2m}{n+m})$ processors.*

Proof Disregarding relations between solutions, we assign $O(\frac{nm}{n+m})$ processors to each solution in the neighborhood. This allows to compute the goal function for a single solution in time $O(n+m)$ (see Bożejko [5]). Next, we need to choose the minimal value among the $n-1$ computed ones. This can be done in time $O(\log n)$ using $O(n/\log n)$ processors. The overall time complexity is still $O(n+m)$ and the number of processors used is $(n-1)O(\frac{nm}{n+m}) = O(\frac{n^2m}{n+m})$. ■

The described method for the $F^*||C_{max}$ problem is not cost-optimal as its efficiency drops quickly as n grows. Let us note that the algorithm from Theorem 1, for the $F^*||\sum C_i$ problem and the API neighborhood is cost-optimal with efficiency $O(1)$ as the accelerator is not applicable for this problem.

Next we will make use of the relations between solutions in the neighborhood and the accelerator to obtain a considerably stronger result.

Theorem 2 *The search of the API neighborhood for the $F^*||C_{max}$ problem can be done in time $O(n+m)$ on the CREW PRAM using $O(\frac{nm}{n+m})$ processors.*

Proof Let $v = (a, a+1)$ be a pair of parallel positions. We will now employ the API accelerator in this parallel algorithm. The values $r_{s,t}, q_{s,t}$ are generated once, at the start of the API neighborhood search process in time $O(n+m)$ on the PRAM model using $O(\frac{nm}{n+m})$ processors. That method is cost-optimal.

Now, the API neighborhood search process can be divided into groups, with $\left\lceil \frac{n}{p} \right\rceil$ position swaps in each group, where $p = \lceil \frac{nm}{n+m} \rceil$ is the number of processors used. The goal function calculations are independent in each group. Each processor $k = 1, 2, \ldots, p$ will search part of the neighborhood that is obtained with moves v in the form of:

$$v = ((k-1)\left\lceil \frac{n}{p} \right\rceil + a, (k-1)\left\lceil \frac{n}{p} \right\rceil + a + 1),$$

where $a = 1, 2, \ldots, \left\lceil \frac{n}{p} \right\rceil$ for $k = 1, 2, \ldots, p-1$, and moves v in the form:

$$v = ((p-1)\left\lceil \frac{n}{p} \right\rceil + a, (p-1)\left\lceil \frac{n}{p} \right\rceil + a + 1),$$

where $a = 1, 2, \ldots, n-(p-1)\left\lceil \frac{n}{p} \right\rceil - 1$ for $k = p$. The last group might be smaller than the others. Because the process of determining all values $C_{max}(\pi_{(v)})$ in a single group has complexity $\left\lceil \frac{n}{p} \right\rceil O(m) = O(\frac{nm}{p}) = O(\frac{nm}{\frac{nm}{n+m}}) = O(n+m)$, thus the complexity of determining all values $C_{max}(\pi_{(v)})$ for all moves v will be the same. Each processor, sequentially calculating its portion of values $C_{max}(\pi_{(v)})$, can store the best value. To that end, a number of comparisons equal to the group size minus 1 has to be made, meaning: $\left\lceil \frac{n}{p} \right\rceil - 1 = O(\frac{n}{p}) = O(\frac{n}{\frac{nm}{n+m}}) = O(\frac{n+m}{m})$, which keeps the complexity of the entire method at $O(n+m)$. In order to determine the best move from the entire neighborhood we need to find the minimal element among p best goal function values stored for each group. This can be done in time $O(\log n)$ using $p = O(\frac{nm}{n+m})$ processors due to Fact 2. The complexity $O(\log p)$ of this stage, through the following sequence of inequalities:

$$\log p = \log \left\lceil \frac{nm}{n+m} \right\rceil < \log(\frac{nm}{n+m} + 1) =$$

$$= \log(\frac{nm+n+m}{n+m}) = \log(\frac{(n+1)(m+1)-1}{n+m}) =$$

$$= (\log((n+1)(m+1)-1) - \log(n+m) < \log((n+1)(m+1)) =$$

$$= \log(n+1) + \log(m+1) < n+1+m+1 \tag{12}$$

does not increase the complexity $O(n+m)$ of the entire method. ■

The method is cost-optimal.

Below we present a different method of searching the API neighborhood for the $F^*||C_{max}$ problem, running in shorter time, namely $O(\log(n+m)\log(nm))$. However, this is at the cost of increased number of processors.

Theorem 3 *The API neighborhood for $F^*||C_{max}$ can be searched on the CREW PRAM in time $O(\log(n+m)(\log(nm)))$ using $O(n^3m^3/\log(nm))$ processors.*

Proof We employ the lattice graph $G(\pi)$ from Sect. 2 and the API accelerator with modified calculation scheme. The values $r_{s,t}$, $q_{s,t}$ representing the lengths of the longest paths respectively ending and starting at vertex (s,t) can be determined in time $O(\log(n+m)(\log(nm)))$ on the PRAM with $O(n^3m^3 / \log(nm))$ processors by employing the method of determining all longest paths between pairs of vertices. From the properties of graph $G(\pi)$ we know that the longest path ending at vertex (s,t) starts at vertex (1,1), while the longest path starting at (s,t) ends at (n,m). Thus, after determining the array of longest paths A, it can be used directly access values $r_{s,t}$, $q_{s,t}$ for each vertex (s,t), $s = 1,2,\ldots,m$, $t = 1,2,\ldots,n$. Next, we assign $O(m^2/\log m)$ processors to each of the $n-1$ solutions from the API neighborhood and we calculate the goal function value for a single solution obtained by move $v = (a, a+1)$ in time $O(\log m)$ using the formulas (9)–(11). This process can be described as follows. We write down (11) as:

$$d_s = \max\{r_{s,a-1} + p_{s,\pi(a+1)},\ r_{s-1,a-1} + p_{s-1,\pi(a+1)} + p_{s,\pi(a+1)}, \ldots$$

$$\ldots, r_{1,a-1} + p_{1,\pi(a+1)} + p_{2\pi(a+1)} + \cdots + p_{s\pi(a+1)}\} =$$

$$\max_{1\le k\le s}(r_{k,a-1} + \sum_{t=k}^{s} p_{t,\pi(a+1)}) = \max_{1\le k\le s}(r_{k,a-1} + P^s_{k,\pi(a+1)}), \tag{13}$$

where $P^s_{k,j} = \sum_{t=k}^{s} p_{t,j}$, $k = 1,2,\ldots,s$ are prefix sums, which can be calculated for a given s in time $O(\log m)$ using $O(m/\log m)$ processors in accordance with Fact 1. We need the values $P^s_{k,j}$ for all $s = 1,2,\ldots,m$ and those can be determined in parallel using $O(m^2/\log m)$ processors. After that we will have access to values $P^s_{k,j}$ for each $s = 1,2,\ldots,m$ i $k = 1,2,\ldots,s$ for a given job j = $\pi(a+1)$. There is no more than m sums $r_{k,a-1} + P^s_{k,\pi(a+1)}$ in the formula (13), thus they can be calculated in time $O(\log m)$ on $O(m/\log m)$ processors (see Fact 3). To sum it up, for each

$s = 1, 2, \ldots, m$ the value d_s can be calculated in time $O(\log m)$ on $O(m/\log m)$ processors. Thus, all such values are determined in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors. The same technique can be applied to formula (10):

$$d'_s = \max\{d'_{s-1}, d_s\} + p_{s,\pi(a)} =$$

$$\max\{d_s + p_{s\pi(a)}, d_{s-1} + p_{s-1,\pi(a)} + p_{s\pi(a)}, \ldots, d_1 + p_{1,\pi(a)} + p_{2,\pi(a)} + \cdots + p_{s\pi(a)}\} =$$

$$\max_{1\le k\le s}(d_k + \sum_{t=k}^{s} p_{t,\pi(a)}) = \max_{1\le k\le s}(d_k + P^s_{k,\pi(a)}). \tag{14}$$

Because all the required prefix sums $P^s_{k,\pi(a)}$ can be determined in time $O(\log m)$ using $O(m^2/\log m)$ processors and values d_s were determined earlier, the calculation of all values d'_s, $s = 1, 2, \ldots, m$ can be done in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors employing the rules for determining d_s. In result, we can determine $C_{\max}(\pi_{(v)}) = \max_{1\le s\le m}(d'_s + q_{s,a+2})$ (see the formula (10)) in time $O(\log m)$ using $O(m/\log m)$ processors. This operation consists on performing m additions $d'_s + q_{s,a+2}$, $s = 1, 2, \ldots, m$, which can be done in time $O(\log m)$ on $O(m/\log m)$ processors (see Fact 3). Next, we determine the value of (10), that is we calculate maximum from the m-element set, which can be done in time $O(\log m)$ using $O(m/\log m)$ processors (see Fact 2). Finally, using $(n-1)O(m^2/\log m) = O(nm^2/\log m)$ processors we can determine the value of the goal function for all solution of the API neighborhood in time $O(\log m)$. Next, we determine the solution with the minimal value of the goal function in time $O(\log n)$ using $n-1$ processors. The entire method requires

$$O(\max\{n-1, \frac{nm^2}{\log m}, \frac{n^3m^3}{\log(nm)}\}) = O(\frac{n^3m^3}{log(nm)}) \tag{15}$$

processors and has time complexity of

$$O(\max\{\log m, \log n, \log(n+m)(\log(nm))\}) = O(\log(n+m)\log(nm)). \tag{16}$$

Thus, we can search the API neighborhood in parallel with the same time complexity as determining the value of the goal function for a single solution. Theorem 2 is an example of cost-optimal algorithm for this neighborhood.

4 Insert Neighborhood

Direct search of the INS (Insert) neighborhood implies time complexity of $O(n^3m)$. For this neighborhood and C_{max} goal function, an accelerator is known (see [13]), which allows to search the neighborhood in time $O(n^2m)$ for the $F^*||C_{max}$ problem. In this section we will show stronger results for parallel algorithms.

Theorem 4 *The INS neighborhood for the $F^*||C_{max}$ problem can be searched in time $O(n+m)$ on CREW PRAM using $O(\frac{n^2m}{n+m})$ processors.*

Proof Let $v = (a, b)$, $a \neq b$ be a move that generates a solution in the INS neighborhood. The move consists in modifying permutation π by removing job $\pi(a)$ from it and inserting it back into π such so the job ends up on position p in the resulting permutation $\pi_{(v)}$. Let $r_{s,t}, q_{s,t}, s = 1, 2, \ldots, m, t = 1, 2, \ldots, n-1$, be values calculated from formulas (7) and (8) for $(n-1)$-element permutation obtained from π by removing job $\pi(a)$. For each position $a = 1, 2, \ldots, n$ the values $r_{s,t}$, $q_{s,t}$ can be determined in time $O(n+m)$ on a PRAM with $O(\frac{nm}{n+m})$ processors (see Bożejko [5]). By using $O(\frac{n^2m}{n+m})$ processors, this process can be completed in time $O(n+m)$ for all $(n-1)$-element permutations obtained from π by removal of job $\pi(a)$, $a = 1, 2, \ldots, n$. For any given a, the value $C_{\max}(\pi_{(v)})$ obtained by inserting job $\pi(a)$ into position $b = 1, 2, \ldots, n$, $b \neq a$ can be calculated using the formula (9) in time $O(m)$. We divide the process of determining the goal function for the neighborhood elements into $p = \left\lceil \frac{n^2m}{n+m} \right\rceil$ groups assigned to a single processor each. Employing the aforementioned property and the fact that the INS neighborhood contains $(n-1)^2 = O(n^2)$ solutions, the time complexity of determining all values $C_{\max}(\pi_{(v)})$ is $\left\lceil \frac{(n-1)^2}{p} \right\rceil O(m) = O(n+m)$. Next, we need to find the neighborhood element with minimal value of goal function, which can be done in time $O(\log(n^2)) = O(2\log n) = O(\log n)$ using n processors. The entire method has time complexity of $O(n+m+\log n) = O(n+m)$ and requires the use of $O(\frac{n^2m}{n+m})$ processors. ■

The proposed method is cost-optimal.

Below we present a different method of searching the INS neighborhood in the $F^*||C_{max}$ problem, which allows to reduce time complexity to $O(\log(n+m)(\log(nm)))$ at the cost of more processors.

Theorem 5 *The INS neighborhood for $F^*||C_{max}$ can be searched in time $O(m+\log(n+m)(log(nm)))$ on CREW PRAM using $O(n^3m^3/\log(nm))$ processors.*

Proof Let $G(\pi)$ be a graph defined in Sect. 2 for a solution π that generates the INS neighborhood. Let $r_{s,t}$, $q_{s,t}$, $s = 1, 2, \ldots, m$, $t = 1, 2, \ldots, n-1$, be values determined according to formulas (7) and (8) for $(n-1)$-element permutation obtained from π by removing job $\pi(a)$. Values $r_{s,t}$, $q_{s,t}$ representing the lengths of longest paths respectively ending and starting at vertex (s, t) can be determined in time $O(\log(n+m)(\log(nm)))$ using $O(n^3m^3/\log(nm))$ processors by employing the method of determining all longest paths between pairs of vertices. From the properties of the graph $G(\pi)$ it follows that the longest path ending at vertex (s, t) starts at (1,1) and longest path starting at (s, t) ends at (n, m). Thus, after calculating the array of longest paths $\max_{dist}$, the values $r_{s,t}$, $q_{s,t}$ for any vertex (s, t), $s = 1, 2, \ldots, m$, $t = 1, 2, \ldots, n$ can be directly accessed from it. Next, we assign to each one of $O(n^2)$ elements of the INS neighborhood a single processor. Thus, the goal function value:

$$C_{\max}(\pi_{(v)}) = \max_{1\le i\le m}(d_i + q_{i,b+1}), \tag{17}$$

where

$$d_i = \max\{r_{i,b}, d_{i-1}\} + p_{i\pi(a)},\ i = 1, 2, \ldots, m, \tag{18}$$

for a single neighborhood element corresponding to a move $v = (a,b)$, $a \neq b$ can be determined in time $O(m)$. Using $O(n^2)$ processors, we determine the value of the goal function for all neighborhood elements independently in time $O(m)$. Finally, we find the minimal of those goal function values in time $O(\log n^2) = O(2\log n) = O(\log n)$ using $O(n^2)$ processors. The entire method thus requires:

$$O(max\{n^2, n^3m^3/log(nm)\}) = O(n^3m^3/log(nm))$$

processors for the time complexity of

$$O(max\{m, \log(n+m)(\log(nm))\}) = O(m + log(n+m)(log\ (nm))). \qquad \blacksquare$$

The next theorem shows how the time complexity from Theorem 5 can be reduced from $O(m + \log(n + m(log(nm)))$ to $O(\log(n + m)(log(nm)))$, while maintaining the number of processors at $O(n^3m^3/\log(nm))$.

Theorem 6 *The INS neighborhood for $F^*||C_{max}$ can be searched on CREW PRAM in time $O(\log(n+m)(\log(nm)))$ using $O(n^3m^3/\log(nm))$ processors.*

Proof We proceed similarly to the previous theorem. With the help of formulas (17) and (18) we determine the value of $C_{\max}(\pi_{(v)})$ in parallel for each from $n(n-1)$ neighborhood elements. Earlier, in time $O(\log(n+m)(\log nm))$ we determine values $r_{s,t}, q_{s,t}, s = 1, 2, \ldots, m, t = 1, 2, \ldots, n-1$ which are calculated based on (10) for a $(n-1)$-element permutation obtained from π by removing job $\pi(a)$. This is done using $O(n^3m^3/\log(nm))$ processors. Next in order to compute each of the $n(n-1)$ neighborhood elements we assign $O(m^2/\log m)$ processors and transform formula (18) as follows:

$$d_i = \max\{r_{i,b}, d_{i-1}\} + p_{i,\pi(a)} =$$

$$= \max_{1\le k\le i}(r_{k,b} + \sum_{t=k}^{i} p_{t,\pi(a)}) = \max_{1\le k\le i}(r_{k,b} + P^i_{k,\pi(a)}), \tag{19}$$

where $P^i_{k,j} = \sum_{t=k}^{i} p_{t,j},\ k = 1, 2, \ldots, i$ are prefix sums that can (Fact 1) be determined in time $O(\log m)$ with $O(m/\log m)$ processors for a given i. Since we need $P^i_{k,j}$ for all $i = 1, 2, \ldots, m$, thus they can be calculated in parallel during preliminary stage using $O(m^2/\log m)$ processors (number used before to check single neighborhood element). There is at most m sums $r_{k,b} + P^i_{k,\pi(a)}$,$k = 1, 2, \ldots, i$ in

formula (19), thus they can be determined in time $O(\log m)$ using $O(m/\log m)$ processors (Fact 3). Therefore, for each $i = 1, 2, \ldots, m$ we can obtain value d_i in time $O(\log m)$ on $O(m/\log m)$ processors. All such values can be determined in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors. Next, in order to calculate $C_{\max}(\pi_{(v)}) = \max\limits_{1\le i\le m}(d_i + q_{i,b+1})$ we perform m parallel additions $d_i + q_{i,b+1}$, $i = 1, 2, \ldots, m$ and we calculate the maximum from m-element set, using $O(m/\log m)$ processors and time $O(\log m)$ (see Facts 3 and 2). Thus, using $n(n-1)O(m^2/\log m) = O(n^2m^2/\log m)$ processors we can determine the value of goal function for all neighborhood elements in time $O(\log m)$. Next, we find element with minimal foal function value in time $O(\log n^2) = O(2\log n) = O(\log n)$ using n^2 processors. The entire method uses

$$O(\max\{n^2, \frac{n^2m^2}{\log m}, \frac{n^3m^3}{\log(nm)}\}) = O(\frac{n^3m^3}{\log(nm)})$$

processors with time complexity

$$O(\max\{\log m, \log n, \log(n+m)(\log(nm))\}) = O(\log(n+m)\log(nm)).$$ ■

We conclude that the INS neighborhood can be searched in the same time complexity as determining the goal function value for a single neighborhood element. There also exists a cost-optimal algorithm (Theorem 4).

5 Non-adjacent Pair Interchange Neighborhood

We start with the description of a sequential accelerator for the NPI (Non-adjacent Pair Interchange) neighborhood which we will use in this section. The accelerator has time complexity $O(n^2m)$ compared to complexity $O(n^3m)$ of a direct approach without the use of the accelerator .

Let v= (a,b), $a \neq b$ be a pair of jobs ($\pi(a)$, $\pi(b)$) that we can swap to obtain permutation $\pi_{(v)}$. Without the loss in generality we can assume $a < b$. Let $r_{s,t}$, $q_{s,t}$, $s = 1, 2, \ldots, m$, $t = 1, 2, \ldots, n$ be values calculated from (8) for n-element permutation π. Let $D^{x,y}_{s,t}$ denote the length of the longest path between vertices (s,t) and (x,y) in grid graph $G(\pi)$. Then the calculation of $C_{\max}(\pi_{(v)})$ can be expressed as follows. First, we determine the length of the longest path ending at (s, a), including job $\pi(b)$ due to swap of job v on position a:

$$d_s = \max\{d_{s-1}, r_{s,a-1}\} + p_{s,\pi(b)},\ s = 1, 2, \ldots, m, \tag{20}$$

where $d_0 = 0$. Next we calculate the length of the longest path ending at $(s, b-1)$, including the fragment of the graph between jobs on positions from $a+1$ to $b-1$

inclusive, which is invariant in regards to $G(\pi)$:

$$d'_s = \max_{1\leq w\leq s}(d_w + D^{s,b-1}_{w,a+1}),\ s = 1,2,\ldots,m. \tag{21}$$

Next we calculate the length of the longest path ending at vertex (s, b), including job $\pi(a)$ swapped on position b:

$$d''_s = \max\{d''_{s-1}, d'_s\} + p_{s,\pi(a)},\ s = 1,2,\ldots,m, \tag{22}$$

where $d''_0 = 0$. Finally, we obtain:

$$C_{\max}(\pi_{(v)}) = \max_{1\leq s\leq m}(d''_s + q_{s,b+1}). \tag{23}$$

Calculation of $C_{\max}(\pi_{(v)})$ is possible provided we have appropriate values $D^{x,y}_{s,t}$. Those can be calculated recursively for a given t and $y = t+1, t\ +\ 2,\ldots,n$, using the following formula:

$$D^{x,y+1}_{s,t} = \max_{s\leq k\leq x}(D^{ky}_{s,t} + \sum_{i=k}^{x} p_{i\pi(y+1)}), \tag{24}$$

where $D^{xt}_{s,t} = \sum_{i=s}^{x} p_{i\pi(t)}$. Alternatively, we can state this formula as:

$$D^{s,t+1}_{s,t} = D^{s,t}_{s,t} + p_{s,\pi(t+1)},\quad D^{x,0}_{s,t} = D^{0,y}_{s,t} = 0, \tag{25}$$

$$D^{x,y+1}_{s,t} = \max\{D^{x,y}_{s,t}, D^{x-1,y}_{s,t}\} + p_{x,\pi(y+1)}, \tag{26}$$

$x = 1,2,\ldots,m,\ y = 1,2,\ldots,n$, which allows, for a given (s,t), to determine all $D^{x,y}_{s,t}$, $x = 1,2,\ldots,m,\ y = 1,2,\ldots,n$ in time $O(nm)$. Finally, we sequentially determine all $O(n^2)$ values $C_{\max}(\pi_{(v)})$ and, before that, calculation of $D^{x,y}_{s,t}$, $x, s = 1,2,\ldots,m, y, t = 1,2,\ldots,n$ can be done sequentially in time $O(n^2m^2)$ (see [13]).

Theorem 7 *The NPI neighborhood for the $F^*||C_{max}$ problem can be searched in time $O(nm)$ on a CREW PRAM using $O(n^2m)$ processors.*

Proof We will now describe the counterpart to the sequential accelerator. Let us assign $O(m)$ processors to each of $\frac{n(n-1)}{2}$ neighborhood elements. For a given (s, t) value $D^{x,y}_{s,t}$ and all $x = 1,2,\ldots,m,\ y = 1,2,\ldots,n$ can be determined sequentially in time $O(nm)$. By using $O(nm)$ processors we can determine $D^{x,y}_{s,t}$ for all $x, s = 1,2,\ldots,m, y,\quad t = 1,2,\ldots,n$ in time $O(nm)$ and do that only once, during preliminary stage, for all neighborhood elements. Let us focus on determining value $C_{\max}(\pi_{(v)})$ for a given neighborhood element associated with some move v. We determine value d_s in (20) sequentially in time $O(m)$. Calculation of maximum of m values from (21) can be done in parallel for all s using $O(m)$ processors in time $O(m)$.

As for (22), we compute it sequentially for each s in time $O(m)$. The determining of value $C_{\max}(\pi_{(v)})$ in (23) is equivalent to independently performing m additions and then computing the maximum of those m values. We can do that sequentially in time $O(m)$. To sum it up, parallel computation of all $O(n^2)$ values $C_{\max}(\pi_{(v)})$ can be done in time $O(m)$ using $O(n^2m)$ processors. However, because the process of generating $D_{s,t}^{x,y}$ had time complexity $O(nm)$, this becomes the complexity of the entire method. ■

The following theorems strengthens the above result, either obtaining better time complexity or cost-optimality of the method.

Theorem 8 *The NPI neighborhood for the $F^*||C_{max}$ can be searched in time $O(m + \log(n+m)(\log(nm)))$ on CREW PRAM using $O(n^3m^3/\log(nm))$.*

Proof The proof is similar to previous theorem, except values $D_{s,t}^{x,y}$, are determined using $O(n^3m^3/\log(nm))$ processors in time $O(\log(n+m)(\log nm))$ (see [5]). With $D_{s,t}^{x,y}$ we can compute each $C_{\max}(\pi_{(v)})$ in time $O(m)$ using $O(m)$ processors. Thus, assigning in this stage $O(n^2m)$ processors, the time complexity of searching the NPI neighborhood is:

$$O(\max\{m, \log(n+m)\log(nm)\}) = O(m + \log(n+m)\log(nm)),$$

using $O(\max\{n^2m, n^3m^3/\log(nm)\}) = O(n^3m^3/\log(nm))$. processors. ■

Theorem 9 *The NPI neighborhood in the $F^*||C_{max}$ can be searched in time $O(n+m)$ on a CREW PRAM using $O(\frac{n^2m^2}{n+m})$ processors.*

Proof By employing recursive definition (26) for $D_{s,t}^{x,y}$, we can obtain $D_{s,t}^{x,y}$ for a given pair (s,t) and all $x = 1, 2, \ldots, m$, $y = 1, 2, \ldots, n$ in time $O(n+m)$ using $O(\frac{nm}{n+m})$ processors. By using nm times more processors, meaning $O(\frac{n^2m^2}{n+m})$, we can in parallel compute $D_{s,t}^{x,y}$, $x = 1, 2, \ldots, m$, $y = 1, 2, \ldots, n$ for all $s = 1, 2, \ldots, m$, $t = 1, 2, \ldots, n$ while keeping the time complexity at $O(n+m)$. Next we employ similar process as in the previous theorem, except the parallel determining of $C_{\max}(\pi_{(v)})$ for the neighborhood elements is split on $p = \left\lceil \frac{n^2m}{n+m} \right\rceil$ groups, each consisting of $O(m)$ processors. Thus the total number of processors is $O(pm) = \mathrm{O}(\frac{n^2m^2}{n+m})$. The process of determining a single $C_{\max}(\pi_{(v)})$ value for a given move v is performed like described in Theorem 8 in time $O(m)$ using $O(m)$ processors. Thus, the time complexity of determining all n^2 values $C_{\max}(\pi_{(v)})$, when computations are split on p independent groups (threads) is

$$\frac{n^2}{p}O(m) = O\left(\frac{n^2}{\left\lceil \frac{n^2m}{n+m} \right\rceil}m\right) = O(n+m) \tag{27}$$

and such will be the time complexity of the entire method. Because each of p threads had $O(m)$ processors, then the total number of processors is $O(pm) = \mathrm{O}(\frac{n^2m^2}{n+m})$. ■

The method is cost-optimal.

Theorem 10 *The NPI neighborhood for $F^*||C_{max}$ can be searched on a CREW PRAM in time $O(\log(n+m)(\log(nm)))$ using $O(n^3m^3/\log(nm))$ processors.*

Proof Let values $D_{s,t}^{x,y}$ be defined as in (24). The lengths of the longs paths between (s,t) and (x,y) in $G(\pi)$ can be determined using $O(n^3m^3 / \log(nm))$ processors in time $O(\log(n+m)(\log(nm)))$ (see [5]). Let us assign $O(m^2/\log m)$ processors to each of n^2 neighborhood elements. Let us focus on determining $C_{\max}(\pi_{(v)})$ for a single neighborhood element, obtained by some move v. Calculation of d_s in (20) can be done in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors. Prior to that we transform (20) into:

$$d_s = \max\{r_{s,a-1} + p_{s\pi(b)},\ r_{s-1,a-1} + p_{s-1,\pi(b)} + p_{s\pi(b)}, \ldots$$

$$\ldots, r_{1,a-1} + p_{1,\pi(b)} + p_{2\pi(b)} + \cdots + p_{s\pi(b)}\} =$$

$$= \max_{1\le k\le s}(r_{k,a-1} + \sum_{t=k}^{s} p_{t,\pi(b)}) = \max_{1\le k\le s}(r_{k,a-1} + P_{k,\pi(b)}^{s}), \tag{28}$$

where $P_{k,j}^{s} = \sum_{t=k}^{s} p_{t,j}$, $k = 1, 2, \ldots, s$ are prefix sums, which (according to Fact 1) can be calculated for a given s in time $O(\log m)$ using $O(m/\log m)$ processors. Because we need $P_{k,j}^{s}$ for all s $= 1, 2, \ldots, m$, then we can determine them in parallel using $O(m^2/\log m)$ processors. Afterwards, we will have access to values $P_{k,j}^{s}$ for all $s = 1, 2, \ldots, m$ and $k = 1, 2, \ldots, s$ assuming a given job j $= \pi(b)$. The is at most m sums $r_{k,a-1} + P_{k,\pi(b)}^{s}$ from formula (28), thus they can be determined in time $O(\log m)$ using $O(m/\log m)$ processors (Fact 3). Finally, for each $s = 1, 2, \ldots, m$ the value d_s can be determined in time $O(\log m)$ using $O(m/\log m)$ processors, thus all such values are determined in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors. The the determining the maximum of the m values in (21) can be done in parallel for all s using $O(m^2/\log m)$ processors in time $O(\log m)$. Prior to that we compute m sums $d_w + D_{w,a+1}^{s,b-1}$ in time $O(\log m)$ using $O(m/\log m)$ processors (Fact 3). We transform (22) into:

$$d''_s = \max\{d''_{s-1}, d'_s\} + p_{s,\pi(a)} =$$

$$= \max_{1\le k\le s}(d'_k + \sum_{t=k}^{s} p_{t,\pi(a)}) = \max_{1\le k\le s}(d'_k + P_{k,\pi(a)}^{s}).$$

Because we can determine all the necessary prefix sums $P_{k,\pi(a)}^{s}$ in time $O(\log m)$ using $O(m^2/\log m)$ processors and values d'_s are all already determined, thus determining of all values d''_s, $s = 1, 2, \ldots, m$ can be done in parallel in time $O(\log m)$ using $O(m^2/\log m)$ processors, as according to rules shown for com-

puting d'_s. Finally we determine $C_{\max}(\pi_{(v)}) = \max\limits_{1\le s\le m}(d'_s + q_{s,a+2})$ in time $O(\log m)$ using $O(m/\log m)$ processors. Such operation consists in performing m additions $d'_s + q_{s,a+2}$, $s = 1, 2, \ldots, m$, which can be done in time $O(\log m)$ on $O(m/\log m)$ processors (Fact 3). Next, for determining $C_{\max}(\pi_{(v)}) = \max\limits_{1\le s\le m}(d'_s + q_{s,a+2})$ we compute the maximum from an m-element set, in time $O(\log m)$ using $O(m/\log m)$ processors (Fact 2). Thus, in total using $(n^2)O(m^2/\log m) = O(n^2m^2/\log m)$ processors we can determine the goal function values for all neighborhood elements in time $O(\log m)$. Next, we need to find the element with the minimal goal function values, which can be done in time $O(\log n^2) = O(2\log n) = O(\log n)$ using n^2 processors. The entire method thus requires:

$$O(\max\{n^2, \frac{n^2m^2}{\log m}, \frac{n^3m^3}{\log(nm)}\}) = O(\frac{n^3m^3}{\log(nm)}) \tag{29}$$

processors and its time complexity is:

$$O(\max\{\log m, \log n, \log(n+m)(log(nm))\}) = O(\log(n+m)(\log(nm))).$$ ■

To sum it up, we can search the NPI neighborhood in parallel in the same asymptotic time as determining the goal function value for a single solution. Among described parallel algorithms there is a cost-optimal one (Theorem 9).

6 Conclusions

In the paper, we presented the results of our research on the parallelization of the most costly element of the local search algorithms that solves the permutation flow shop problem with makespan criterion, which is the generation and search of neighborhood. Three popular types of moves were considered: adjacent interchange (swap), any position swap, and insert type move. For each of those moves we have proposed effective (cost-optimal) methods as well as methods allowing to choose the best neighbor in the same time as evaluating a single solution.

Acknowledgements This work was partially funded by the National Science Centre of Poland, grant OPUS number 2017/25/B/ST7/02181.

References

1. Belabid, J., Aqil, S., Allali, K.: Solving permutation flow shop scheduling problem with sequence-independent setup time. In: Journal of Applied Mathematics (2020)
2. Bocewicz, G.: Robustness of multimodal transportation networks. Eksploatacja i Niezawodność—Maint. Reliab. **16**(2), 259–269 (2014)
3. Bocewicz, G., Wójcik, R., Banaszak, Z.A., Pawlewski, P.: Multimodal processes rescheduling: cyclic steady states space approach. In: Mathematical Problems in Engineering, 2013, art. no. 407096 (2013)

4. Bocewicz, G., Nielsen, P., Banaszak, Z., Thibbotuwawa, A.: Routing and scheduling of unmanned aerial vehicles subject to cyclic production flow constraints. Adv. Intell. Syst. Comput. **801**, 75–86 (2019)
5. Bożejko, W.: Solving the flow shop problem by parallel programming. J. Parallel Distrib. Comput. **69**(5), 470–481 (2009). https://doi.org/10.1016/j.jpdc.2009.01.009
6. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. McGraw-Hill Higher Education (2001)
7. Deb, S., Tian, Z., Fong, S., et al.: Solving permutation flow-shop scheduling problem by rhinoceros search algorithm. Soft Comput. **22**, 6025–6034 (2018)
8. Emmons, H., Vairaktarakis, G.: Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications. Springer Science & Business Media (2012)
9. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium, pp. 287–326. Elsevier (1979)
10. Jagiełło, S., Rudy, J., Żelazny, D.: Acceleration of Neighborhood Evaluation for a Multi-objective Vehicle Routing. Artificial Intelligence and Soft Computing. Springer International Publishing, pp. 202–213 (2015)
11. Komaki, G.M., Sheikh, S., Malakooti, B.: Flow shop scheduling problems with assembly operations: a review and new trends. Int. J. Prod. Res. **57**(10), 2929–2955 (2019). https://doi.org/10.1080/00207543.2018.1550269
12. Naderi, B., Ruiz, R.: The distributed permutation flow shop scheduling problem. Comput. Oper. Res. **37**(4), 754–768 (2010)
13. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the permutation flow-shop problem. Eur. J. Oper. Res. **91**(1), 160–175 (1996). https://doi.org/10.1016/0377-2217(95)00037-2
14. Potts, C., Strusevich, V.: Fifty years of scheduling: a survey of milestones. J. Oper. Res. Soc. **60**, S41–S68 (2009)
15. Rudy, J., Żelazny, D.: GACO: A Parallel Evolutionary Approach to Multi-objective Scheduling. Evolutionary Multi-Criterion Optimization. Springer International Publishing, pp. 307–320 (2015)
16. Wodecki, M., Bożejko, W.: Solving the Flow Shop Problem by Parallel Simulated Annealing. Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, pp. 236–244 (2002)
17. Wójcik, R., Pempera, J.: Designing cyclic schedules for streaming repetitive job-shop manufacturing systems with blocking and no-wait constraints. IFAC-PapersOnLine **52**(10), 73–78 (2019)
18. Xu, J., Yin, Y., Cheng, T.C.E., Wu, Ch-C, Gu, S.: An improved memetic algorithm based on a dynamic neighbourhood for the permutation flowshop scheduling problem. Int. J. Product. Res. **52**(4), 1188–1199 (2014)