Solving multi-objective permutation flowshop scheduling problem using CUDA

Dominik Żelazny
Department of Control Systems and Mechatronics
Wroclaw University of Technology, Poland
Email: dominik.zelazny@pwr.edu.pl

Jarosław Pempera
Department of Control Systems and Mechatronics
Wroclaw University of Technology, Poland
Email: jaroslaw.pempera@pwr.edu.pl

***Abstract*—In this paper we propose a parallel tabu search algorithm for the bi-criteria scheduling problem implemented on CUDA platform. The idea presented in this paper, refers to bi-criteria permutation flow shop case: minimization of total completion time (makespan) and total flow time. Proposed parallel Tabu Search algorithm uses multi-start with varying criteria weights in order to improve algorithms effectiveness. For the set of common benchmarks, proposed approach finds superior approximation of the Pareto front to other methods and obtains it in significantly shorter computation time compared to sequential methods.**

I. Introduction

The multiobjective flow shop is commonly considered as a test-bed for the use of advanced parallel, Nature's inspired algorithms recommended for applying in scheduling theory and practice. The problem is computationally expensive because of its strong NP-hardness and demands of analysis of huge variety of non-dominated solutions. Metahuristics inspired by Nature are able to provide set of solutions of accepted quality in a reasonable time. Parallel calculations offer additional benefits.

The permutation flow shop scheduling problem (PFSP), has been an object of investigations for decades. This widely studied problem from the scheduling theory links the simplicity of mathematical model, sophisticated solution algorithms with the moderate real world applications. The vast majority of scientific papers for the flow shop is devoted to single objective optimization, however most of modern production systems require multiobjective approaches for the use in decision making processes. Therefore, in recent years, many results concerning multiobjective scheduling have been published. Among used to this aim approaches, Pareto front (between dominated/nondominated solutions) is the most suitable for the decision making process. In practice, evolutionary algorithms are the most often used optimization tools for multiobjective optimization, because they provide in a natural way an approximation of the Pareto front.

Because of the NP-hard nature of the flow shop problem, efforts of designers of optimization algorithms were focused on different search methods running through the solution space. Highly effective optimization algorithms for single criterion flow shop problem are based on metaheuristics such as: tabu search [15], [10], simulated annealing [19], [5], ant colony system [22], harmonic search [20], swarm search [7]. Their efficiency significantly depends on the techniques for diversification and intensification of the search processes which influence on the calculation cost, which can be reduced using parallel computing environments.

Genetic Local Search Algorithm (GLSA) provided in [11], applied local search procedure to maximize fitness value for each individual generated by genetic operators. Hybrid algorithm integrating two-phase local search with Pareto local search was devised in [6]. The algorithm solved multiobjective PFSP with selected two among four listed criteria: a) makespan, b) total completion time, c) total weighted lateness and d) total flow time. Another complex hybrid method called multiobjective Particle Swarm Optimization (MOPS), was proposed in [17]. It operates on criteria flow and total tardiness. Elitist Tabu Search algorithm was used to initialize the swarm, while the local search procedure improves the solution represented by each particle. In paper [4] a gradual-priority weighting approach (GPW) was proposed. It sought Pareto optimal solutions by using genetic algorithms and genetic local search methods. They applied their approach, among others, to the flow shop problem with: a) bi-criteria function - minimizing the time of completion of the tasks and the total tardiness, and b) tri-criteria function - minimizing the time of completion of the tasks (makespan), the total tardiness and total flow time. Compared with different weighted approaches they demonstrated that the GPW approach may provide better results than the best known up to now. Pempera *et al.* proposed an interesting implementation of multi-objective simulated annealing algorithm for MOFSP in paper [16].

A review of multi-objective permutation flow shop scheduling problem (MOFSP) literature can be found in paper [21]. Yenisey and Yagmahan provide a brief literature review of the contributions to MOFSP.

The disadvantage of intensive local search is a long computation time. This can be eliminated by the use of parallel processing. In this paper we developed a new method of data aggregation for parallel computing value of $C_{\max}$ and C_{sum} for insert-type neighborhood in the flow shop scheduling problem.

II. The problem

The production task given by the set of jobs $J = \{1, \ldots, n\}$, has to be processed using the set of machines $M = \{1, \ldots, m\}$ in the order given by indices of machines. The processing of job j on machine k cannot be interrupted and requires time $p_{jk} > 0$, $j \in J$, $k \in M$. The order of processing jobs on each machine must be the same. So the processing order on each machine can be described by a permutation π on the set $\{1, \ldots, n\}$. The schedule for the

978-1-4799-8701-6/15/$31.00 ©2015 IEEE.

given π can be represented by job completion times C_{jk} of job j on machine k, satisfying the following constraints:

$$C_{jk} \geq p_{jk}, \;\; j = 1, \ldots, n, \; k = 1, \ldots, m, \tag{1}$$

$$C_{\pi(j),k} - p_{\pi(j),k} \geq C_{\pi(j),k-1}, \;\; j = 1, \ldots, n, \; k = 2, \ldots, m, \tag{2}$$

$$C_{\pi(j),k} - p_{\pi(j),k} \geq C_{\pi(j-1),k}, \;\; j = 2, \ldots, n, \; k = 1, \ldots, m, \tag{3}$$

The completion times C_{jk} can be determined using recursive formulae:

$$C_{\pi(j),k} = \max(C_{\pi(j-1),k}, C_{\pi(j),k-1}) + p_{\pi(j),k}, \quad j = 1, \ldots, n, \; k = 1, \ldots, m, \tag{4}$$

with $\pi(0) = 0$, $C_{j,0} = 0$ for $j = 1, \ldots, n$ and $C_{0,k} = 0$ for $k = 1, \ldots, m$. The schedule can be computed in time $O(nm)$. The optimization goal is to determine Pareto optimal set of permutations for the following criteria functions: (1) maximal completion time (makespan)

$$C_{\max}(\pi) = \max_{j=1,\ldots,n} \{C_{\pi(j),m}\}, \tag{5}$$

and (2) total flow time (sum of completion times)

$$C_{sum}(\pi) = \sum_{j=1}^{n} C_{\pi(j),m}. \tag{6}$$

III. Parallel approaches

Over the last years, researchers applied parallel computing to solve different combinatorial optimization problems, including scheduling problems. Parallel Tabu Search implementation presented in [2] provides a new fast methods of parallel calculations for the hybrid flow shop problem. A parallel cooperative meta-heuristic for bi-objective flow shop problem was proposed in paper [13]. Results confirmed that the use of grid computing allows to fully exploit parallel models and their combination for solving large-size problem instances. In paper [18] a multiobjective parallel genetic algorithm was shown. Evaluation was Pareto-based and used procedure Redirect, which helped the algorithm to overcome the local optima. Obtained results showed good quality of this algorithm. Solving Timetabling Problems on GPU (Graphics Processing Unit) was introduced in [3]. Authors applied a parallel tabu search algorithm to solve the general problem of timetabling. The problem of timetabling (also known as scheduling) was first expressed as a graph coloring problem and then good approximate solutions were obtained with use of concurrent metaheuristic algorithm for GPU. An interesting parallel coevolutionary algorithm for other discrete optimization problem was proposed by Bożejko *et al.* in paper [1]. Computational experiments were conducted in a neighbourhood of clusters and aimed to examine the impact of parallelization algorithm on the computation time and the quality of the obtained solutions.

IV. Neighbourhood

In the literature two type of neighborhoods for the permutation flow shop problem were considered: insert and interchange. For example, in paper [9] the comparative work on local search algorithms for scheduling problems was presented. Objective was minimization of the sum of weighted completion times of jobs in a permutation flow shop. Those two types of neighborhood structures were compared in computational experiments with multi-start descent. A comparison of heuristics for flowtime minimization, divided into simple and composite approaches, most of which used insertion neighborhood was presented in [8]. The best of simple heuristics were outperformed by all the composite heuristics. Results were described in more detail in the paper.

Formally, the neighborhood $N(\pi)$ of current solution π consists of the set solutions similar to π. The set $N(\pi)$ is generated by set of moves $V(\pi)$. Each move v from $V(\pi)$ generate new permutation π_v by small modification of π. The insert-type moves for permutation are defined as follows:

$$V(\pi) = \{(x, y) : x \neq y, \; x, y = 1, \ldots, n\}. \tag{7}$$

The solution π_v, $v = (x, y) \in V(\pi)$ takes for $x < y$ the following form

$$\pi_v = (\pi(1) \ldots, \pi(x-1), \pi(x+1), \ldots, \pi(y-1), \pi(x), \pi(y), \ldots, \pi(n)) \tag{8}$$

whereas for $x > y$ the form of

$$\pi_v = (\pi(1), \ldots, \pi(y-1), \pi(x), \pi(y), \ldots, \pi(x-1), \pi(x+1), \ldots, \pi(n)). \tag{9}$$

V. Parallel analysis of neighborhood

Generally, determining the value of the objective function takes the dominant amount of time in the algorithms based on local search methods. This is particularly important in algorithms which search the whole neighborhood such as descending search, tabu search etc. The computation time can be reduced due to parallel computing.

There exist two models of parallel computing realized in real-life computer systems. The first SIMD (Single Instruction stream, Multiple Data stream) embrace the vector processors which manipulate on vectors of simple type of data such as integers and floats. Beginning with the emergence of MMX Intrinsics, the vector processing instructions were embedded in processors commonly used in PCs. An advance vector processing is used in GPU computing. The second model MIMD (Multiple Instruction stream, Multiple Data stream) can be realized on supercomputers equipped with large number of independent processors (cores) with dedicated memory.

Implementation of parallel computing in MIMD model is obvious. Having n^2 processors, the insert neighborhood can be computed in $O(nm)$ time. Processing of parallel computing requires synchronization of processors. Unfortunately, long synchronization lags (several dozen of ms in fast computer networks) in comparison to short computing times (a few ms) reduce the efficiency. In the case of SIMD we propose a new method of parallel computing of the insert neighborhood. The proposed method can be implemented and executed on both of the mentioned devices of vectoring computing.

Without the loss of generality, we conduct description of the proposed method on the natural sequence of jobs $\pi = (1, \ldots, n)$. Let $\pi^{j,a}$ $j, a = 1, \ldots, n$, be the permutation formed from π by removing job $j \in J$ from the position j and placed it on the position a. The set of all such formed permutations

can be divided into 3 subsets with respect to the position a: (1) for $j < a$, (2) for $j = a$, and (3) for $j > a$. In the case 2) $\pi^{j,a} = \pi$. In the case (*1*) job j is removing from position j some jobs (including job a placed on position a) are shifted to the left by one position, whereas in the case (3) the proper jobs are shifted to the right.

For the n-elements permutation, the vector computations are realized in $n + 2$ steps for each machines $k \in M$ and lies on selective applying of formula (4) for each vectoring processor. For this purpose we extend each permutation $\pi^{j,a}$ with two blank elements, for these elements the formula (4) is not applied and the value of the completion time remains unchanged. For the permutation $\pi^{j,a}$, the first blank element is placed on position j if $a > j$ and $j + 2$ if $a < j$, whereas the second on the position $a + 2$ if $a > j$ and a if $a < j$.

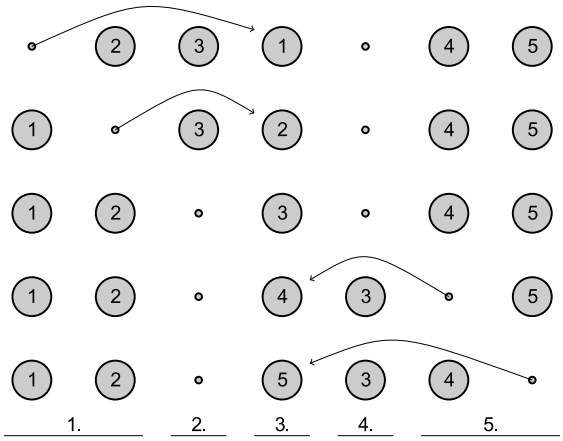

Fig. 1. Permutations for the insert position.

For the permutation $\pi = (1, \ldots, 5)$, the extended permutations for the insert position $a = 3$ are shown on the Figure 1. The blank elements are drawn as a small circles. The arrows illustrate the shifts of jobs. It is easy to see, that the position of the most of jobs in extended permutations are preserved.

Let $C_k = (C_k^1, C_k^2, \ldots, C_k^n)$ be a vector consists n elements. The element C_k^j corresponds to the permutation $\pi^{j,a}$ and for s-th tick of vectoring computation denote completion time of s-th job in extended permutations. For the vectoring processing the formula (4) takes the form:

$$C_k^j = \max(C_k^j, C_{k-1}^j) + p_{*,k}, \quad j = 1, \ldots, n, \ k = 1, \ldots, m, \tag{10}$$

where $p_{*,k}$ for the given j denote processing time of job placed on position s in permutation $\pi^{j,a}$. Note, that value C_k^j on the left side of equation denote C_k^j for $s - 1$ tick of parallel computing.

The pseudo code of the proposed method is presented in Figure 2. The keyword *parallel* highlights the selective vector computation. The code is divided into five phase. The first for $s = 1, \ldots, a - 1$ corresponds to the removing job from position 1 to $a - 1$, whereas the fifth for $s = a + 3, \ldots, n + 2$ corresponds to removing jobs from position $a + 1$ to n. The parallel computations for insertion of job on position a are realized in phase 3 for $s = a + 1$. This phase immediately precedes the phase of determining the completion time for job a, where $j > a$, and immediately after it follows the phase of determining the completion time for job a, where $j < a$.

```
1.      for s = 1 to a − 1 do
1.1.      for k = 1 to m do
1.1.1.      parallel for j do C_k^j = max{C_k^j, C_{k−1}^j} + p_{s,k} for j = s

2.      for k = 1 to m do
2.1.      parallel for j do C_k^j = max{C_k^j, C_{k−1}^j} + p_{a,k} for j < a

3.      for k = 1 to m do
3.1.      parallel for j do C_k^j = max{C_k^j, C_{k−1}^j} + p_{j,k}

4.      for k = 1 to m do
4.1.      parallel for j do C_k^j = max{C_k^j, C_{k−1}^j} + p_{a,k} for j > a

5.      for s = a + 1 to n do
5.1.      for k = 1 to m do
5.1.1.      parallel for j do C_k^j = max{C_k^j, C_{k−1}^j} + p_{s,k} for j = s
```

Fig. 2. Scheme of schedule determining.

Finally, the features of the proposed method are formulated as obvious theoretical properties.

Property 1: The values of $C_{\max}$ and C_{sum} of insertion on given position a for each of n can be found on vector machine with n processors in the time $O(nm)$.

Property 2: The values of $C_{\max}$ and C_{sum} for insert type of neighborhood can be found on n vector machines with n processors in the time $O(nm)$.

VI. Implementation

The CUDA parallel computing platform logically consists of a flexible number of advanced vector (stream) processors equipped with flexible number of cores. Each processor has independent fast shared read-write memory and fast shared read-only memory. Unfortunately, both of those types of memories are relatively small. The processors exchange the data with big size RAM memory by a wide bus.

The proposed method is easy to implement on CUDA platform for instance sizes up to $n = 100$ and $m = 20$, because:

- all processing times can be stored in read-only memory,
- the vectors C_k, $k = 0, ..., m$ can be stored in shared memory,
- the vector operations can be implemented as an atomic.

VII. Computational experiments

Two main goals of the computational experiment set up: (1) to determine the real speedup on parallel computing GPU device over traditional CPU computing, and (2) to evaluate the quality of Pareto front approximation. We use full insertion type neighborhood. Each solution generated by TS algorithm was evaluated by following formulae.

$$f(\pi) = \alpha \cdot C_{sum}(\pi) + (1 - \alpha) \cdot C_{\max}(\pi), \qquad (11)$$

where $0 \leq \alpha \leq 1$ is a direction of search control parameter. The α value changes from 0 to 1, directs the search from solutions near the $C_{\max}$ optimum, throughout intermediate solutions, to solutions near the C_{sum} optimum. Algorithm starts $rep = 10$ times with parameter α distributed uniformly from 0 to 1.

The tabu list contains pairs of jobs. Each pair (a, b) stored in the tabu list defines a precedence constraint and denotes that job a must be performed before job b in each unforbidden permutation. In each iteration of the algorithm the pair of jobs denoted as $(\pi(x), \pi(x+1))$ if $x < y$ or $(\pi(x-1), \pi(x))$ if $x > y$ is stored on tabu list, where $v^* = (x, y)$ and π_{v^*} is the best unforbidden neighbor of current solution π.

Algorithms were tested on 90 benchmark instances proposed in [14] and divided into following groups of $n \times m$: 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5 and 100×20. Each of those groups contains 10 instances of the same size.

Algorithms were implemented in C++ programming language and compiled with Microsoft Visual Studio 2010. Two versions of TS was tested which differ computer types and methods of neighborhood computing. In TS_{CPU}, we use sequential computing executed on host processor, whereas in TS_{GPU} we use parallel computing executed on GPU device.

Experiments on TS_{CPU} and TS_{GPU} algorithms were carried out on a computer equipped with (Intel® Core i7 CPU X 980 @ 3.33GHz) and (24GB of RAM) and (nVidia Tesla S2050 - 4GPUs inside, each equipped with 448 cores and 3072 MB of RAM).

A. Speedup test

We compare the execution times of TS_{CPU} and TS_{GPU} versions of the TS algorithm. Based on the computation times $(Time(A) : A \in \{TS_{CPU}, TS_{GPU}\})$ collected for all of the tested instances, we calculated the speedup ratio as a quotient $Time(TS_{CPU})/Time(TS_{GPU})$. Both algorithms searched through the same number of solutions during our tests.

From group 50×5 the GPU version of algorithm outperforms CPU version in execution times. The speedup ratio increases with the size of the instances i.e. both with the number of jobs and the number of machines. For the group of biggest size of the instance the GPU version works nearly 16 times faster than the algorithm tested on CPU. For the small instances the CPU version is faster than the GPU implementation.

TABLE I. Aggregation of Pareto solutions

Group	$\lvert P_{TS}\rvert/\lvert P\rvert$	$\lvert P_{bench}\rvert/\lvert P\rvert$	$\lvert D_{TS}\rvert/\lvert D\rvert$	$\lvert D_{bench}\rvert/\lvert D\rvert$
	%	%	%	%
20×5	86,08	63,29	36,71	13,92
20×10	83,18	87,73	12,27	16,82
20×20	84,75	84,75	15,25	15,25
50×5	79,67	20,33	79,67	20,33
50×10	37,74	62,26	37,74	62,26
50×20	24,53	75,47	24,53	75,47
100×5	78,33	21,67	78,33	21,67
100×10	57,08	42,92	57,08	42,92
100×20	38,02	61,98	38,02	61,98

TABLE II. Hyper–volume indicator values

Group	$I_H(TS)$	$I_H(bench)$	$I_H(TS)/I_H(bench)$
20×5	0,0470	0,0469	0,9976
20×10	0,0515	0,0517	1,0040
20×20	0,0496	0,0497	1,0026
50×5	0,0560	0,0547	0,9775
50×10	0,0534	0,0545	1,0210
50×20	0,0472	0,0501	1,0600
100×5	0,0439	0,0420	0,9574
100×10	0,0475	0,0472	0,9943
100×20	0,0453	0,0480	1,0589

B. Performance test

In order to evaluate the quality of the Pareto sets generated by the TS algorithm, for each test instance we determined: S_x – Pareto set generated by the an algorithm or taken from benchmarks, P – a set of Pareto points designated for the sum of all sets S_x, D – a set of unique Pareto points designated for the sum of all sets S_x, $P_x = S_x \cap P$ – a subset of the set P, and $D_x = P_x/P$ – a subset of unique Pareto solutions, where $x \in \{\text{TS}, \text{bench}\}$. This indicator allows the evaluation of the quality of the Pareto sets only in a quantitative manner. Unfortunately, they ignore dominated solutions located near Pareto approximation.

Hypervolume Indicator (I_H), proposed and studied in [12], [23], is a numerical representation of the area of the space between the reference solution and the approximation of the Pareto front of a given algorithm. The reference point remains constant for all the compared sets (algorithms) for given instance and is taken as the point with values of criteria corresponding to 120% of the worst values from all sets of solutions. This indicator is consistent with Pareto efficiency. For each instance set we evaluated: $I_H(TS)$ and $I_H(bench)$ respectively, for the sets S_{TS} and S_{bench}.

Tables I and II contain experimental results of the algorithm collected and compared with benchmark values. Comparison with available benchmark results was performed using data from sequential version of the algorithm. Computation time of the algorithm was set to be of exact or lower value than single run of one of the algorithms used to create benchmarks in [14].

In table I, the second column contains the average number of non-dominated solutions for the group of instances generated by the algorithm TS found in the set P divided by the number of solutions in set P, the third column is the same size for set from the benchmarks provided in [14]. In the

fourth column we placed the average number of unique Pareto solutions generated by TS algorithm divided by the number of solutions in set P. The corresponding values for the benchmark sets were collected in the fifth column. In table II, in the second and third columns were collected mean coefficient of I_H for sets: a) generated by algorithm TS and b) devised from benchmarks. While the fourth column contains the quotient of hyper-volume ratio of our results to benchmark values.

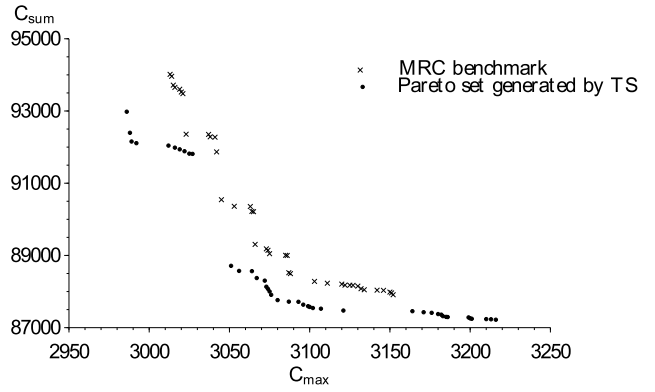

Fig. 3. Pareto fronts generated by TS and from benchmarks - TA45.

Figure 3 shows the Pareto fronts from benchmark and generated by the TS algorithm for instance TA45. This picture is typical for instances of sizes 50×10, 50×20 and 100×20. What is worth noting, for those groups of instances TS algorithm generates solutions with significantly lower values for both criterion functions than those from benchmarks.

Overall, both the number of the non-dominated solutions as well as the number of unique Pareto solutions were higher for proposed the TS algorithm. Out of all non-dominated solutions around 48% were unique solutions provided by TS. Moreover, I_H indicators of TS were up to 9% bigger than those evaluated from benchmarks, while an average increase of I_H was equal to 0,82%. Thus, TS proved to be superior than benchmark values. Benchmarks combined results from multiple testing of 16 different algorithms. In order to maintain similar test conditions, run times of the algorithm were limited to certain values. Proposed TS proved to be superior and were able to find new non-dominated solutions, which were omitted in earlier mentioned benchmarks. Additionally, for instance sizes 50×10, 50×20 and 100×20 number Pareto optimal solutions and unique non-dominated solutions was much better for TS algorithm. Moreover, for instance with the number of machines $m = 5$ TS algorithm obtained similar or slightly lower values of I_H.

VIII. Conclusions

Proposed algorithm proved to be efficient both in terms of speed and quality. Approximations of Pareto front were evaluated up to 16 times faster and found solution sets which had better values of Hyper-volume Indicator as well as more Pareto and unique non-dominated solutions. Overall performance was more than satisfactory and the parallelization of TS algorithm provided better approximations of Pareto front in lower computation time. Moreover, an original way to control the process of searching was proposed and allowed the algorithm to explore different portions of the Pareto front.

Multi-criteria optimization problems are mainly solved using evolutionary methods. Solely used local search methods, like Tabu Search and Simulated Annealing, were less effective in approximating Pareto-front due to theirs search scheme. We proposed a new way of approximating Pareto-front using control parameter α, which allowed us to lead search in all directions of the true Pareto front. This way, we were able to find more uniformly spread approximations of Pareto-front of all benchmark instances.

Albeit permutation flow shop problem is relatively simple, it originated from production engineering and was modeled from manufacturing systems and assembly lines. Methods developed for solving PFSP can be employed for wider class of combinatorial optimization problems, which exhibit similar characteristics. Additionally, multiple criteria usually translate into greater computational complexity and require more computing power. Proposed parallel method significantly reduced the computation time, while maintaining high quality of the results both in terms of I_H values and number of Pareto solutions.

Acknowledgment

This work is co-financed by the European Union as part of the European Social Fund, grant no. DG-G/3455/14, and the *Młoda Kadra*, grant no. B40112.

References

[1] W. Bożejko and Ł. Kacprzak and M. Wodecki, *Parallel Coevolutionary Algorithm for Three-Dimensional Bin Packing Problem*, Lecture Notes in Artificial Intelligence, vol. 9119, pp. 319–328, 2015.

[2] W. Bożejko and J. Pempera and C. Smutnicki, *Parallel tabu search algorithm for the hybrid flow shop problem*, Computers and Industrial Engineering, vol. 65 (3), pp. 466–474, 2013.

[3] W. Bożejko and Ł. Gniewkowski and M. Wodecki, *Solving timetabling problems on GPU.*, Lecture Notes in Computer Science, vol. 8468, pp. 445–455, 2014.

[4] P–C. Chang and J–C. Hsieh and S–G. Lin, *The development of gradual–priority weighting approach for the multi–objective flowshop scheduling problem*, International Journal of Production Economics, vol. 79 (3), pp. 171–183, 2002.

[5] L. Chinyao and Y. Jinn–Yi and H. Kai–I, *A robust simulated annealing heuristic for flow shop scheduling problems*, International Journal of Advanced Manufacturing Technology, vol. 23 (9–10), pp. 762–767, 2004.

[6] J. Dubois–Lacoste and M. López-Ibáñez and T. Stützle, *A hybrid TP+PLS algorithm for bi–objective flow–shop scheduling problems*, Computers and Operations Research, vol. 38, pp. 1219–1236, 2011.

[7] M. Fatih Tasgetiren and Y–C. Liang and M. Sevklic and G. Gencyilmaz, *A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem*, European Journal of Operational Research, vol. 177 (3), pp. 1930–1947, 2007.

[8] J. M. Framinan and R. Leisten and R. Ruiz-Usano, *Comparison of heuristics for flowtime minimization in permutation flowshops*, Computers and Operations Research, vol. 32 (5), pp. 1237–1254, 2005.

[9] C. A. Glass and C. N. Potts, *A comparison of local search methods for flow shop scheduling*, Annals of Operations Research, vol. 63 (4), pp. 489–509, 1996.

[10] J. Grabowski and J. Pempera, *New block properties for the permutation flow shop problem with application in tabu search*, Journal of the Operational Research Society, vol. 52 (2), pp. 210–220, 2001.

[11] H. Ishibuchi and T. Murata, *A multiobjective Genetic Local Search Algorithm and its Application to Flowshop Scheduling*, IEEE Transaction on Systems, Man and Cybernetics – Part C: Applications and Reviews, vol. 28 (3), 1998, 392–403.

[12] J. Knowles and L. Thiele and E. Zitzler, *A tutorial on the performance assessment of stochastic multiobjective optimizers*, Technical Report, ETH, Zurich, 2006.

[13] N. Melab and M. Mezmaz and E. G. Talbi, *Parallel cooperative metaheuristics on the computational grid. A case study: the bi-objective Flow-Shop problem*, Parallel Computing, vol. 32 (9), pp. 643–659, 2006.

[14] G. Minella and R. Ruiz and M. Ciavotta, *A Review and Evaluation of Multiobjective Algorithms for the Flowshop Scheduling Problem*, INFORMS Journal on Computing, vol. 20 (3), pp. 451–471, 2008.

[15] E. Nowicki and C. Smutnicki, *A fast tabu search algorithm for the permutation flow–shop problem*, European Journal of Operational Research, vol. 91 (1), pp. 160–175, 1996.

[16] Pempera J., Smutnicki C., Żelazny D., Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm, Procedia Computer Science, vol. 18, pp. 936–945, 2013.

[17] A. R. Rahimi–Vahed and S. M. Mirghorbani, *A multi–objective particle swarm for a flow shop scheduling problem*, Journal of Combinatorial Optimization, vol. 13, pp. 79–102, 2007.

[18] E. Rashidi and M. Jahandar and M. Zandieh, *An improved hybrid multiobjective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines*, International Journal of Advanced Manufacturing Technology, vol. 49 (9–12), pp. 1129–1139, 2010.

[19] I. H. Osman and C. N. Potts, *Simulated annealing for permutation flowshop scheduling*, Omega, vol. 17 (6), pp. 551–557, 1989.

[20] L. Wang and Q–K. Panb and M. Fatih Tasgetiren, *Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms*, Expert Systems with Applications, vol. 37 (12), pp. 7929–7936, 2010.

[21] M.M. Yenisey and B. Yagmahan, *Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends*, Omega, vol. 45, pp. 119–135, 2014.

[22] K–C. Ying and C–J. Liao, *An ant colony system for permutation flow–shop sequencing*, Computers and Operations Research, vol. 31 (5), pp. 791–801, 2004.

[23] E. Zitzler and L. Thiele, *Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach*, IEEE Transactions on Evolutionary Computation, vol. 3 (4), pp. 257–271, 1999.