Paper

Effective Design of the Simulated Annealing Algorithm for the Flowshop Problem with Minimum Makespan Criterion

Jarosław Hurkała and Adam Hurkała

Institute of Control and Computation Engineering, Warsaw University of Technology, Warsaw, Poland

Abstract—In this paper we address the n-job, m-machine flowshop scheduling problem with minimum completion time (makespan) as the performance criterion. We describe an efficient design of the Simulated Annealing algorithm for solving approximately this NP-hard problem. The main difficulty in implementing the algorithm is no apparent analogy for the temperature as a parameter in the flowshop combinatorial problem. Moreover, the quality of solutions is dependent on the choice of cooling scheme, initial temperature, number of iterations, and the temperature decrease rate at each step as the annealing proceeds. We propose how to choose the values of all the aforementioned parameters, as well as the Boltzmann factor for the Metropolis scheme. Three perturbation techniques are tested and their impact on the solutions quality is analyzed. We also compare a heuristic and randomly generated solutions as initial seeds to the annealing optimization process. Computational experiments indicate that the proposed design provides very good results – the quality of solutions of the Simulated Annealing algorithm is favorably compared with two different heuristics.

Keywords—flowshop, heuristics, makespan, simulated annealing.

1. Introduction

The flowshop problem has been studied by many researches because of the educational character and many real-life applications. Various optimization techniques with different assumptions have been used to solve this problem. The regular flowshop problem consists of a group of m machines and a set of n jobs to be processed on these machines. Each job is processed one at a time and only once on each machine (preemption is not allowed). A job cannot be processed simultaneously on more than one machine. The same processing order of jobs applies to each of the m machines.

In this paper, we consider the classical flowshop-sequencing problem with minimum completion time (makespan) and assume infinite buffer at any machine in the processing sequence, so that jobs may form queues and wait between the machines without blocking them. The makespan criterion can be defined as a completion time, at which all jobs complete processing or equivalently as a maximum completion time of jobs. The flowshop scheduling problem with the makespan criterion is indicated by $n/m/F/C_{max}$ and the aim is to find the order of jobs that minimizes the makespan.

The n-job m-machine flowshop problem belongs to the class of NP-hard problems [1]. Because the search space grows exponentially as the number of jobs increases, obtaining the optimal solutions for large-size problems with exact methods in reasonable time is impossible. As a consequence, many researchers have been developing various heuristics for this problem. These include constructive heuristics [2], [3], metaheuristics like Simulated Annealing [4]–[6] and Tabu Search [7]–[10], evolutionary algorithms, such as Genetic Algorithm [11], [12], and other neighbor search approaches [13].

In this paper, we comprehensively describe the design of the Simulated Annealing algorithm for the purpose of effectively solving the flowshop problem. In Section 2, we present the objective function and propose an alternative approach for calculating the makespan. In Sections 3 and 4, we explain in details the design of the Simulated Annealing algorithm for the flowshop problem and briefly describe two constructive heuristics that we compare the outcomes with. The results of the experiments are shown in Section 5. In Section 6 some concluding remarks are presented.

2. Problem Definition

The classical flowshop problem, we focus on in this paper, can be defined as follows, using the notation by Nowicki, Smutnicki [10], Grabowski, Pempera [8] and Wodecki, Bożejko [6]. We consider a set of n jobs $J = \{1,2,\ldots,n\}$, and a set o m machines $M = \{1,2,\ldots,m\}$. Job $j \in J$, consists of a sequence of operations $O_{j1}, O_{j2}, \ldots, O_{jm}$, where operation O_{jk} corresponds to the processing of job j on machine k and takes p_{jk} time. The goal is to find the sequence of jobs that minimizes the completion time of all jobs.

Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ be a permutation of jobs and Π be the set of all permutations. Each permu-

tation $\pi \in \Pi$ defines a processing order of jobs on each machine. We want to find a permutation $\pi^* \in \Pi$ such that:

$$C_{max}(\pi^*) = \min_{\pi} C_{max}(\pi), \tag{1}$$

where $C_{max}(\pi)$ is the makespan of the processing order given by π, and can be found by the following recursive formula:

$$C_{jk}(\pi) = \max\left\{C_{\pi(j-1)k},\ C_{\pi(j)k-1}\right\} + p_{\pi(j)k}, \tag{2}$$

$$C_{max}(\pi) = C_{nm}(\pi), \tag{3}$$

where $\pi(0) = 0,\ C_{0k} = 0,\ k = 1,2,\ldots,m,\ C_{j0} = 0,\ j = 1,2,\ldots,n$.

It is also well known in the literature that the makespan associated with permutation π can be found by:

$$C_{max}(\pi) = \max_{1 \le t_1 \le \cdots \le t_{m-1} \le n} \left(\sum_{j=1}^{t_1} p_{\pi(j)1} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m} \right). \tag{4}$$

This optimization problem can be considered as finding the longest (critical) path from node (1, 1) to (m, n) in a grid graph. Each path in such graph is composed of horizontal and vertical sub-paths.

In this paper, we propose another approach of calculating the makespan. Instead of using the recursive formula or solving the longest path problem, we have created an algorithm that simulates the flowshop, hence finding the maximum completion time of jobs. For the overview of the algorithm see Algorithm 1.

Algorithm 1 Flowshop simulation

```
1: C_max ← 0, δt ← 0, q_1 ← π
2: repeat
3:    C_max ← C_max + δt
4:    δt ← ∞
5:    for k from m downto 1 do
6:       if r_k > 0 then
7:          r_k ← r_k − δt
8:          if r_k = 0 then
9:             if k + 1 ≤ m then
10:               add(q_{k+1}, c_k)
11:            end if
12:            c_k ← ∅, δt ← 0
13:         else
14:            δt ← min{δt, r_k}
15:         end if
16:      else if q_k ≠ ∅ then
17:         c_k ← removeFirst(q_k)
18:         r_k ← p_{c_k k}
19:         δt ← min{δt, r_k}
20:      end if
21:   end for
22: until δt = ∞
23: return C_max
```

The design of the algorithm is fairly simple. Let $c_k^i \in J \cup \{\emptyset\}$ be the job processed on machine k in iteration i, r_k^i be the remaining time of processing this job on machine k in iteration i, and $q_k^i \subset J \cup \{\emptyset\}$ be the job queue at machine k in iteration i. The makespan can be calculated from the following formula:

$$C_{max}(\pi) = \sum_i \delta t^i, \tag{5}$$

where δt^i is the time step by which we increase the makespan in iteration i:

$$\delta t^i = \min_{k=1,\ 2,\ldots,m} r_k^i. \tag{6}$$

The remaining time of processing on machine k in iteration $i+1$ is calculated as follows:

$$r_k^{i+1} = \begin{cases} r_k^i - \delta t^i & \text{if } r_k^i > 0 \\ p_{c_k^{i+1}k} & \text{if } r_k^i = 0 \wedge q_k^i \neq \{\emptyset\} \\ 0 & \text{otherwise} \end{cases}, \tag{7}$$

where $\delta t^0 = 0, r_k^0 = 0,\ c_k^0 = \{\emptyset\}, k = 1,2,\ldots,m$.

The job processed on machine k in iteration $i+1$ is found by:

$$c_k^{i+1} = \begin{cases} c_k^i & \text{if } r_k^i > 0 \wedge r_k^i - \delta t^i > 0 \\ q_k^i(1) & \text{if } r_k^i = 0 \wedge q_k^i \neq \{\emptyset\} \\ \{\emptyset\} & \text{otherwise} \end{cases}, \tag{8}$$

where $q_k^i(1)$ is the first element in the job queue, $k = 1,2,\ldots,m$.

The queue at machine k in iteration $i+1$ is calculated as follows:

$$q_k^{i+1} = \begin{cases} q_k^i \cup \left\{c_{k-1}^i\right\} & \text{if } r_{k-1}^i > 0 \wedge r_{k-1}^i - \delta t^i = 0 \\ q_k^i - \{q_k^i(1)\} & \text{if } r_k^i = 0 \wedge q_k^i \neq \{\emptyset\} \\ q_k^i & \text{otherwise} \end{cases}, \tag{9}$$

where $q_1^0 = \pi, q_{k+1}^0 = \{\emptyset\}, k = 1,2,\ldots,m-1$.

3. Simulated Annealing

The Simulated Annealing (SA) was first introduced by Kirkpatrick [14], while Černý [15] pointed out the analogy between the annealing process of solids and solving combinatorial problems. Researchers have been studying the application of the SA algorithm in various fields of

optimization problems, but more importantly, it was shown that SA can be applied to sequencing problems [16].
The process of Simulated Annealing can be described as follows. First, an initial solution must be specified as a starting point. Then, repeatedly, a candidate solution is randomly chosen from the neighborhood of the current solution. If the newly generated solution is better than the current one, it is accepted and becomes the new current solution. Otherwise, it still has a chance to be accepted with, so called, acceptance probability. This probability is determined by the difference between objective function of the current and the candidate solution, and depends on a control parameter, called temperature, taken from the thermodynamics. After a number of iterations the temperature is decreased and the process continues as described above. The annealing process is stopped either after a maximum number of iterations or when a minimum temperature is reached. The best solution that is found during the process is considered a final. For the algorithm overview see Algorithm 2.

Algorithm 2 Simulated Annealing

Require: Initial schedule π_0
1: $\pi^* \leftarrow \pi_0$
2: **for** i from 1 to N **do**
3: **for** t from 1 to N_{temp} **do**
4: $\pi \leftarrow perturbate(\pi_0)$
5: $\delta \leftarrow C_{max}(\pi) - C_{max}(\pi_0)$
6: **if** $\delta < 0$ or $e^{-\delta/k\tau} > random(0,1)$ **then**
7: $\pi_0 \leftarrow \pi$
8: **end if**
9: **if** $C_{max}(\pi) < C_{max}(\pi^*)$ **then**
10: $\pi^* \leftarrow \pi$
11: **end if**
12: **end for**
13: $\tau \leftarrow \tau * \alpha$
14: **end for**
15: **return** π^*

In order to solve the flowshop problem with the SA algorithm, the annealing process needs to be adapted to this particular problem and values of several parameters must be determined.
The main step of the SA is the procedure of generating a candidate solution from the neighborhood of the current one, which is often called a perturbation scheme or transition operation. Although there are many ways to accomplish this task, we have examined the three most popular techniques:

- Interchanging two adjacent jobs.
- Interchanging two jobs.
- Moving a single job.

The key element of SA is to define the temperature decrease schedule, also called the cooling scheme. The main issue at this point is to determine values for the following parameters:

– initial temperature,

– function of temperature decrease in consecutive iterations,

– the number of iterations at each temperature (Metropolis equilibrium),

– minimum temperature at which the algorithm terminates or alternatively the maximum number of iterations as the stopping criterion.

The cooling process is usually simulated by decreasing the temperature by a factor, called the reduce factor. Let τ be the temperature and α be the reduce factor. Then the annealing scheme can be represented as the following recursive function:

$$\tau^{i+1} = \alpha * \tau^i, \tag{10}$$

where i is the number of current iteration in which the cooling schedule takes place.
Another building block of SA that has to be customized is the acceptance probability function, which determines whether to accept or reject candidate solution that is worse than the current one. The most widely used function is:

$$p(\delta, \tau) = e^{-\delta/k\tau}, \tag{11}$$

where δ is the difference between the objective function of the candidate (π) and the current solution (π_0):

$$\delta = C_{max}(\pi) - C_{max}(\pi_0), \tag{12}$$

and k is the Boltzmann constant found by:

$$k = \frac{\delta^0}{\log \frac{p^0}{\tau^0}}, \tag{13}$$

where δ^0 is an estimated minimal difference between objective function of two solutions, p^0 is the initial value of the acceptance probability and τ^0 is the initial temperature. Notice that we use decimal logarithm rather than natural, which is most widely seen in the literature. Moreover, rather than average, we use estimation of the minimal difference between solutions.
After thorough analysis of the SA application for the flowshop problem, we have arrived at the following initial values of all the aforementioned parameters that should be used to achieve the best results and make the most of the Simulated Annealing algorithm – see Table 1.

Table 1
Initial values of Simulated Annealing parameters

Param.	Description	Value
α	Reduce factor	$1-\frac{7}{N}$
τ^0	Initial temperature	0.99
δ^0	Estimated minimal difference between solutions	1
p^0	Initial value of acceptance probability	1
k	Boltzmann constant	$1/\log\left(\frac{1}{0.99}\right)$
N_{temp}	Number of iterations at each temperature	10
N	Number of SA iterations	1000000

4. Heuristic Algorithms

4.1. CDS Algorithm

The flowshop problem with two machines and the makespan criterion $(n/2/F/C_{max})$ can be solved by applying the famous Johnson's optimal rule, saying that job i precedes job j in an optimal sequence if:

$$\min\left\{p_{i1}, p_{j2}\right\} \leq \min\left\{p_{i2}, p_{j1}\right\}. \tag{14}$$

The Johnson's algorithm implementing this rule can be described in the following four steps:

1. Let $U = \left\{j : p_{j1} < p_{j2}\right\}$ and $V = \left\{j : p_{j1} \geq p_{j2}\right\}$.
2. Sort U in non-descending order by p_{j1}.
3. Sort V in non-ascending order by p_{j2}.
4. Set $\pi^* = U \cup V$ is the optimal job sequence.

Many researchers have tried to extend the rule for larger problems with more machines. An algorithm named CDS was proposed in [2] for the flowshop problem with makespan performance criterion, that effectively solves instances with any number of machines.
The algorithm is based on a heuristic application of the Johnson's rule to a two-machine sub-problem, obtained by merging machines to artificial machine centers.
The CDS algorithm creates $m-1$ two-machine sub-problems:

$$p^*_{j1} = \sum_{k=1}^{i} p_{jk}, \tag{15}$$

$$p^*_{j2} = \sum_{k=i+1}^{m} p_{jk}, \tag{16}$$

where i is the number of sub-problem, $j = 1,2,\ldots,n$.

Each sub-problem is solved with the Johnson's algorithm and one of the obtained $m-1$ sequences with the lowest makespan becomes the final solution of the main m-machine problem.

4.2. NEH Algorithm

Another approach for solving the flowshop problem is to construct the schedule by adding one job at a time to the sequence of jobs instead of calculating the makespan for the entire sequence of jobs at once. An excellent example of such algorithm is the NEH heuristic proposed in [3], which is considered the best constructive heuristic for the makespan flowshop problem.
This heuristic method is based on the assumption that in the process of constructing the schedule a job with higher value of total processing time on all machines should have higher priority and be taken into consideration before other jobs.
The algorithm consists of the following four steps:

1. Sort jobs in non-ascending order of total processing time on all machines.
2. Take the first of the remaining (unscheduled) jobs.
3. Find a position of the job in the partial sequence that minimizes makespan of the extended by this job partial sequence.
4. If there are more unscheduled jobs, go to Step 2.

At each iteration there are k possible places, at which a job can be inserted, where k is the iteration number. At the last iteration, the best partial sequence extended by the remaining job is the final schedule and the solution of the makespan problem.

5. Results

The design of the Simulated Annealing algorithm has been tested on a subset of the collection of flowshop problems developed by Taillard [17]. We have selected following different problem sizes: $n = \{20,50,100\} \times m = \{5,10,20\}$, and chosen first 4 instances of each of the 9 problem classes, which gave us a total of 36 instances. Each instance was solved 20 times and the best result was taken as final.
The difference between the algorithms was calculated by the following formula:

$$\eta(x,y) = \frac{x-y}{y}. \tag{17}$$

For the small-size problems (instances with 20 jobs or 5 machines) the SA algorithm is beyond compare. For large-size problems (50×10, 50×20, 100×10, 100×20) it outperforms the CDS algorithm on average by 13% and

is better than the NEH algorithm by more than 4% (see Tables 2–4 for more detailed results).

Table 2
Average results of the CDS algorithm

$\eta(CDS,T)$ [%]		m			Avg
		5	10	20	
n	20	7.48	14.96	11.73	11.39
	50	6.65	14.78	15.92	12.45
	100	5.22	10.21	14.44	9.96
				Total average:	11.27

Table 3
Average results of the NEH algorithm

$\eta(NEH,T)$ [%]		m			Avg
		5	10	20	
n	20	2.69	4.75	3.19	3.54
	50	0.62	5.04	6.59	4.09
	100	0.76	2.11	5.36	2.74
				Total average:	3.46

Table 4
Average results of the SA algorithm

$\eta(SA,T)$ [%]		m			Avg
		5	10	20	
n	20	0.00	0.00	0.00	0.00
	50	0.00	0.57	0.83	0.47
	100	0.02	0.16	0.99	0.39
				Total average:	0.29

While the SA algorithm is superior in terms of solution quality, it requires more time to compute the results than heuristic algorithms like CDS or NEH. Nevertheless, on a 3.1 GHz CPU it has found all the solutions in a reasonable time – ranging from less than 30 s for 20×5 instances to about 6 minutes for 100×20 instances (see Table 5).

Table 5
Average solution times of the SA algorithm

$SA_{Time}[s]$		m		
		5	10	20
n	20	28	65	128
	50	43	105	208
	100	72	178	356

In order of brevity, we present the results obtained only with 'move a single job' permutation scheme, since it generally finds better solutions than the other two presented techniques. This can be explained by the fact that this particular scheme generates the largest neighborhood of the current solution and it requires only $O(n)$ operations to move from the current to any permutation (transition path length). The main reason this technique outperforms the other two, is that, it changes position not only of a pair jobs, but can also change position of every job in the entire sequence in just one execution. See Table 6 for comparison of permutation schemes.

Table 6
Comparison of the perturbation schemes

Permutation scheme	Neighborhood size	Transition path length	Number of positions changed
Interchanging two adjacent jobs	$n-1$	$O(n^2)$	2
Interchanging two jobs	$\frac{n(n-1)}{2}$	$O(n)$	2
Moving a single job	$(n-1)^2$	$O(n)$	$O(n)$

The Simulated Annealing algorithm has found 21 optimal solutions: 11 for 5-machine instances, 6 for 10-machine instances and 4 for 20-machine instances. The best result of the NEH algorithm is the solution of instance #12 – only 0.18% worse than optimal, while for the CDS algorithm it is 0.66% (instance #2). On the other hand, in the worst case of the SA algorithm, the makespan of instance #17 is only 1.14% higher than optimal, while for NEH it is 7.88% (instance #31) and for CDS it is 19.59% (instance #14). See Table 7 for detailed results of all the flowshop problem instances.
We have tested three types of starting conditions of the SA optimization process:

- Initial permutation is chosen at random.
- Solution generated by the CDS algorithm is taken as the initial permutation.
- Solution generated by the NEH algorithm is taken as the initial permutation.

As the SA algorithm finds solutions equal or better than both the CDS and the NEH algorithms approximately after half of the cycle or earlier, the initial permutation has little impact on the final solution. This property, however, makes the proposed design of SA algorithm self-sufficient.

6. Conclusions

We have presented an effective design of the Simulated Annealing algorithm for the flowshop problem with minimum makespan criterion and have shown that it outperforms both the CDS and NEH heuristics in terms of solution quality. Even though SA is not the fastest heuristic

Table 7
Detailed results of the Taillard flowshop problem instances

#	$n \times m$	Results				$\eta(SA,ET)$	$\eta(NEH,ET)$	$\eta(CDS,ET)$	$\eta(NEH,SA)$	$\eta(CDS,SA)$
		Taillard	SA	NEH	CDS	[%]	[%]	[%]	[%]	[%]
1	20×5	1278	1278	1286	1334	0.00	0.63	4.38	0.63	4.38
2	20×5	1359	1359	1365	1368	0.00	0.44	0.66	0.44	0.66
3	20×5	1081	1081	1159	1253	0.00	7.22	15.91	7.22	15.91
4	20×5	1293	1293	1325	1409	0.00	2.47	8.97	2.47	8.97
5	50×5	2724	2724	2733	2934	0.00	0.33	7.71	0.33	7.71
6	50×5	2834	2834	2843	3020	0.00	0.32	6.56	0.32	6.56
7	50×5	2621	2621	2640	2856	0.00	0.72	8.97	0.72	8.97
8	50×5	2751	2751	2782	2843	0.00	1.13	3.34	1.13	3.34
9	100×5	5493	5493	5519	5901	0.00	0.47	7.43	0.47	7.43
10	100×5	5268	5268	5348	5466	0.00	1.52	3.76	1.52	3.76
11	100×5	5175	5175	5219	5378	0.00	0.85	3.92	0.85	3.92
12	100×5	5014	5018	5023	5303	0.08	0.18	5.76	0.10	5.68
13	20×10	1582	1582	1680	1771	0.00	6.19	11.95	6.19	11.95
14	20×10	1659	1659	1729	1984	0.00	4.22	19.59	4.22	19.59
15	20×10	1496	1496	1557	1735	0.00	4.08	15.98	4.08	15.98
16	20×10	1377	1377	1439	1547	0.00	4.50	12.35	4.50	12.35
17	50×10	2991	3025	3135	3386	1.14	4.81	13.21	3.64	11.93
18	50×10	2867	2887	3032	3306	0.70	5.76	15.31	5.02	14.51
19	50×10	2839	2852	2986	3243	0.46	5.18	14.23	4.70	13.71
20	50×10	3063	3063	3198	3565	0.00	4.41	16.39	4.41	16.39
21	100×10	5770	5770	5846	6255	0.00	1.32	8.41	1.32	8.41
22	100×10	5349	5352	5453	6004	0.06	1.94	12.25	1.89	12.18
23	100×10	5676	5679	5824	6155	0.05	2.61	8.44	2.55	8.38
24	100×10	5781	5812	5929	6461	0.54	2.56	11.76	2.01	11.17
25	20×20	2297	2297	2410	2587	0.00	4.92	12.63	4.92	12.63
26	20×20	2099	2099	2150	2351	0.00	2.43	12.01	2.43	12.01
27	20×20	2326	2326	2411	2565	0.00	3.65	10.28	3.65	10.28
28	20×20	2223	2223	2262	2490	0.00	1.75	12.01	1.75	12.01
29	50×20	3850	3893	4082	4424	1.12	6.03	14.91	4.85	13.64
30	50×20	3704	3722	3921	4260	0.49	5.86	15.01	5.35	14.45
31	50×20	3640	3666	3927	4204	0.71	7.88	15.49	7.12	14.68
32	50×20	3723	3760	3969	4403	0.99	6.61	18.26	5.56	17.10
33	100×20	6202	6271	6541	7263	1.11	5.47	17.11	4.31	15.82
34	100×20	6183	6239	6523	7064	0.91	5.50	14.25	4.55	13.22
35	100×20	6271	6338	6639	7193	1.07	5.87	14.70	4.75	13.49
36	100×20	6269	6323	6557	7002	0.86	4.59	11.69	3.70	10.74

algorithm, the computation time on modern computers is acceptable. Furthermore, the design proposed in this paper is similar to the general design and can be easily adapted and used to solve other combinatorial problems (by just changing the value of estimated minimal difference between solutions).

References

[1] M. R. Garey, "The complexity of flowshop and jobshop scheduling", *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, 1976.

[2] H. G. Campbell, R. A. Dudek and M. L. Smith, "A heuristic algorithm of the *n*-job, *m*-machine sequencing problem", *Manag. Sci.*, vol. 16, pp. 630–637, 1970.

[3] M. Nawaz, E. Enscore Jr, and I. Ham, "A heuristic algorithm for the m-machine, n-job flowshop sequencing problem", *OMEGA Int. J. Manag. Sci.*, vol. 11, pp. 91–95, 1983.

[4] F. A. Ogbu and D. K. Smith, "The application of the simulated annealing algorithm to the solution of the $n/m/Cmax$ flowshop problem", *Comput. Oper. Res.*, vol. 17, no. 3, pp. 243–253, 1990.

[5] J. Hurkała and A. Hurkała, "Effective design of the simulated annealing algorithm for the flowshop problem with minimum makespan criterion", in *9th Int. Conf. Decision Support Telecomm. Inform. Society DSTIS 2011* , Warsaw, Poland, 2011.

[6] M. Wodecki and W. Bożejko, "Solving the flow shop problem by parallel simulated annealing", *LNCS*, vol. 2328, pp. 597–600, 2006.

[7] E. Taillard, "Some efficient heuristic methods for flow shop sequencing", *Eur. J. Oper. Res.*, vol. 47, pp. 65–74, 1990.

[8] J. Grabowski and J. Pempera, "New block properties for the permutation flow-shop problem with application in TS", *J. Oper. Res. Soc.*, vol. 52, pp. 210–220, 2001.

[9] E. Nowicki, "The permutation flow shop with buffers: a tabu search approach", *Eur. J. Oper. Res.*, vol. 116, pp. 205–219, 1999.

[10] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for the permutation flowshop problem", *Eur. J. Oper. Res.*, vol. 91, pp. 160–175, 1996.

[11] C. R. Reeves, "A genetic algorithm for flowshop sequencing". *Comput. Oper. Res.*, vol. 22, pp. 5–13, 1995.

[12] C. R. Reeves and T. Yamada, "Genetic algorithms, path relinking, and the flowshop sequencing problem", *Evol. Comput.*, vol. 6, no. 1, pp. 230–234, 1998.

[13] S. R. Hejazi and S. Saghafian, "Flowshop-scheduling problems with makespan criterion: a review", *Int. J. Prod. Res.*, vol. 43, no. 14, pp. 2895–2929, 2005.

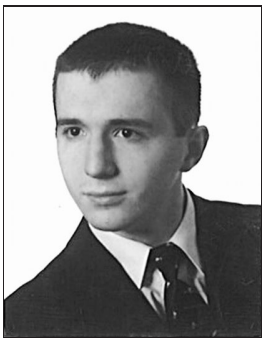
[14] S. Kirkpatrick, C. D. Gellat and M. P. Vecchi, "Optimization by simulated annealing", *Science*, vol. 220, pp. 671–680, 1983.

[15] V. Černý, "Thermodynamical approach to travelling salesman problem: An efficient simulation algorithm". *J. Optim. Theory Appl.*, vol. 45, pp. 41–51, 1985.

[16] C. Koulamas, S. R. Antony, and R. Jaen, "A survey of simulated annealing applications to operations research problems", *Omega*, vol. 22, no. 1, pp. 41–56, 1994.

[17] E. Taillard, "Benchmarks for basic scheduling problems", *Eur. J. Oper. Res.*, vol. 64, pp. 278–285, 1993.

Jarosław Hurkała received his M.Sc. degree in Computer Science with honors from the Warsaw University of Technology, Poland, in 2010. Currently, he is a Ph.D. student in the Institute of Control and Computation Engineering at the Warsaw University of Technology. His research area focuses on scheduling problems, heuristic algorithms, fairness and multicriteria optimization.
E-mail: j.hurkala@elka.pw.edu.pl
Institute of Control and Computation Engineering
Warsaw University of Technology
Nowowiejska st 15/19
00-665 Warsaw, Poland



Adam Hurkała received his M.Sc. degree in Computer Science with honors from the Warsaw University of Technology, Poland, in 2010. Currently, he is a Ph.D. student in the Institute of Control and Computation Engineering at the Warsaw University of Technology. His research area focuses on information security and cryptography.
E-mail: a.hurkala@elka.pw.edu.pl
Institute of Control and Computation Engineering
Warsaw University of Technology
Nowowiejska st 15/19
00-665 Warsaw, Poland