Bartosz PIASECKI, Jerzy JÓZEFCZYK
Politechnika Wrocławska

EVOLUTIONARY ALGORITHM FOR JOINT TASK SCHEDULING AND DEPLOYMENT OF EXECUTORS

Summary. The basis for considerations is the classical problem of task scheduling on many identical executors with different release dates and the total completion time as the objective function. There has been considered its generalization considering the spatial location of both jobs and executors as well as leading to two dependent subproblems: task scheduling and deployment of executors. An evolutionary algorithm has been proposed. Its evaluation via numerical experiments is provided including the comparison with the performance of the CPLEX CP Optimizer solver.

ALGORYTM EWOLUCYJNY DLA ŁĄCZNEGO PROBLEMU SZEREGOWANIA ZADAŃ I ROZMIESZCZENIA REALIZATORÓW

Streszczenie. Podstawą rozważań jest klasyczne zagadnienie szeregowania zadań na wielu identycznych realizatorach z różnymi momentami gotowości i kryterium w postaci sumy terminów zakończenia zadań. Rozpatrzono jego uogólnienie, polegające na uwzględnieniu przestrzennej lokalizacji zarówno zadań jak i realizatorów i prowadzące do dwóch zależnych podproblemów: szeregowania zadań i rozmieszczenia realizatorów. Zaproponowano ewolucyjny algorytm rozwiązania i eksperymentalnie oceniono jego jakość poprzez porównanie z działaniem solvera CPLEX CP Optimizer.

1. Introduction

Nowadays, the task scheduling, as well as the facility location called in this work also the deployment of executors (machines) are widely known optimal decision-making problems which are crucial elements in strategic planning for logistic systems, e.g. [1,5,12]. They play a significant role in production and manufacturing systems as well as in information processing environments. A sequential solving of both problems is the most common approach when the deployment of executors is followed by the task scheduling. Then, locations of executors as solutions of a facility location problem with a distance-based criterion are data for the classical task scheduling. However, a simple analysis shows that the optimal solution of the joint problem, when the task scheduling criterion is only taken into account, may not be composed of optimal results of mentioned individual issues.

Such a generally outlined joint problem of task scheduling and deployment of executors referred to as ScheLoc (Scheduling and Location) has been firstly presented in [7]. It can also be defined as a particular case of task scheduling problems with different release dates, the value of which depends on the location of the executor (a single executor has been only taken into account). The approach with discrete executor positions presented in [7] is based on the graph model, the vertices of which represent the job storage locations. The algorithm of the polynomial complexity developed there used the ERD (Earliest Release Date) rule and was limited to the case with the makespan as the objective function. In [3] and [4] a polynomial algorithm for solving the ScheLoc problem in continuous space to minimize the makespan was proposed. Its authors also used the ERD rule as the solution algorithm. A geometric method based on the branch and bound technique can be found in [15]. A solution similar to this one is presented in [10]. It is based on the BTST (Big Triangle Small Triangle) algorithm [2] and allows obtaining results for cases of the joint task scheduling and deployment of executors with the makespan or the total completion time as the criterion. The case studies with many identical executors were carried out by the authors of [14]. The proposed mathematical model allows finding a solution in both continuous and discrete space for the makespan as the criterion. The unloading of ships by cranes can be mentioned as an example of real-world applications. Ships waiting for the unloading at sea can be treated as jobs. Locations at an embankment for cranes as executors need the determination as well as the order of service of ships after their reaching the embankment.

In this paper, the new version of ScheLoc is considered with many executors and the sum of completion times as the criterion. It directly refers to [14] and [10]. The differences consist in the criterion and the number of executors. The makespan and a single executor as the only differences to our problem are assumed in [14] and [10], respectively. Moreover in this paper, an evolutionary algorithm is presented that allows solving the considered joint problem.

The reminder of the paper is organized as follows. The next section provides the mathematical model of the considered joint optimization problem. Then in Sections 3 and 4, the description of the heuristic evolutionary algorithm and its evaluation via series of numerical experiments are respectively given. Final remarks complete the paper. This work comprises selected results widely presented in the master thesis [12].

2. Problem formulation

Let us specify a set of n jobs $J = \{J_1, J_2, \dots, J_j, \dots, J_n\}$. Each job has to be performed without preemptions by a single executor (machine) taken from a set $M = \{M_1, M_2, \dots, M_i, \dots, M_m\}$ of m identical executors (machines). It is assumed that each job J_j has a fixed location $a_j = [a_j^{(1)}, a_j^{(2)}] \in \mathbb{R}^2$ and is characterized by the processing time $p_j > 0$, the ready time $\sigma_j \geq 0$, and the travel speed $v_j > 0$, i.e. the rate of the position change expressed as a distance per unit time. For the executors, their locations $x_i = [x_i^{(1)}, x_i^{(2)}] \in \mathbb{R}^2$, $i \in \{1, 2, \dots, m\}$ have to be determined. It is assumed that all executors have to be located within the defined area $S = \{x = [x^{(1)}, x^{(2)}] \in \mathbb{R}^2 \mid x_{min}^{(1)} \leq x^{(1)} \leq x_{max}^{(1)} \land x_{min}^{(2)} \leq x^{(2)} \leq x_{max}^{(2)}\}$, $x_{min}^{(1)}, x_{max}^{(1)}, x_{min}^{(2)}, x_{max}^{(2)} \in \mathbb{R}$.

Assuming that $d(a_j, x_i) = \sqrt{\left(a_j^{(1)} - x_i^{(1)}\right)^2 + \left(a_j^{(2)} - x_i^{(2)}\right)^2}$ is the distance between the location of jth job and the position of ith executor, the variable release dates can be calculated as follows

$$r_j(x_i) = \sigma_j + \frac{1}{v_j} d(a_j, x_i), \forall i \in \{1, 2, \dots, m\}, \forall j \in \{1, 2, \dots, n\}. \tag{1}$$

The sequence of jobs to be performed on the ith executor can be defined by the permutation $\pi_i = \left(\pi_i^{(1)}, \pi_i^{(2)}, \dots, \pi_i^{(l_i)}\right)$ where $\sum_{i=1}^{m} l_i = n$, and $0 \le l_i \le n$ denotes the number of jobs performed by this executor. The entry $\pi_i^{(k)} = j$ means that the job J_j is performed by executor M_i as the kth. Two sets $X = \{x_1, x_2, \dots, x_m\}$ and $\Pi = \{\pi_1, \pi_2, \dots, \pi_m\}$ are decision variables. The completion times for all jobs can be calculated using the following recursive formula

$$C_{\pi_i^{(1)}}(x_i) = r_{\pi_i^{(1)}}(x_i) + p_{\pi_i^{(1)}}, \forall i \in \{1, 2, \dots, m\}, \tag{2}$$

$$C_{\pi_i^{(k)}}(x_i) = \max\left\{C_{\pi_i^{(k-1)}}(x_i), r_{\pi_i^{(k)}}(x_i)\right\} + p_{\pi_i^{(k)}}, \; i = \overline{1, m}, \; k = \overline{2, n}. \tag{3}$$

The total completion time $\sum C_j$ has been chosen as the objective function $f(X, \Pi)$ which evaluates decisions X and Π. It can be expressed as follows

$$f(X, \Pi) = \sum_{i=1}^{m} \sum_{k=1}^{l_i} C_{\pi_i^{(k)}}(x_i). \tag{4}$$

Hence, the described optimization problem in Graham's triplet notation [6] can be written as $Pm \mid r_j(x) \mid \sum C_j$.

Let us denote as DP all data of the problem, i.e., set of jobs J, set of executors M, job locations a_j, processing times p_j, ready times σ_j, travel speeds v_j, and area S for the deployment of executors. Then, the considered optimization problem deals for given DP with finding such a pair (X^*, Π^*) which minimize (4), i.e., $f^* \triangleq f(X^*, \Pi^*) = \min_{(X,\Pi)} f(X, \Pi)$.

Proposition. *$Pm \mid r_j(x) \mid \sum C_j$ is strongly NP-hard optimization problem.*

Proof. The NP complexity in a strong sense results immediately from such property for both sub-problems. Namely, $Pm \mid r_j \mid \sum C_j$ and the multifacility location problem are strongly NP-hard, as it is reported in [13] and [5], respectively. □

The $Pm \mid r_j(x) \mid \sum C_j$ problem can also be described using mixed-integer linear programming model based on the mathematical model included in the paper [11]. Then, binary optimization variables are defined, which are elements of the three-dimensional decision matrix $W = \left[w_{jik}\right]_{j=1,2,\dots,n; i=1,2,\dots,m; k=1,2,\dots,n}$, where $w_{jik} = 1$, if the jth job is scheduled on the ith executor as the kth and $w_{jik} = 0$ otherwise. Therefore, each executor has n virtual positions on which jobs can be performed. In addition, there is also defined the vector of continuous variables $X = [x_1, x_2, \dots, x_m]^T$, whose elements represent the positions of each executor and the matrix of variables $C = [C_{ik}]_{i=1,2,\ldots,m;k=1,2,\ldots,n}$, where the element C_{ik} is the completion time of a job which is performed by the ith executor as the kth.

The objective function in this case is expressed as

$$f(C) = \sum_{i=1}^{m} \sum_{k=1}^{n} C_{ik}, \tag{5}$$

and the following constraints are imposed on decision variables

$$\sum_{i=1}^{m} \sum_{k=1}^{n} w_{jik} = 1, \forall j \in \{1, 2, \ldots, n\}, \tag{6}$$

$$\sum_{j=1}^{n} w_{jik} \leq 1, \forall i \in \{1,2, \ldots, m\}, \forall k \in \{1, 2, \ldots, n\}, \tag{7}$$

$$C_{ik} \geq \sum_{j=1}^{n} \big(r_j(x_i) + p_j\big) w_{jik}, \forall i \in \{1,2, \ldots, m\}, \forall k \in \{1, 2, \ldots, n\}, \tag{8}$$

$$C_{ik} \geq C_{i(k-1)} + \sum_{j=1}^{n} p_j w_{jik}, \forall i \in \{1, 2, \ldots, m\}, \forall k \in \{2, 3, \ldots, n\}, \tag{9}$$

$$C_{ik} \geq 0, \forall i \in \{1, 2, \ldots, m\}, \forall k \in \{1, 2, \ldots, n\}, \tag{10}$$

$$x_i \in S, \forall i \in \{1, 2, \ldots, m\}, \tag{11}$$

$$w_{jik} \in \{0, 1\}, \forall i \in \{1, 2, \ldots, m\}, \forall j, k \in \{1, 2, \ldots, n\}. \tag{12}$$

Constraints (6) and (7) ensure that each job is performed on exactly one position of exactly one executor and a maximum of one job can be performed on each position of each executor, respectively. Moreover, in a feasible solution, the kth in order job on the ith executor cannot start before its release date or the completion time of a job which is scheduled on the $k-1$ position of the same executor. This condition is met by constraints (8) and (9). Finally, constraints (10), (11) and (12) define the domains of all decision variables.

Hence, the joint task scheduling and deployment of executor deals for given DP with finding such a vector X and matrix W, which satisfy constraints (6)–(12), to minimize (5).

3. Evolutionary algorithm

Such high computational complexity of the considered problem justifies the searching of heuristic methods that allow receiving solutions in an acceptable time. Consequently, the evolutionary approach has been employed for the elaboration of the heuristic solution algorithm.

The representation of the problem in the developed evolutionary algorithm consists of two parts. The first one is the genotype stored as a sequence $G = \big(g_1, g_2, \ldots, g_{2n}, g_{2n+1}, \ldots, g_{2(n+m)}\big)$, $g_k \in [0, 1]$, $k \in \{1, 2, \ldots, 2n, 2n+1, \ldots, 2(n+$

$m)\}$. Each two elements g_{2j-1} and g_{2j}, $j \in \{1, 2, \dots, n\}$ represent the jobs, and they are respectively the priority of processing this job and the number of assigned executor. The pairs of elements $g_{2(n+i)-1}$ and $g_{2(n+i)}$, $i \in \{1, 2, \dots, m\}$ express the values of coordinates of the ith executor's location after the normalization to the interval $[0, 1]$.

Due to the fact that the case of ScheLoc problem with a single executor does not require the assignment of executors to jobs, the genotype is then reduced to sequence $G = (g_1, g_2, \dots, g_{n+2})$. Hence, the elements from g_1 to g_n denote the priorities of corresponding jobs, and g_{n+1} and g_{n+2} are the values of the first and the second coordinate of the executor's location after the normalization to the interval $[0, 1]$.

The second part of the problem representation is the phenotype that is the pair (X, Π) consisting of the set of points X at a plane, in which executors are located, and sequences Π indicating the order of performing the jobs by executors.

Transformation of the genotype into a phenotype consists of two stages. First of all, the first $2n$ elements from the genotype have to be split into pairs of elements (g_{2j-1}, g_{2j}), $j \in \{1, 2, \dots, n\}$. Next, they are divided into m sequences based on the value of the second component of each pair. If the inequality $g_{2j} \le \frac{i}{m}$, $\forall i \in \{1, 2, \dots, m\}$, $\forall j \in \{1, 2, \dots, n\}$ is satisfied, the ith executor is assigned to the jth job. Next, the elements of each sequence are sorted in the non-increasing order based on the value of element g_{2j-1}, $j \in \{1, 2, \dots, n\}$. For the case with one executor, the assignment phase is omitted, and the order of jobs is determined according to the non-increasing order of the corresponding values of the genotype.

The second stage of the transformation is related to the conversion of the values of the next $2m$ elements of the genotype to the actual coordinates of the location of all executors. This is done using the following expressions

$$x_i^{(1)} = g_{2(n+i)-1} * \left(x_{max}^{(1)} - x_{min}^{(1)}\right) + x_{min}^{(1)}, \forall i \in \{1, 2, \dots, m\}, \tag{13}$$

$$x_i^{(2)} = g_{2(n+i)} * \left(x_{max}^{(2)} - x_{min}^{(2)}\right) + x_{min}^{(2)}, \forall i \in \{1, 2, \dots, m\}. \tag{14}$$

The initialization of the evolutionary algorithm consists in the random generation of $N_{Pop} > 0$ solutions in the form of the genotype. Their features are encoded as real numbers obtained from continuous uniform distribution $U(0, 1)$. The selection method is based on the tournament operator of $N_{Tur} > 1$ individuals. The definition of the crossover operator uses the BLX-α (Blend Crossover) method which was described in [8]. It involves the generation of two offspring individuals based on parental solutions through drawing values from the continuous uniform distributions which ranges for $\alpha \in \mathbb{R}_+$ are determined in the following way

$$\left[g_k^{min} - \alpha\Delta g_k, g_k^{max} + \alpha\Delta g_k\right], \forall k \in \{1, 2, \dots, |G|\}, \tag{15}$$

$$\Delta g_k = g_k^{max} - g_k^{min}, \forall k \in \{1, 2, \dots, |G|\}, \tag{16}$$

$$g_k^{min} = \min\left\{g_k^{(1)}, g_k^{(2)}\right\}, \forall k \in \{1, 2, \dots, |G|\}, \tag{17}$$

$$g_k^{max} = \max\left\{g_k^{(1)}, g_k^{(2)}\right\}, \forall k \in \{1, 2, \dots, |G|\}. \tag{18}$$

If the obtained boundary values go beyond the domain of genotype elements, they are appropriately equated to 0 or 1. The entire procedure is performed after fulfilling the crossover probability $\mu_c \in [0,1]$ condition. The mutation of the solution consists in iterating all elements of the genotype and drawing a new value from the continuous uniform distribution $U(0,1)$ only after the mutation probability $\mu_m \in [0,1]$ condition is fulfilled. The evolutionary algorithm stops after elapsing $N_{Gen} > 0$ generations or when the maximum number of generations without improving the best-obtained result N_{const} is exceeded. The latter value is computed by the following formula

$$N_{const} = \lceil u_{const} * N_{Gen} \rceil, \tag{19}$$

where $u_{const} \in [0,1]$ is an additional percentage parameter of the algorithm.

Assuming that the parameters of the evolutionary algorithm are fixed, it has a computational complexity equals to $O(nlogn + nm)$. It is also simplified to the linear-log time complexity $O(nlogn)$ when solving a problem with a single executor. The operation of the entire developed algorithm is shown as the following pseudocode.

Algorithm 1 (evolutionary algorithm)

```
Require: $N_{Gen}$, $N_{Pop}$, $N_{Tur}$, $\alpha$, $\mu_c$, $\mu_m$, $u_{const}$, $n$, $m$ and data of the problem $DP$
Ensure: $f^{\mathrm{EVO}}$, $X^{\mathrm{EVO}}$, $\Pi^{\mathrm{EVO}}$
1:  Let $N_{const}$ be equal to the ceiling value of $N_{Gen}$ multiplied by $u_{const}$.
2:  Set $k_{Gen} \leftarrow 0$.
3:  Set $k_{const} \leftarrow 0$.
4:  Set $Population \leftarrow Initialize(N_{Pop}, n, m)$.
5:  Set $(X^{\mathrm{EVO}}, \Pi^{\mathrm{EVO}}) \leftarrow GetBestPhenotype(Population, DP)$
6:  Set $f^{EVO} \leftarrow f(X^{EVO}, \Pi^{EVO})$.
7:  while $k_{Gen} < N_{Gen}$ and $k_{const} < N_{Const}$ do:
8:      set $Population \leftarrow CreateGeneration(Population, N_{Pop}, N_{Tur}, \alpha, \mu_c, \mu_m)$,
9:      set $(X, \Pi) \leftarrow GetBestPhenotype(Population, DP)$,
10:     if $f(X,\Pi) < f(X^{\mathrm{EVO}}, \Pi^{\mathrm{EVO}})$ then:
11:         set $(X^{\mathrm{EVO}}, \Pi^{\mathrm{EVO}}) \leftarrow (X, \Pi)$,
12:         set $f^{EVO} \leftarrow f(X, \Pi)$,
13:         set $k_{const} \leftarrow 0$,
14:     else
15:         set $k_{const} \leftarrow k_{const} + 1$,
16:     end if
17:     $k_{Gen} \leftarrow k_{Gen} + 1$,
18: end while
```

4. Experimental evaluation

The goal of the performed numerical experiments was to assess the quality of the obtained solutions and the algorithm's computational time for different values of selected problem's parameters. In particular, the comparison with corresponding results generated by the CPLEX CP Optimizer solver ([9]) has been performed. All required calculations were carried out on the computer with the 2,4GHz i7-3630QM processor equipped with 8GB of RAM. Values of the objective function f and the time of operation t as evaluations of both algorithms were collected. They were then averaged out of 500 repetitions for the evolutionary algorithm. The work of the solver was interrupted after 1 hour. Additionally, the values of N_{Gen}, N_{Pop}, N_{Tur}, α, μ_c, μ_m and u_{const} parameters of the evolutionary algorithm were tuned for each data set according to the function that had been prepared earlier and described in [12]. During the comparative study, the relative percentage difference between the solutions obtained by the solver and the evolutionary algorithm was calculated using the following performance index $\delta = (f^{\mathrm{EVO}} - f^{\mathrm{CP}})/f^{\mathrm{CP}} \cdot 100\%$ where f^{CP} and f^{EVO} are the values of the objective function obtained by the used solver and the evolutionary algorithm, respectively. The dependence of δ and t on n and m has been verified during the experiment.

The research was carried out on randomly generated data sets according to the continuous uniform distribution. By default, $n = 100$ and $m = 2$ were assumed. The storage locations a_j of jobs were generated in the grid layout on the plane $[0, 1000] \times [0, 1000]$, the processing times p_j were selected from the interval $[1, 51]$ and the storage arrival times were set to $\sigma_j = 0$ for all jobs. The travel speeds v_j were calculated using the parameter $\gamma = 1$, which denotes the coefficient of approximation of travel and performing times for each job. This was done by the following expression

$$v_j = \frac{1}{\gamma p_j} d\left(a_j, c(S)\right), \forall j \in \{1, 2, \dots, n\} \tag{20}$$

where $c(S)$ is the central point of the executors deploying area S which was also set as constant and equals $S = \left\{\left[x^{(1)}, x^{(2)}\right] \mid x^{(1)} \in [347.16, 637.91] \wedge x^{(2)} \in [207.23, 553.74]\right\}$. The results are presented in Tables 1 and 2. They show that the evolutionary algorithm works relatively repetitively in terms of the received objective function values. This is confirmed by the obtained values of standard deviations which are on average at the level of $1 - 2\%$ of the average values of the criterion. Comparing the quality of the proposed algorithm in relation to the CPLEX CP Optimizer solver, it can be noticed that for $n \geq 20$ it obtained on average better solutions than the solver, and these differences are rapidly growing in favor of the evolutionary algorithm as the number of jobs increases. Also, the solver is characterized by a very fast increase in the computational time, which for $n \geq 12$ met a time limitation on the used platform, so that it was not able to return the optimal solutions. Whereas, the evolutionary algorithm, even for $n = 100$, can cope with this problem in about 8 seconds.

Table 1

Dependence of f, t, and δ on n

n	f^{EVO}		$t^{EVO}[s]$		f^{CP}	$t^{CP}[s]$	δ
	AVG	STD	AVG	STD			
6	329.95	7.20	1.16	0.33	325.79	94.22	1,28%
8	455.04	22.00	1.34	0.40	434.53	29.64	4,72%
10	565.95	9.08	1.48	0.42	539.41	565.38	4,92%
12	740.33	11.87	1.60	0.52	721.65	3600.03	2,59%
14	960.82	15.22	1.77	0.60	936.07	3600.03	2,64%
16	1162.42	16.20	2.00	0.62	1133.88	3600.05	2,52%

18	1515.56	16.67	2.16	0.75	1489.09	3600.05	1,78%
20	1998.51	19.34	2.39	0.79	2013.26	3600.06	-0,73%
40	6278.21	27.87	4.61	1.45	6409.91	3600.20	-2,05%
60	14820.04	47.19	6.77	1.63	15195.42	3600.47	-2,47%
80	26251.32	67.27	6.86	1.61	33288.86	3600.87	-21,14%
100	42620.07	116.20	7.89	1.91	66831.53	3601.25	-36,23%

Table 2

Dependence of f and t on m for the evolutionary algorithm

m	f^{EVO}		$t^{EVO}[s]$	
	AVG	STD	AVG	STD
1	104228.59	40.73	7.05	0.60
2	53254.13	120.57	7.97	1.82
3	36414.41	169.97	7.23	1.68
5	23072.40	197.21	6.83	1.69
8	15695.99	201.88	6.35	1.60
10	13286.60	195.27	6.31	1.65
15	10115.38	174.48	6.28	1.60
20	8602.81	171.58	6.31	1.65
25	7738.44	152.36	6.38	1.71
30	7172.22	144.59	6.58	1.75
40	6514.16	129.64	6.97	1.96
50	6133.97	109.85	7.56	1.94

As the conclusions of the experiments, it is worth noting that growing the number of jobs n increases values of both the criterion and the computational time. The employment of more number of executors results in decreasing the value of criterion which is initially rapid, but it slows down for $m \geq 15$. However, this is connected with increased computational times.

5. Final remarks

This paper investigates the joint problem of task scheduling and deployment of many identical executors in a two-dimensional continuous space. There was proposed the evolutionary algorithm allowing for solving this computationally difficult problem. As a result of the experimental evaluation, it was shown that the developed evolutionary algorithm is only slightly worse than the exact algorithm represented by the solver. Additionally, it can give even better result when the operation of the exact algorithm is restricted to one hour.

Searching for other more useful algorithms can be pointed out as the direction of further work. The development of the proposed evolutionary algorithm for other cases is also planned, e.g., for uniform and unrelated executors. Moreover, the proposed evolutionary algorithm may be adapted for the usage in stochastic models that can better represent real-world applications.

REFERENCES

1. Drezner Z. (Ed.): Facility Location. A survey of Applications and Methods. Springer-Verlag New-York, 1995.
2. Drezner Z., Suzuki A.: The Big Triangle Small Triangle Method for the Solution of Nonconvex Facility Location Problems. Operations Res., 52(1), 2004, p. 128-135.
3. Elvikis D., Hamacher H. W., Kalsch M. T.: Scheduling and Location (ScheLoc): Makespan Problem with Variable Release Dates. Univ. of Kaiserslautern, 2007.
4. Elvikis D., Hamacher H. W., Kalsch M. T.: Simultaneous scheduling and location (ScheLoc): The planar ScheLoc makespan problem. Journal of Scheduling, 12 (4), 2009, p. 361- 374.
5. Farahani R. Z., Hekmatfar M.: Facility Location: Concepts, Models, Algorithms and Case Studies, Heidelberg, Physica, 2009.
6. Graham R. L., Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G.: Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. Annals of Discrete Mathematics, Elsevier, 5, 1979, p. 287-326.
7. Hennes H., Hamacher H. W.: Integrated Scheduling and Location Models: Single Machine Makespan Problems. Report in Wirtschaftsmathematik, 82, 2002.
8. Herrera F., Lozano M., Sánchez A. M.: A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. International Journal of Intelligent Systems, 18 (3), 2003, p. 309-338.
9. IBM: CPLEX CP Optimizer. https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-cp-optimizer (available at 17.03.2018).
10. Kalsch M. T., Drezner Z.: Solving scheduling and location problem in the plane simultaneously. Computers & Operations Res., Elsevier, 37(2), 2010, p. 256-264.
11. Kooli A., Serairi M.: A mixed integer programming approach for the single machine problem with unequal release dates. Computers & Operations Res., Elsevier, 51, 2014, p. 323-330.
12. Piasecki B.: Application of AI-based algorithms for joint problem of task scheduling and deployment of executors (in Polish). Master Thesis. Wroclaw University of Science and Technology, Wrocław 2018.
13. Pinedo M. L.: Scheduling: Theory, Algorithms, and Systems. Springer-Verlag New York, Nowy Jork, 2012.
14. Rajabzadeh M., Ziaee M., Bozorgi-Amiri A.: Integrated approach in solving parallel machine scheduling and location (ScheLoc) problem. International Journal of Industrial Engineering Computations, 7(4), 2016.
15. Scholz D.: Deterministic Global Optimization: Geometric Branch-and-bound Methods and their Applications. Springer New York, 2012.