Computers & Industrial Engineering 90 (2015) 212–220

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A new approach for multi-criteria scheduling

Czesław Smutnicki [a,*], Jarosław Pempera [b], Jarosław Rudy [b], Dominik Żelazny [b]

[a] Department of Computer Engineering, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland
[b] Department of Control Systems and Mechatronics, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland

ARTICLE INFO

Article history:
Received 18 December 2013
Received in revised form 8 February 2015
Accepted 4 September 2015
Available online 16 September 2015

Keywords:
Multi-criteria optimization
Flow shop scheduling
Pareto frontier
Simulated annealing
Vector processing

ABSTRACT

In this paper we propose a new approach, recommended for solving certain class of multi-criteria scheduling problems, with the use of cloud exploration of the solution space supported by fine-grained parallel computing. The approach allow us to approximate Pareto front more accurately than other known algorithms in competitive time. To show and check advantageous properties of the proposed approach, the new solution algorithm, called VESA, was implemented for the case of bi-criteria flow shop scheduling problem and tested against a number of high-quality benchmarks known in the literature. Vector processing technologies are used to enhance the efficiency of solution search and acceptance rates for the extended simulated annealing metaheuristic. Results are compared using, among others, the independent Hyper-Volume Indicator (I_H) measure. Computer test of VESA confirms excellent approximation of the Pareto front.

1. Introduction

Technological advancement is inseparably connected with higher customer expectations and the necessity of performance increase. Companies, in order to maintain their position in highly competitive market, are compelled to use more advanced systems for production planning. Optimization of production scheduling remains a difficult task that relies on complex computational models, especially when the optimization concerns multiple criteria (objectives).

Approaches used to solve optimization tasks generated in problems of control, planning, designing and management have completely changed during recent years. Cases with unimodal, convex, differentiable scalar goal functions disappeared from research labs, because a lot of satisfactory efficient methods were already developed. Thus, all that remains are the hard cases: multimodal, multi-criteria, non-differentiable, NP-hard, discrete problems, often with huge dimensionality. These practical tasks, generated by industry and market, have caused serious troubles in seeking global optimum. Great effort has been done by scientists in recent years in order to reinforce power of solution methods and to fulfill expectations of practitioners. The results obtained from algorithms development are still moderate at best, proving the remaining needs for further research in this area. In this paper we focus on the multiple-criteria cases, thanks to increasing power of computational capabilities of modern computer systems.

2. State of the art

Solving multi-criteria discrete optimization problems requires a method of comparing different solutions in terms of a vector of goal functions. Excellent taxonomy of possible approaches depending on user preferences, forms and time moments of their expressing, one can find in (Parveen & Ullah, 2011). The concept of Pareto efficiency is the one among most commonly used, since leaves the user final decision about the choice of solution without unknown or unexpressed a priori user preferences. Because of the strong NP-hardness of almost all practical scheduling problems, exact approaches (chiefly B&B schemes, examined mainly for two-machine cases), although theoretically excellent, in fact remain useless for practitioners, Tyagi, Varshney, and Chandramouli (2013). Hence, sufficiently good approximations of Pareto front are still welcome. The majority of approaches used to this aim are focused on quite sophisticated evolutionary algorithms and local search method, with few of them designed for independent multiple threads parallel computing in mind. Quite rarely appeared simulated annealing algorithms applied to the considered case, despite their evident simplicity of implementation; we perceive this fact as a lack of advanced ideas suitable for parallel SA. Coming back to the scheduling area, the commonly considered is the bi-criteria flow-shop scheduling problem, having a moderately complicated model and relatively well developed sequencing

* Corresponding author. Tel.: +48 71 3203281.
E-mail address: czeslaw.smutnicki@pwr.edu.pl (C. Smutnicki).

http://dx.doi.org/10.1016/j.cie.2015.09.003
0360-8352/

algorithms with rich set of benchmarks. Thus, provided below a short survey of algorithms and techniques is oriented on the flow shop problem (also leading in our research) and a few dominant approximate approaches.

A representative among evolutionary algorithms working in single-processor environment is *Multi-Objective Genetic Algorithm* (MOGA), Murata, Ishibuchi, and Tanaka (1996), designed for multi-criteria flow shop and implemented in bi-criteria case. A set of non-dominated solutions is created on the base of successive populations, by the process of solution management. To spread the search through the solution space, the weighted sum of individual criteria was used with random weights assigned at each iteration. Additionally, a method of sustaining elite solutions is included, adding a few Pareto solutions from the elite set to the next generation of population. An improved version of MOGA, called CMOGA proposed next, Murata, Ishibuchi, and Gen (2001), introduced new weight distribution between optimization criteria. The special cell structure allows on better weight selection, which in turn, caused the algorithm to find a better approximation of the Pareto front.

Another variant of genetic algorithm, equipped with initialization procedure inserting four good solutions into initial random population, was proposed by Pasupathy, Rajendran, and Suresh (2006). Their algorithm uses an external population in order to keep already found non-dominated solutions. Its evolution strategy is similar to the one used in NSGA-II algorithm (Deb, Pratap, Agarwal, & Meyarivan, 2002), but improving quality of Pareto frontier is based on two different local search procedures, applied to external population after performing the main part of the algorithm.

Application of simulated annealing has a lot of references, see survey (Hooda & Dhingra, 2011), however among enumerated there papers we do not find neither our approach nor significant progress in multi-criteria view. A simple single-trajectory simulated annealing algorithm was used for solving multicriteria problem transformed to single-criteria case by using weighted sum of two criteria, see Charavarthy and Rajendran (1999). To reinforce algorithm quality, a supporting local search method were introduced. The initial solution is constructed from one of the following methods: (a) *Earliest Due Date* (EDD), (b) *Least Static Slack* (LSS) and (c) NEH heuristic (Nawaz, Enscore, & Ham, 1983). The neighborhood generation is performed by the method of swapping two adjacent jobs.

Inspired by *Pareto Archived Evolution Strategy* (PAES) algorithm, Suresh and Mohanasundaram proposed *Pareto Archived Simulated Annealing* (PASA) algorithm (Suresh & Mohanasundaram, 2004), which is based on a new perturbation method. A scheme called *Segment Random Insertion* (SRI) is used to generate neighborhood of selected solution. To maintain good non-dominated solutions, an external archive is used. Starting solution is generated at random, while new solutions are selected using calibrated weighted sum of criteria.

A multiobjective parallel genetic algorithm was proposed in paper (Rashidi, Jahandar, & Zandieh, 2010). Evaluation was Pareto-based and used procedure called *Redirect*, which helped the algorithm to overcome the local optima. Test results showed that the proposed algorithm obtains solutions of good quality. Another algorithm, a *Genetic Local Search Algorithm* (GLSA), was proposed in (Ishibuchi & Murata, 1998). It applied local search procedure to maximize fitness value for each individual generated by genetic operators.

3. Philosophy of the new approach

Metaheuristics designed for conventional scheduling problems usually generate final solutions by the means of transforming the current solution (or set of solutions) into another over the course of subsequent iterations. The goal is to find the optimal solution (single-criteria case) or a Pareto front (multi-criteria case) or their approximation. Thus various metaheuristics employ various iteration-to-iteration transformation and produce different step-by-step trajectories of solutions. In Fig. 1, we can compare the search policies employed by existing approaches, namely SA (a) and GA (b). The first algorithm (Fig. 1a) considers single solution per iteration and chooses the next solution by employing a neighborhood search method (NS). Similar approach is shared by tabu search (TS). Ultimately, only single solutions are processed, which affects the quality of Pareto front approximation, even if external Pareto archive is used.

The GA (Fig. 1b) operates on sets of solutions (called populations or clouds), by the use of several genetic operators (GA) including mutation, crossover and selection, aiming to obtain a sufficient number of almost uniformly distributed non-dominated solutions. Such population-based metaheuristics (GA, ACO, PSO) considered by many authors as the most preferable for the generation of Pareto front, because the Pareto front itself (or its approximation) usually consists of many solutions (a population). However, the final results strongly depends on the transformation used to generate new populations and although great effort has been done in this field, GA and similar approaches for multi-criteria scheduling are still in the developing phase.

While the population-based approaches are often more popular, we observed that the SA algorithm performs well in the single- and even some multi-criteria cases. From there, we devised the possibility of a non-trivial extension of classic SA approach (Fig. 1a), by including the notion of solution sets (clouds), with the aim to improve the search process and allow more non-dominated solutions to be included into our archive. The proper formulation of such approach includes discussion and research on several basic topics, namely: (1) how to define the "cloud" of solution, (2) which solution should be accepted from current "cloud" to generate the next iteration cloud, (3) how to avoid local optima on the trajectory of the search and (4) what strategies should be employed in order to better approximate the Pareto front. Starting from the well-known single-trajectory simulated annealing (SA) approach we design desired algorithm in the way similar to parallel simulated annealing (pSA), having in mind the simplicity of implementation of both mentioned cases. Going over applications of SA to multi-criteria problems with Pareto fronts, we do not find in the literature neither

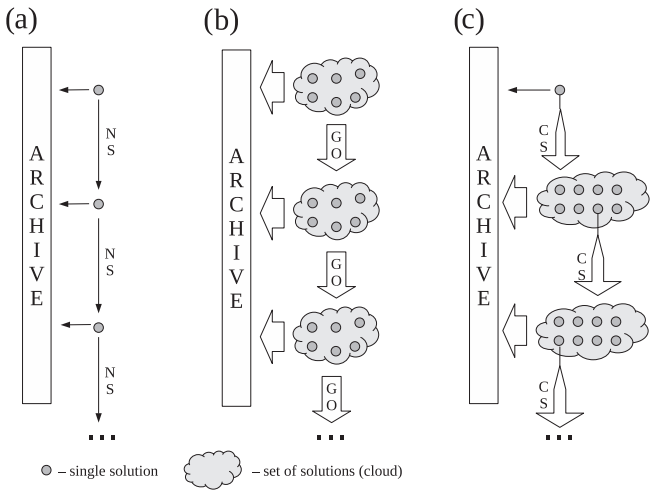

Fig. 1. Trajectory and Pareto archive update policy for different algorithms: (a) single trajectory (SA), (b) trajectory of clouds (GA), and (c) proposed approach (VESA). NS – neighborhood search, GO – genetic operators, CS – cloud search.

such approach nor any other approach similar to ours, so this proposal is original. In the subsequent, we present this approach in a more detailed way.

When following a trajectory, a neighboring solution x is taken from the neighborhood $N(x)$ of the current solution x. If the solution x is accepted, new neighborhood $N(x)$ is generated and further solutions from the previous neighborhood $N(x)$ are discarded. Though it is possible that optimal solution or, in case of the multi-criteria optimization, one or multiple Pareto-optimal solutions were among the unchecked solutions. Following that concept, it would be beneficial to further analyze the current neighborhood. Hence, the idea of generating accompanying "clouds", which can contain the solution from the trajectory that would normally be unchecked and thus would not be included in the Pareto archive. However, generating all the solutions from the neighborhood is time consuming and could lead to complete search, so only a certain number of the neighboring solutions is generated at random.

Research concluded beforehand on single criterion problem has shown, that sorting solutions from the "clouds" and accepting the best solution does not yield good results. Such approach tends to fall in the trap of local optima. Thus, our approach accepts first solution that satisfies the certain conditions and archives all non-dominated solutions found in the "cloud". Thus the concept of "cloud" of solutions is used to enhance one of the existing metaheuristic algorithms, creating the *Enhanced Simulated Annealing* (or ESA) algorithm.

Above mentioned concept has a natural aptitude for parallelization. Apart from distributed computing and computations on GPGPUs, there are instructions implemented in most of the common CPUs, which can be used to speed up the computations without the need for a specialized hardware. Hence, we propose a *Vector Evaluated Simulated Annealing* (or VESA) algorithm, which combines the "clouds" generation method of ESA with the fine-grained parallel computing techniques.

4. The test problem

In order to evaluate the suitability of the proposed approach we consider the flow shop scheduling problem, well known in the operations research theory. This is the manufacturing system with sequential structure of service stages (machines). In this system, n jobs from set $J = \{1, \ldots, n\}$ are to be carried out on m machines from set $M = \{1, \ldots, m\}$. Each job $j \in J$ is executed exactly once on each machine in order consistent with machines enumeration. Processing time of job j, $j \in J$, on machine k, $k \in M$, requires amount of time $p_{jk} > 0$. At any given moment machine can only execute one job at a time. The aim of job scheduling in production system is to determine time moments of start and completion of jobs on each machine in the system, while meeting the aforementioned constraints. In this paper a *permutation flow shop scheduling problem* (PFSSP) is considered, where single processing order (permutation on J) of loading jobs into the system has to be determined, whereas internal queues work with FIFO service rule and simply repeat the loading sequence (one can say that jobs are performed in the same order on each machine).

Let $C_{j,k}$, $j \in J$, $k \in M$ be the time moment of completion of job j on machine k. For permutation $\pi = (\pi(1), \ldots, \pi(n))$ defining the order of jobs execution, those moments can be determined from the recursive Eq. (1), which can be computed using iterative method in time O (nm), for $k = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$,

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

where $\pi(0) = 0$, $C_{j,0} = 0$ for $j = 1, \ldots, n$ and $C_{0,k} = 0$ for $k = 1, \ldots, m$.

We want to determine the set of Pareto-efficient permutations for maximal completion time (makespan) (2) and mean sum of completion times (3) criteria functions, namely

$$C_{\max}(\pi) = \max_{1 \leqslant j \leqslant n} C_{\pi(j),m}, \tag{2}$$

$$\overline{C}(\pi) = C_{\text{avg}}(\pi) = \frac{1}{n} \sum_{j=1}^{n} C_{\pi(j),m}. \tag{3}$$

Note that, values C_{jk} computed from Eq. (1) are the smallest possible, so they minimize both criteria functions at once for appointed permutation π.

The flow shop scheduling problem with minimization of makespan function or minimization of mean sum of completion times is considered NP-hard problem of discrete optimization. Therefore, the featured multi-objective problem is NP-hard as well.

5. ESA – Cloud exploration for simulated annealing

One of the most effective methods used in discrete optimization problems is an algorithm called simulated annealing (SA). This metaheuristic is based on thermodynamic process of annealing in metallurgy and it was first proposed in paper (Kirkpatric, Gelatt, & Vecchi, 1983). Moreover, SA is one of the limited number of metaheuristic approaches for which the convergence has been successfully proved in some cases for both the sequential (Granville, Krivanek, & Rasson, 1994) and parallel (Meise, 1998) version. The SA algorithm was also used by the authors previously to solve bi-objective permutation flowshop problem (Pempera, Smutnicki, & Żelazny, 2013). Obtained solution sets were compared with benchmarks from literature and proposed algorithm proved to provided good results in similar computation time. Thus, the SA was chosen to implement our cloud exploration philosophy and create the ESA algorithm which will be described in this chapter.

In case of single-trajectory, single criterion function $f()$ optimization, in each iteration of the SA algorithm, a perturbed solution x' is generated based on the current solution x and its neighborhood. If solution x' is no worse than solution x, then it takes its place in the next iteration. Otherwise, it takes its place with certain probability. This probability of accepting worse solution decreases with the growth of difference of criteria function values between solutions x' and x. It also decreases with the temperature drop, which likens SA to the annealing process occurring in Nature. Accurately, the probability of a worse solution being accepted can be formulated as follows

$$P = e^{-\Delta/T}, \tag{4}$$

where $\Delta = f(x') - f(x)$ and T is a temperature in the current iteration.

As mentioned previously the main idea behind the ESA algorithm is based on generating a number of solutions (a cloud) from the same neighborhood in advance, while the basic SA generates only one. This is done despite the fact that at most one of the generated solutions will be accepted. This is somewhat similar to the prefetching techniques used in processors, where many instruction are fetched from memory in advance, even though some of them may not execute in the end. Let us consult Fig. 2(a) for an example of basic SA algorithm. Full and dotted circles indicate accepted and rejected solutions respectively. Arrows indicate solution generation and evaluation. We start with solution S_0 and generate candidate solution S_{01}. Let us assume S_{01} is not accepted. S_0 remains the current solution and subsequent solutions S_{02} and S_{03} are generated in the next two iterations. Only the last solution, namely S_{03}, is accepted. Then from solution S_{03} two solutions (S_{031} and S_{032}) are generated and the second one is accepted. Next S_{0321} is

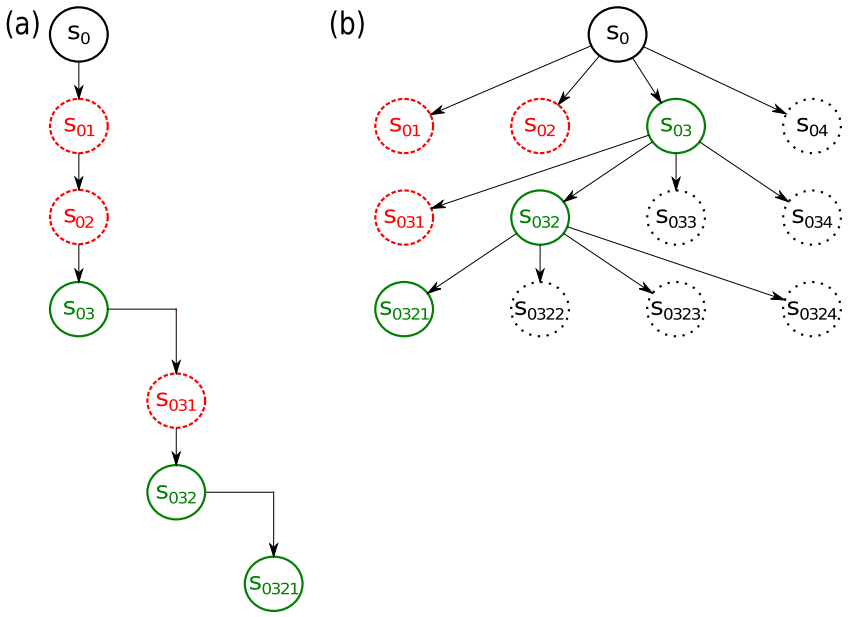

Fig. 2. Comparison of next solution generation in simulated annealing, (a) sequential and (b) parallel.

generated from S_{032} and accepted. In result, it took us 6 iterations to evaluate 6 solutions, to accept 3 of them, and to consider 6 solutions for updating the Pareto archive.

Solutions S_{01} and S_{02} were not accepted and therefore, S_{01}, S_{02} and S_{03} were derived from the same base solution (S_0). We can take advantage of that and evaluate S_{02} and S_{03} in advance, before we know whether S_{01} was accepted or not. This is justified, because in the basic algorithm the acceptance probability is relatively low, meaning that several candidate solutions will be evaluated on average, before one of them is accepted as the new current solution. Thus a cloud of solutions is generated instead of a single solution and the resulting trajectory can be similar to what is shown in Fig. 2(b).

The presented cloud approach technique ESA generates two methodology dilemmas. First, more than one solution among candidates might be accepted due to the probability. However, only one solution can be accepted per iteration. Because of that, a policy for choosing one solution from all accepted (or would-be-accepted) is needed. We could either choose the best among accepted solution or stop at the first one accepted and discard the others (if any). The initial research indicated that the latter approach yields better results in our case, thus we adopted that policy.

The second dilemma stems from the generation of the solution cloud itself. While generation of s solutions might improve our final results (by helping to avoid local optima for example), it does take s times more time to compute. In this case reduction of the execution time for the cloud generation procedure is needed. We will deal with this issue in the next chapter. For now, let us a define some indicator to help us measure the solution processing rate of SA and ESA algorithms

$$\eta_A = \frac{\Delta s_A(t)}{\Delta t} = \frac{s_A(t+\Delta t) - s_A(t)}{\Delta t}, \tag{5}$$

where Δt is time interval and $s_A(t)$ is the number of solutions processed by the algorithm A from the moment of its execution (time 0) until time t. For our ESA algorithm we would like to obtain the following inequality

$$\eta_{\mathrm{ESA}} > \eta_{\mathrm{SA}}. \tag{6}$$

This would prove that our algorithm processes more solutions per unit of time and thus has better solution space exploration properties. Coupled with the ability to better avoid local optima and other Pareto front quality indicators it will prove the superiority of our algorithm.

However, before that we need to define all the notions needed by metaheuristic such as SA (and ESA by extension). Those notions include the solution representation, method of generating perturbed solution (neighborhood) and a temperature scheme. For considered problem, a solution is represented as a permutation (schedule of jobs), perturbed solution is generated by randomly selecting one from *insert* neighborhood of current solution. Two numbers, a and b, are generated from uniform distribution from the $[1,\ldots,n]$ interval and element $\pi(a)$ is shifted on position b in permutation π.

Based on the proposals introduced in work (Pempera et al., 2013), for updating the archive and application of selection rule, each solution π', which is not dominated by solution π, is treated as a solution not worse than π. On the other hand, each solution π' dominated by solution π is treated as solution worse than π. In case of selecting worse solution, we use conventional probability acceptance criteria (4), with the enhanced meaning of Δ

$$\Delta = \sqrt{\sum_{i=1}^{q} \alpha_i (f_i(\pi') - f_i(\pi))^2}, \tag{7}$$

where $f_i(\pi)$ is the i-th component of vector goal function, α_i are normalization factors, $\sum_{i=1}^{q} \alpha_i = 1$, and q is the number of criteria. In our approach we treated both criteria as equal, as mean sum and max completion time have similar values, thus $\alpha_1 = \alpha_2 = 0.5$. During trial tests of SA algorithm it was determined that good solutions are generated if SA starts from relatively low temperature, amounting to around 100°. At the same time, it was observed that during computations the fraction of rejected solution candidates is around 2/3.

6. VESA – Introducing fine-grained parallel computing

In recent years parallel computing has been an area of constant interest, leading to noticeable and dynamic development of vast range of computer systems and algorithms utilizing multiple processors and other integrated circuits implementing concurrent and parallel data processing as well as multiple computers working together in a computer network.

Parallel computing, when used in fields like multi-criteria optimization, is usually divided into two distinct groups:

1. Coarse-grained. In this case entire algorithms or threads of execution are parallelized, often with little or no modifications to the base algorithm. Examples include simultaneous runs of several instances (processes) of the same random algorithm (perhaps with different parameters) or running entire algorithms or their large portions with the use of software threads. In both cases communication between instances is possible and is relatively inexpensive. Each instance is usually highly independent from the others, searching a different area of the solutions space and with a different trajectory.
2. Fine-grained. In this case usually small portions of algorithms are executed in parallel. This results in smaller pieces of parallel code and increase in cost of communication. Also, in this case the entire algorithm usually follows a single trajectory, which will change during the course of the search. Fine-grained parallelization employs more sophisticated and low-level parallel techniques as small threads, vector processing and so on.

As stated in the previous chapter, the main drawback of our cloud exploration approach is the fact that we have to generate a number of solutions instead of one, which of course increases the overall execution time of the algorithm. In order to overcome this issue we decided to implement the cloud exploration procedure using the techniques of parallel computing. Our parallelization approach is meant to compute the value of the object function for multiple schedules at once, thus allowing us to quickly check them and choose the accepted solution (if any). The goal is to obtain the η indicator superior to basic SA algorithm.

Let us start with general consideration for evaluating a set of permutations (schedules). Let $\{\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(g)}\}$, be a set consisting g permutations and let us denote by $C^{(x)}_{[s],k}$ the completion time of job $\pi^{(x)}(s)$ processed on machine k. By analogy, we mark $p^{(x)}_{[s],k} = p_{\pi^{(x)}(s),k}$.

Next, let $\overline{p}_{[s],k} = (p^{(1)}_{[s],k}, \ldots, p^{(g)}_{[s],k})$ and $\overline{C}_{[s],k} = (C^{(1)}_{[s],k}, \ldots, C^{(g)}_{[s],k})$ be vectors consisting of above-mentioned input and output data. The schedule of jobs execution for g permutations can be computed in time O (nm) performing computations on vector machine, which processes vectors consisting of g elements, realizing the parallel version of (1) given by (8).

$$\overline{C}_{[s],k} = \max(\overline{C}_{[s-1],k}, \overline{C}_{[s],k-1}) + \overline{p}_{[s],k}. \tag{8}$$

Note that the computations using vector machine should be g times faster than sequential ones. Unfortunately, proper calculations must be preceded by initiation of the elements of the vectors $\overline{p}_{[s],k}$, $s = 1, \ldots, n,\ k = 1, \ldots, m$. This initiation, in regard of the elements dependence from $s,\ k$ and permutation $\pi^{(x)},\ x = 1, \ldots, g$ can be performed sequentially in time O (gnm). Initialization can be significantly sped up by using appropriate data organization in memory and by using vector computations.

Without losing generality of the description, we assume that the number of machines m is equal to the number of the elements of vector g. We denote by $\overline{t}_j = (p_{j,1}, \ldots, p_{j,m})$ vector of processing times of job j. For the given position $s,\ s = 1, \ldots, n$, let us define matrix $T = [T_{x,k}]_{m\times m}$ as follows

$$[T_{x,k}]_{m\times m} = \begin{bmatrix} \overline{t}_{\pi^{(1)}(s)} \\ \overline{t}_{\pi^{(2)}(s)} \\ \vdots \\ \overline{t}_{\pi^{(1)}(m)} \end{bmatrix} = \begin{bmatrix} p_{\pi^{(1)}(s),1} & p_{\pi^{(1)}(s),2} & \cdots & p_{\pi^{(1)}(s),m} \\ p_{\pi^{(2)}(s),1} & p_{\pi^{(2)}(s),2} & \cdots & p_{\pi^{(2)}(s),m} \\ \vdots & \vdots & \ddots & \vdots \\ p_{\pi^{(m)}(s),1} & p_{\pi^{(m)}(s),2} & \cdots & p_{\pi^{(m)}(s),m} \end{bmatrix}, \tag{9}$$

and $P = [P_{x,k}]_{m\times m}$ as follows

$$[P_{x,k}]_{m\times m} = \begin{bmatrix} \overline{p}_{[s],1} \\ \overline{p}_{[s],2} \\ \vdots \\ \overline{p}_{[s],m} \end{bmatrix} = \begin{bmatrix} p^{(1)}_{[s],1} & p^{(2)}_{[s],1} & \cdots & p^{(m)}_{[s],1} \\ p^{(1)}_{[s],2} & p^{(2)}_{[s],2} & \cdots & p^{(m)}_{[s],2} \\ \vdots & \vdots & \ddots & \vdots \\ p^{(1)}_{[s],m} & p^{(2)}_{[s],m} & \cdots & p^{(m)}_{[s],m} \end{bmatrix}. \tag{10}$$

From the definition we have $T_{x,k} = t^{(x)}_{\pi^{(k)}(s)} = p_{\pi^{(k)}(s),x}$ and $P_{x,k} = p^{(x)}_{[s],k} = p_{\pi^{(x)}(s),k}$, so the matrix P is a transposition of matrix T.

The initiation of matrix T requires time O (m) using parallel transfer from memory. Transposition of the matrix T, for m being power of number 2, can be done in time O (m log m) using vector processing. Here we presents a method dedicated to performing on processors equipped in SSE set of instruction. Let us consider two SSE instructions PUNPCKHWD and PUNPCKLWD. They operate on 128-bit data, treating it as vector of eight 16-bits numbers (words). The PUNPCKHWD instruction works on the input data (operands) a and b by interleaving the upper four 16-bit words of both a and b into single result data r (register or memory location) as shown in Fig. 3. The PUNPCKLWD instruction works similarly, except that lower four words (0–3) are interleaved, instead of upper ones (4–7).

Similar instructions exist for interleaving data treated as four 32-bit (double words) and 64-bits (quadwords) numbers. These are called PUNPCKHDQ, PUNPCKLDQ, PUNPCKHQDQ and PUNPCKLQDQ. With those 6 instructions we can now define three functions:

$ex1(a,b)$ {a = PUNPCKHWD(a, b); b = PUNPCKLWD(a, b); },
$ex2(a,b)$ {a = PUNPCKHDQ(a, b); b = PUNPCKLDQ(a, b); },
$ex4(a,b)$ {a = PUNPCKHQDQ(a, b); b = PUNPCKLQDQ(a, b); }.

Note that in those functions a and b are both inputs (operands) and outputs, so the result of the first instruction must be stored in a temporary location and assigned to a only after both instructions have been executed.

We now treat each row of $T_{8\times 8}$ matrix as SSE 128-bit vector and calculate the transposition as follows:

1. $ex1(T_1, T_2),\ ex1(T_3, T_4),\ ex1(T_5, T_6),\ ex1(T_7, T_8)$.
2. $ex2(T_1, T_3),\ ex2(T_2, T_4),\ ex2(T_5, T_7),\ ex2(T_6, T_8)$.
3. $ex4(T_1, T_5),\ ex4(T_2, T_6),\ ex4(T_3, T_7),\ ex4(T_4, T_8)$.
4. $swap(T_2, T_5),\ swap(T_4, T_7)$.

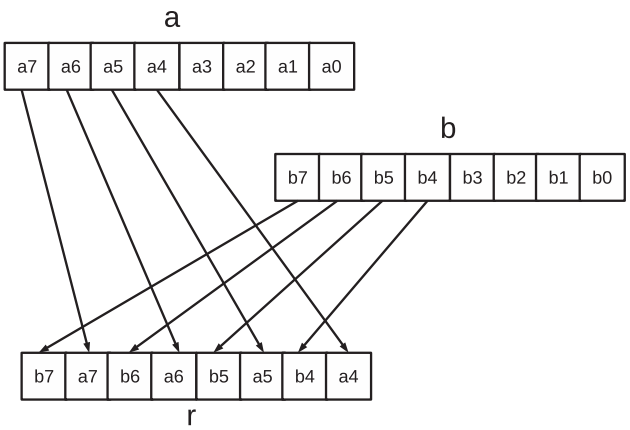

Fig. 3. SSE: PUNPCKHWD instruction interleaving a and b into r.

After the first three steps the columns of the matrix will become its rows. However, some resulting rows are misplaced. In the fourth step two SSE registers swapping operations are performed, properly ordering the rows. Example of the transposition process is shown on Fig. 4.

When assembly language alone is concerned the transposition of 8×8 matrix using standard swapping of 28 pairs of elements takes approximately 65% more instructions (i.e. resulting code size) that the SSE version. Both methods were employed during calculation of millions of matrices in a speed test. Results indicated that SSE version is 1.8 to over 11 times faster (depending on the optimization flags supplied to the compiler, with optimization working in favor of the SSE method).

The above method was implemented in our ESA algorithm, thus resulting in VESA. Let us take a moment to summarize step by step the changes compared to the standard SA:

1. SA. Sequential algorithm that evaluates and adds to the Pareto archive one candidate solution per iteration. Has tendency for trapping in local optima.
2. ESA. Sequential algorithm that evaluates and adds to the Pareto archive a number (8 in our implementation) of solutions per iteration. Has improved solution space exploration properties and better avoids local optima. However, the running time of a single iteration is increased compared to SA. In result $\eta_{ESA} < \eta_{SA}$.
3. VESA. Mostly sequential algorithm with parallel computation of the objective functions for all solutions inside a given "cloud". The execution time of a single iteration is greatly reduced compared to ESA, however this time might still be longer than that of SA, since SSE instructions are relatively complex compared to the regular processor instructions. In result we know for certain that $\eta_{VESA} > \eta_{ESA}$. The question remains whether $\eta_{VESA} > \eta_{SA}$, which will be checked during a computer experiment.

Let us note that all three algorithms (SA, ESA, VESA) can be further enhanced by performing more common coarse-grained parallelization techniques like running several instances of the same algorithm (with identical or various launching parameters). This is further supported by the fact that VESA makes use of SSE instructions only, which leaves all remaining processors (cores) available for other parallel enhancements. Let us also note that the SSE instruction set is available in any modern computer, thus removing the need for specialized hardware like GPUs, vector processors (like Xeon Phi) or computer clusters.

As a final note, the VESA aims to complete more computation than basic SA in the same (fixed) time. This is the application of the Gustafson's law (1988), which is less restricted than Amdahl's law. In result, larger "clouds" could be computed in the same time given more advanced vector processing instructions.

7. Computer experiments

The overall aim of experiments was to evaluate the performance of the proposed VESA algorithm, in terms of the running speed and quality of found Pareto fronts. Both algorithms, with parallel vector processing (VESA) and classic sequential (SA) were implemented in C++, compiled and run under Visual Studio 2010. Tests were performed on a personal computer equipped with Intel Core i7 2.3 GHz processor, albeit all computation were performed using single core. All tests were carried out on instances provided in (Minella, Ruiz, & Ciavotta, 2008), which were based on well known benchmarks proposed by Taillard (1993).

For each instance of the problem, both SA and VESA algorithms were performed from solutions generated using certain construction algorithm, for initial temperature $T_0 = 100$ and final temperature $T_k = 1$. Parameter λ was selected in such a way that algorithms performed 10 000 iterations. Parallel processing was performed using vectors of length $g = 8$.

For each instance and each algorithm, following data were stored: set of the non-dominated solutions, computation times and, in case of VESA algorithm, number of iterations in which parallel processing was performed. Based on those variables the solution processing rates (η) and quality of the generated non-dominated solutions were determined.

By steering random numbers generator we caused both algorithms, SA and VESA, to reproduce the same trajectory of the search. As is, the VESA algorithm generates solutions not worse than SA algorithm. However, the set of non-dominated solutions

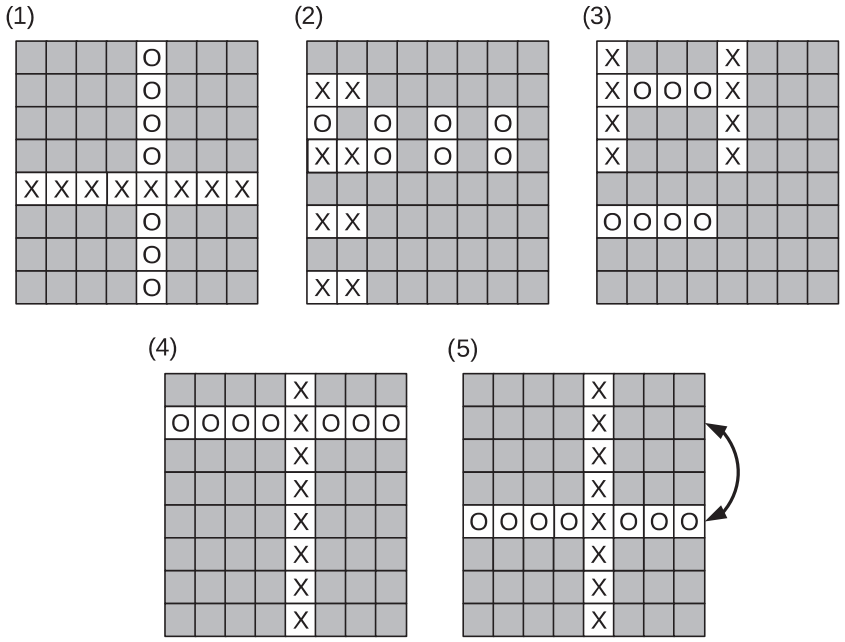

Fig. 4. Steps of matrix transposition using SSE instructions: the original matrix (1), *ex*1, *ex*2 and *ex*4 transformations (2, 3 and 4) and rows swapping (5).

generated by VESA algorithm was updated by all Pareto efficient permutations acquired in prefetching phase of computing.

Each algorithm H was evaluated using two measures: (1) mean running time CPU_H and (2) efficiency coefficient of approximating Pareto front I_H. In the latter case, we use quality indicator called Hyper-Volume Indicator, which was proposed by Zitzler and Thiele in their work (Zitzler & Thiele, 1999) and more extensively studied alongside with Knowles et al. in work (Knowles, Thiele, & Zitzler, 2006). The comparison of the quality of the approximation of Pareto frontier provided by algorithms is performed as follows. Firstly, a reference point is assigned, usually for each of criterion it takes 120% of the highest value obtained by all tested algorithms. Secondly, a size of the area constrained by reference point and Pareto approximation gained from certain algorithm is computed. Thus, the I_H metric relies on the solutions found (objective function values) rather than computation time needed to do so. A graphical interpretation of I_H is shown in Fig. 5.

Several various tests were performed. In Test 1 we observed the convergence of Pareto archive set to target approximation of Pareto front in iterations of ESA and VESA. Typical run is shown in Fig. 6. In order to improve visibility and to support the interpretation, discrete points followed from Pareto archive are linked by linear segments in I_H style. Legend refers to thousands of iterations, e.g. 50 actually means 50,000. Unfortunately, there is no results in the literature of this subject to make the comparison. Fig. 6 lead us to the following observations: (1) Pareto archive converges to best approximation of Parto front constantly (uniformly) during successive iterations, (2) essential changes in Pareto archive (improvements of the approximation) have irregular character, relevant to each considered instance, (3) quality of Pareto front approximation improves with increasing number of iterations and (4) the most important changes appear during the initial iterations of the algorithm.

Test 2 was carried out to observe the detailed behavior of VESA in iterations. For various instances, we traced parameter I_H in iterations. Typical run is shown in Fig. 7. Note that iterations in the figure are given in logarithmic scale. One can find the following conclusions: (1) Pareto archive converges to approximation of Pareto front effectively, (2) the most essential change is observed in the initial iterations as later iterations provide less improvement. This is evident by the shape of the resulting curve, which is approximately linear while using logarithmic scale, thus proving that increasing the number of iterations 10 times results in roughly twice the improvement and (3) the resulting improvement, compared to the starting solution, is 28/19 (in millions), meaning 45% increase in I_H.

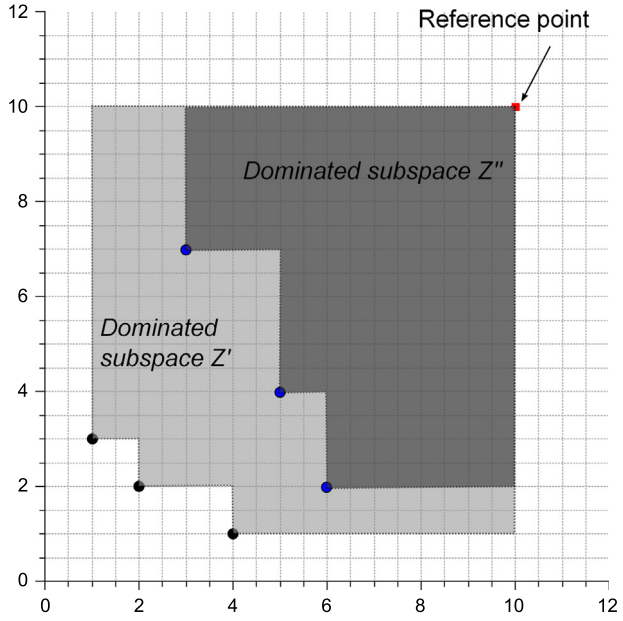

Fig. 5. Interpretation of the Hyper-Volume Indicator (I_H).

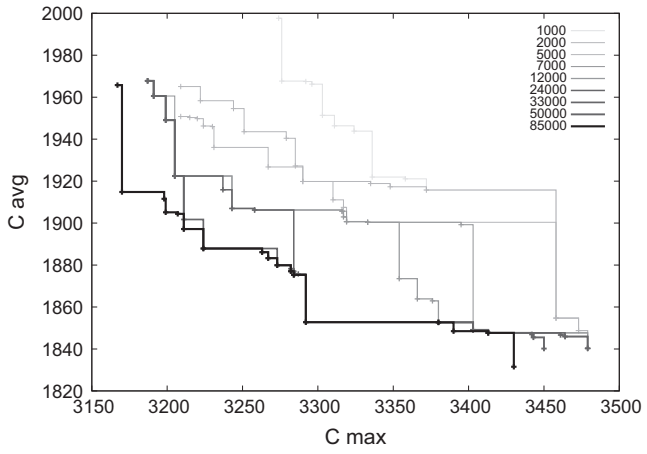

Fig. 6. ESA and VESA comparison. Convergence of Pareto archive sets to Pareto front based on the number of iterations. Instance TA41.

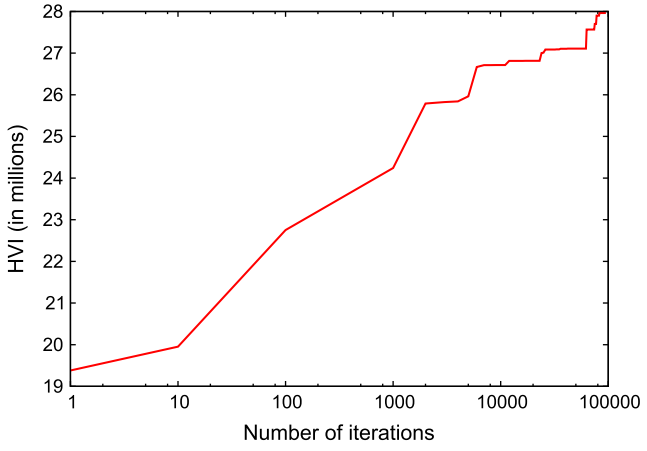

Fig. 7. ESA and VESA comparison. Convergence of I_H in iterations. Instance TA41.

To compare efficiency of ESA&VESA with respect to SA we made Test 3. In Fig. 8 there is shown typical relation between Pareto fronts found using both mentioned approaches. It is evident that conventional SA cannot be recommended for multicriteria applications. Pareto front of VESA contains more points than SA and is located "deeper".

Test 4 refers to mass experiments concerning running times and solution processing rates. Results are collected in Table 1. In the second and the third column real (absolute) computation time for SA and VESA algorithms is shown, in the fourth one a ratio (R_{sol}) of the number of solutions processed by VESA algorithm to the number of solutions processed by SA algorithm is shown. In the last two columns two ratios are presented, real-time ratio $R_{time} = CPU_{VESA}/CPU_{SA}$ and ratio of the solution processing rates, for both algorithms:

$$\frac{R_{sol}}{R_{time}} = \frac{\eta_{VESA}}{\eta_{SA}} \tag{11}$$

The trails results presented in Table 1 show, that VESA algorithm processed instances nearly twice as long as SA algorithm. It is clearly visible in the column representing R_{time} values.

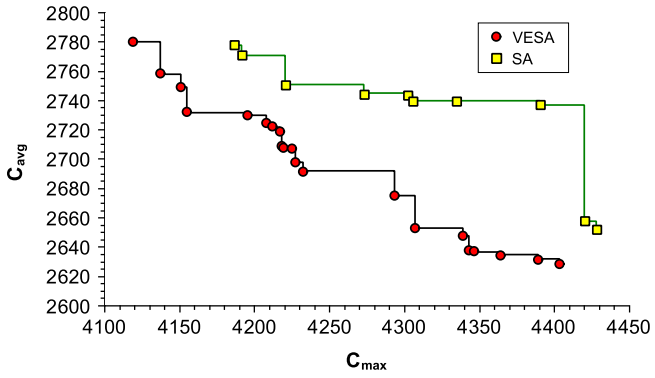

Fig. 8. Comparison of Pareto fronts for SA and VESA. Instance TA51.

Table 1
Absolute and relative computation times and solutions checking rates for the VESA and SA algorithms.

Group	CPU_{SA} (s)	CPU_{VESA} (s)	R_{sol}	R_{time}	R_{sol}/R_{time}
20 × 5	0.10	0.20	2.63	2.00	1.31
20 × 10	0.20	0.41	3.45	2.04	1.68
20 × 20	0.29	0.65	3.94	2.22	1.76
50 × 5	0.19	0.39	2.45	2.04	1.20
50 × 10	0.31	0.67	2.47	2.17	1.14
50 × 20	0.60	1.02	2.76	1.69	1.62
100 × 5	0.24	0.62	2.13	2.56	0.82
100 × 10	0.55	1.22	2.46	2.22	1.11
100 × 20	1.10	1.98	2.66	1.79	1.48
200 × 10	0.92	2.28	2.30	2.50	0.93
200 × 20	2.00	3.80	2.70	1.89	1.42
Average			2.70	2.08	1.31

However, VESA algorithm processed in average 2.7 times more solutions than SA. We conclude that when solution processing rate is considered, the VESA algorithm performed averagely 1.31 times better than SA algorithm and thus $\eta_{VESA} > \eta_{SA}$.

This processing rate ratio value increases with the number of machines and decreases with the number of jobs. Those results support our previous assumption – even though VESA takes roughly twice the time to compute, it evaluates and accepts solutions roughly 2.6 times more often. The increase of computational time is caused by the computational complexity of SSE instructions used to evaluate solutions and the need to compare each new evaluated solution with the archived Pareto set. The ability to evaluate more solutions results in better approximation of the Pareto frontier, as can be seen in Fig. 8.

In Table 2 a comparison of benchmark solution sets provided in paper (Minella et al., 2008) and VESA algorithms is presented. The first column represents a group of instances described earlier, in the second and the third one average values of I_H are shown, while in the fourth an average cover ratio of VESA algorithms I_H value to the values from benchmarks. As it can be seen, the average ratio drops with the increase of instance size. The benchmarks we compared with results (called hereinafter MRC) provided in the paper of Minella et al. (2008), extracted from 16 different algorithms, each executed 10 times, and are considered as best known up to date. Although VESA Pareto sets have lower values of I_H indicator, they were provided in shorter or similar computation time than computation times of algorithms tested by Minella et al. Moreover, changing the parameters of the algorithm (i.e. increasing number of runs or iterations) did not change the results drastically. Proposed VESA algorithm provided good results in computation times comparable to those from benchmarks.

Table 2
Comparison of Hyper-Volume Indicator for the VESA and MRC results.

Group	MRC I_H	VESA I_H	I_H ratio (%)
20 × 5	0.073	0.070	95.9
20 × 10	0.076	0.070	92.1
20 × 20	0.075	0.069	92.0
50 × 5	0.078	0.068	87.2
50 × 10	0.089	0.071	79.8
50 × 20	0.086	0.066	76.7
100 × 5	0.071	0.063	88.7
100 × 10	0.082	0.066	80.5
100 × 20	0.084	0.063	75.0
Average	0.079	0.067	85.3

Since the main aspect of the proposed algorithm was the new implementation of parallel computing, using aforementioned SSE instructions, obtained results of I_H indicator are good in comparison to available benchmarks. Especially considering its parallel structure and no need for specialized computing equipment, like GPUs or multi-core processors.

8. Conclusions

In this paper a new enhanced version of the simulated annealing algorithm, called VESA, was proposed for bi-criteria flow shop scheduling problem. VESA considers a cloud of candidate solutions instead of a single one and uses vector parallel processing to reduce the execution time of those additional calculations. Furthermore, all parallel-evaluated permutations, accepted or not, were used for approximation of the Pareto frontier, therefore taking further advantage of the multi-objective properties of the problem. This also server to help the algorithm to better avoid local optima. The test results of the algorithm clearly show that VESA is comparatively faster, as it evaluates around 31% more solutions per unit of time than the basic algorithm ($\eta_{VESA} = 1.31\eta_{SA}$). Moreover, previously performed tests showed that it gets better results than classic SA. Compared to benchmarks from literature (16 different algorithm), it provides good quality solution sets in short computation time.

References

Charavarthy, K., & Rajendran, C. (1999). A heuristic for scheduling in a flow shop with the bi-criteria of makespan and maximum tardiness minimization. *Production Planning and Control, 10*, 707–714.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197.

Granville, V., Krivanek, M., & Rasson, J.-P. (1994). Simulated annealing: A proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 16*(6), 652–656.

Gustafson, J. L. (1988). Reevaluating Amdahl's law. *Communications of the ACM, 31*, 532–533.

Hooda, N., & Dhingra, A. K. (2011). Flow shop scheduling using simulated annealing: A review. *International Journal of Applied Engineering Research, 2*, 234–249.

Ishibuchi, H., & Murata, T. (1998). A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 28*(3), 392–403. http://dx.doi.org/10.1109/5326.704576.

Kirkpatric, S., Gelatt, C., & Vecchi, M. (1983). Optimisation by simulated annealing. *Science, 220*, 671–680.

Knowles, J., Thiele, L., & Zitzler, E. (2006). *A tutorial on the performance assessment of stochastic multiobjective optimizers*. Tech. rep., ETH Zurich.

Meise, C. (1998). On the convergence of parallel simulated annealing. *Stochastic Processes and their Applications, 76*(1), 99–115.

Minella, G., Ruiz, R., & Ciavotta, M. (2008). A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing, 20*(3), 451–471.

Murata, T., Ishibuchi, H., & Gen, M. (2001). Specification of genetic search directions in cellular multi-objective genetic algorithms. In *Proceedings of the first international conference on evolutionary multi-criterion optimization, EMO '01* (pp. 82–95).

Murata, T., Ishibuchi, H., & Tanaka, H. (1996). Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers & Industrial Engineering, 30*(4), 957–968.

Nawaz, M., Enscore, E. E., Jr., & Ham, I. (1983). A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. *Omega, 11*(1), 91–95.

Parveen, S., & Ullah, H. (2011). Review on job-shop and flow-shop scheduling using multi criteria decision making. *Journal of Mechanical Engineering, 41*(2), 130–146.

Pasupathy, T., Rajendran, C., & Suresh, R. (2006). A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. *International Journal of Advanced Manufacturing Technology, 27*(7-8), 804–815.

Pempera, J., Smutnicki, C., & Żelazny, D. (2013). Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm. *Procedia Computer Science, 18*(0), 936–945 (International Conference on Computational Science 2013).

Rashidi, E., Jahandar, M., & Zandieh, M. (2010). An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. *International Journal of Advanced Manufacturing Technology, 49*(9-12), 1129–1139. http://dx.doi.org/10.1007/s00170-009-2475-z.

Suresh, R. K., & Mohanasundaram, K. M. (2004). Pareto archived simulated annealing for permutation flow shop scheduling with multiple objectives, in: *2004 IEEE conference on cybernetics and intelligent systems* (Vol. 2, pp. 712–717).

Taillard, É. D. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*(2), 278–285.

Tyagi, N., Varshney, R. G., & Chandramouli, A. B. (2013). Six decades of flowshop scheduling research. *International Journal of Scientific and Engineering Research, 4*, 854–864.

Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *Transactions on Evolutionary Computation, 3*(4), 257–271.