Simulation in Produktion
und Logistik 2017
Sigrid Wenzel & Tim Peter (Hrsg.)
kassel university press, Kassel 2017

Design of Project-oriented Calculation Models for Job Priorities by Using a Customized Genetic Algorithm

Entwicklung projektspezifischer Berechnungsmodelle für Auftragsprioritäten mit einem Genetischen Algorithmus

Thorsten Schmidt, Mathias Kühn, Paul Richard Genßler, TU Dresden, Dresden (Germany), thorsten.schmidt@tu-dresden.de, mathias.kuehn@tu-dresden.de, paul.genssler@tu-dresden.de

Abstract: In this paper, the resource-constrained multi-project scheduling problem (RCMPSP) is solved by using project-oriented calculation models for job priorities. Many studies show that effectiveness of simple priority rules depends on the problem class and type of instance. Better results can be achieved with individually designed calculation models for job priorities such as the approach for weighted sum of priority rules used in this work. The focus of the paper is an extension of this approach where each weight is computed project specifically by a customized genetic algorithm (GA) using the theory of simulation based optimization. The algorithm is implemented in a specially developed python based scheduling and optimization framework. For evaluation, we compared our concept to simple priority rules and other heuristics over selected benchmark sets with the objective of total makespan. Results show a similar or better performance and robustness on different models.

1 Introduction

Simple Priority Rules (SPR), e.g. Earliest Due Date (EDD) or First-In-First-Out (FIFO), are often used for scheduling. Manufacturers who produce every time the same or similar products, SPR can be quite effective (Kaban et al. 2012; Pinedo 2009). For decision-making in complex and flexible production environments, e.g. with frequently changing production programs, SPR provide inefficient results (Tay and Ho 2008; Pinedo 2009). With such observations it can be concluded that the performance of SPRs greatly varies with the increase in problem complexity and instance type. Due to the global trend towards customization resulting in higher production requirements and thus increasing complexity, alternative methods for solving the scheduling problem are becoming more important to guarantee efficiency.

In this paper, we focus on solving resource-constrained multi-project scheduling problems (RCMPSP) with an extension of the approach of weighted sum of priority rules. This approach can be assigned to the category of dynamic priority rules in which the schedule is determined by updating the job priority. The job priority is calculated by combining different priority rules normalized with the weighted sum approach. In general, the weight sets for calculation of the priority are identically for each job. In our extension, we generate an individual weight set for each project by using a customized genetic algorithm (GA). We have chosen this approach because of high practical relevance and better performance for job floor control in comparison with other approaches such as genetic programming (Kuczapski et al. 2010). For the problem class of RCMPSP exists a benchmark library (Homberger 2017), which allows us to verify our approach.

2 Problem Definition: Resource-constrained Multi-project Scheduling Problem (RCMPSP)

The considered extension of the resource-constrained project scheduling problem (RCPSP) (Brucker and Knust 2012) to the resource-constrained multi-project scheduling problem (RCMPSP) can be stated as follows (Homberger 2007):

The given set of information includes a time horizon *[0, T]*, a set of *I* projects numbered from *1* to *m* which have to be planned simultaneously and a set of J_i of non-preemptable jobs for each project $I \in I$. The tasks in J_i are numbered from $n_{i-1}+1$ to n_i, with $n_0 = 0$ and where $n_{i-1} + 1$ and n_i are dummy jobs. The dummy jobs need no resources and have processing time zero. Furthermore, precedence constraints are defined between some jobs. They are given by a set A_i of finish-start precedence relations between these activities. A release date rd_i for each project $I \in I$ is given, which describes the earliest time when project *I* can start. Project *1* arrives at $rd_1 = 0$, while the others arrives at $rd_i \geq 0$. Furthermore, a set L_i of local resources for each project $I \in I$ is given. In addition, a set *G* of global renewable resources $(|G| \geq 1)$ is given, which is shared by all projects. Job *j* must be processed for p_j time units. A constant capacity amount of resource $l_i \in L_i$ units, denoted by $rl_i \in IM$ and/or a constant capacity amount of resource $g_o \in G$ units, denoted by $rg_o \in IM$ is occupied by the job j, $1 \leq j \leq n_m$, during this time period, where $0 \leq rl_i \leq al_i$; al_i denotes the constant amount of capacity units of $l_i \in L_i$ available in each time period $t = 0, \ldots, T$ and $0 \leq rg_o \leq ag_o$; ag_o denotes the constant amount of capacity units of $g_o \in G$ available in each time period $t = 0, \ldots, T$. The vector $S_i(f_{n_{i-1}+1}, \ldots, f_{n_i})$ is referred to the finish times of all jobs in J_i and describes a schedule of project *i*. S_i is called feasible if all resource and precedence constraints are fulfilled. The start time of the first dummy job is also the start time of the project and is denoted as $s_{n_{i-1}+1}$. The finish time f_{n_i} of the last dummy job is also the finish time of the project. Information asymmetry is not assumed.

We chose the objective to minimize the total makespan (TMS) (Eq. 1):

$$TMS = max\{f_{n_i}\} - min\{s_{n_{i-1}+1}\} \tag{1}$$

where, f_{n_i} is the finish time of a job and $s_{n_{i-1}+1}$ is the arrival time of a job in the system.

3 State of the Art

Whenever multiple jobs are queued, the jobs must be prioritized in order to assign limited resources. To determine the job with the highest priority, dispatching rules are often used. In general, priority rules can be mainly classified into two types (Panwalkar and Iskander 1977; Tay and Ho 2008): Simple Priority Rules (SPR) and Composite Priority Rules (CPR).

SPRs are characterized by the use of one job attribute for sequencing. Based on the job attributes, SPRs can be divided into different classes (Bloech 2014). With regard to the temporal influence on a priority, SPRs can be characterized into static and dynamic rules. While for static rules (e.g. SPT-shortest processing time) the priority in a queue does not change, for dynamic rules (CR-critical ratio) the priority is recalculated whenever the capacity of a resource changes. A further division is based on the required information requirements. Local rules (e.g. EDD-earliest due date) use only information on the resource in question. Global rules (e.g. WINQ-least work in next queue) use information from the entire system. There are also classifications according to the mode of action of SPRs. Research on SPRs is being published on regular basis, which is why these rules are listed in the state of the art with their simplicity and practical relevance. However, no rule can be categorized as the dominating one. To balance this disadvantage, many approaches were developed which combine good characteristics of SPR-the so called CPR. The design process can be done in different ways (see review Branke et al. 2016) . A popular way is the generating of CPR with genetic programming (GP). Tay and Ho (2008) introduced a GP-Framework for solving the flexible job shop problem. They used seven different terminals (e.g. job attribute due date) and five different functions. Results show, that the CPR outperforms the SPR. With this approach, the rule length and thus, rule computation time could be very long. Hildebrandt and Freitag (2015) took up this point and investigate the generation of a practical rule length dependent on the objective. They concluded that results of short rule lengths are not significant worse than results of long rule lengths. However, a general conclusion which rule length fits to acceptable results is impossible without an exploration of the solution space. This leads to a high effort with constantly changing products. Another popular way is the linear combination of SPR with computed weights. Ma et al. (2012) calculate the weights over a matrix consisting of the results for different objective functions when applying a priority rule. With application of this calculation model, the result is exactly one weight set. So this procedure is not an optimization and full potential is not exploited. An interesting approach for evolutionary computation of weighted priority rules was proposed by Kuczapski et al. (2010). They used a customized genetic algorithm to compute weights for generating near optimal initial populations. An individual of a population represents a list of priorities for each job which are assigned in advanced. Hence, with this method only static job attributes can be used (e.g. not dynamic attributes like waiting time or arrival time in a queue) and dynamic relationships between different projects cannot be considered. The results of the propose algorithm where compared to GP results and show significant better performance on objective values and computation time. Ingimundardottir and Runarsson (2016) are also using a genetic algorithm to compute the weights but calculate the job priority during the simulation. The results are promising, but it is computationally expensive. They are also using only a non-project specific calculation model. Attributes like waiting time are not considered.

We decided to extend the weighted sum approach because of the practicability (easy to understand priority rules) and also because of the promising results.

Our literature research revealed that there are no applications of the weighted sum approach for the RCMPSP where the calculation models are generated individually for each project. There is also potential of using more job attributes to get more benefits of the SPR (e.g. waiting time). Furthermore, for applying this approach, it is necessary to reduce runtime significantly. We address this requirement by developing a functional and slim simulation and optimization framework.

4 Solution Approach: Project-oriented Calculation Models for Job Priorities

4.1 Chromosome Representation

Each project has individual weights according to the job attributes for calculating the job priority. The weights represent a chromosome (Eq. 2; Eq. 3).

$$PIW_i = \sum_{e=1}^{E} pw_{ie} \tag{2}$$

$$\sum_{e=1}^{E} pw_{ie} = 1 \, , pw_{ie} \in R \tag{3}$$

where PIW_i is the project individual weight set of project i ($i=1 \ldots I$); E is the number of used job attributes PR_e and pw_{ie} is the project individual weight with a real number in the interval of [0, 1].

Each individual consists of chromosomes according to the number of projects I. A single chromosome consists of genes according to the number of job attributes PR_e. A gene represents the individual weight pw_{ie} (Fig. 1).

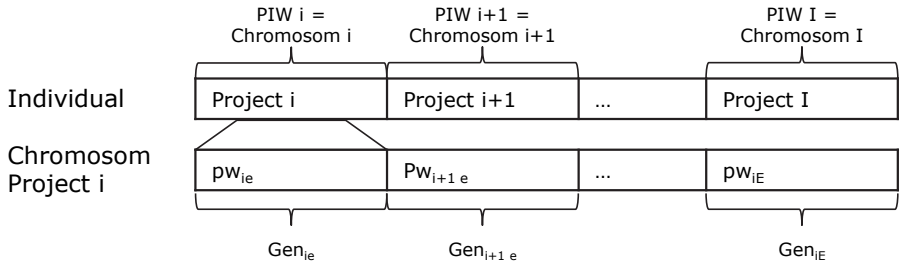

Figure 1: *Chromosome representation*

In this investigation, we use ten different job attributes PR_e (Tab. 1).

Table 1: *Job attributes (PR_e) for weighted sum approach*

Symbol	Ex-pression	Description
RR	rl_{j_i} / rg_{j_i}	Amount of required local / global resources of job j of project i
PT	p_{j_i}	Processing time of job j of project i
WT	w_{j_i}	Waiting time of job j of project i
IS	is_{j_i}	Amount of immediate successors of job j of project i
PJ	p_{j_i}	Amount of parallel jobs j in work in process of project i
PTS	$p_{(j+1)_i}$	Processing time of successor of job j of project i
DD	f_{n_i}	End time of project i
RD	$s_{n_{i-1}+1}$	Start time of project i
LST	lst_{j_i}	Latest start time of job j of project i
RT	rt_{j_i}	Remaining time successors job j on critical path of project i

4.2 Selection, Crossover, Mutation and Forming a Population

To select individuals for crossover and copying, we use a tournament-like system (Blickle and Thiele 1996).

The implementation of the GA operator crossover is based on the standard uniform crossover (Kora and Yadlapalli 2017).

Each weight of a weight set can be mutated. The mutation of a weight is dependent on the rate of mutation, e.g. a mutation rate of 0.8 means that the probability for a mutation of a single weight is 80 %. If a gene for mutation has been selected, a number between 0 and 1 is set. After mutation, the total sum of a weight set will be adjusted to 1 (Fig. 2).

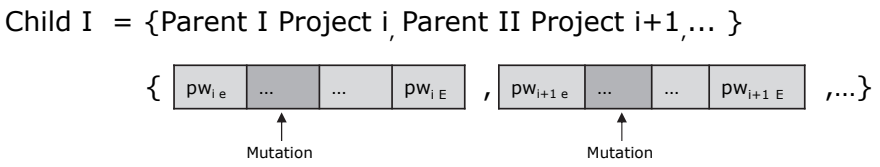

Figure 2: *Mutation*

A population is based on three parts: copied individuals, randomly created individuals and individuals generated by crossover and mutation. The size of each part can be defined by the user.

4.3 Calculation of Job Priorities

For each job in a queue, the individual job priority according to the project individual weight set and job attribute is being computed. After computing the priority, the jobs are sorted descending by their priority. The job with the highest priority will be performed next. The project individual calculation for the job priority in general is as follows (Eq. 4):

$$IP(j_i) = \sum_{e=1}^{E} pw_{ie} * \frac{PR_e(j_i)}{PR_{e\_max}} \quad (4)$$

where $IP(j_i)$ is the individual priority of job *j* of project *i*. $PR_e(j_i)$ is the job attribute of job *j* of project *i*, pw_{ie} is the project individual weight of $PR_e(j_i)$ and PR_{e_max} is the attribute`s maximum value of all jobs *j* in the queue.

4.4 Simple Example

Considered are *i = 2* projects with *j = 7* activities each, *G = 1* global resource with a capacity *ag = 4* resource units (RU). The due date for project 1 is *DD = 13* time units (TU) and for project 2 *DD = 15* TU. The objective is to minimize TMS (Eq. 1). The following network is given (Fig. 3):

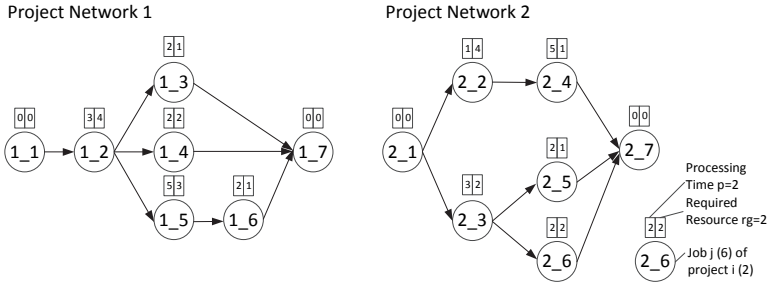

Figure 3: *Activity-on-node network for the example*

In the example, we use two job attributes: required global resource *rg* and processing time *p*. From Equation 4 results the following problem-specific equation (Eq. 5).

$$IP(j_i) = pw_{i\,rg} * \frac{rg(j_i)}{rg_{max}} + pw_{i\,p} * \frac{p(j_i)}{p_{max}} \quad (5)$$

The weight sets are calculated with the proposed GA. For the best value of TMS, following weight sets were applied (Tab. 2):

Table 2*: Weight sets for best TMS*

	$pw_{i\,rg}$	$pw_{i\,p}$
PIW_1 Project 1	0.9	0.1
PIW_2 Project 2	0.8	0.2

The scheduler operates with a parallel generation scheme. Priorities are calculated during the scheduling. The job with the highest priority is performed first. Using the proposed approach and the specified weight set (Tab. 2), the following schedule results (Tab. 3)

Table 3: *Simulation log (extract)*

T [TU]	ag [RU]	Executable Jobs and Calculated Priorities [Project_Job:Priority]	Executed Job
…	…	…	…
0	4	1_2: 1; 2_2: 0.86; 2_3: 0.6	1_2
3	4	1_3: 0.26; 1_4: 0.49; 1_5: 0.77; 2_2: 0.84 ; 2_3: 0.52	2_2
4	4	1_3: 0.34; 1_4: 0.64; 1_5: 1; 2_3: 0.65; 2_4: 0.46	1_5
4	1	1_3: 0.94; 2_4: 1	2_4
…	…	…	…

For the highlighted row (Tab. 3), the calculation of the priorities is demonstrated in the following equations (Eq. 6):

$$IP(1\_3) = 0.9 * \frac{1\,RU}{1\,RU} + 0.1 * \frac{2\,TU}{5\,TU} = 0.94$$
$$IP(2\_4) = 0.8 * \frac{1\,RU}{1\,RU} + 0.2 * \frac{5\,TU}{5\,TU} = 1 \quad (6)$$

The complete result for solving the considered problem with the approach and a comparison to SPR (EDD and SPT) is shown in the following Gantt-Charts (Fig. 4):

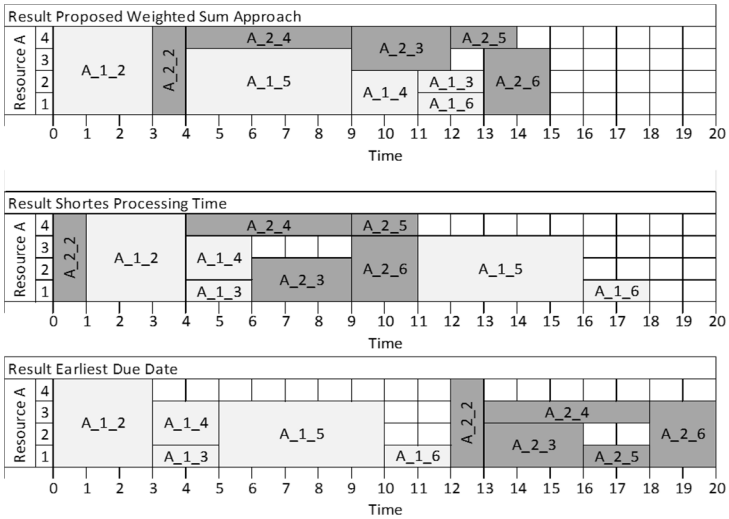

Figure 4: *Gantt-chart of the simple example*

5 Experiments and Results

5.1 PyScOp-Framework

For proofing the proposed concept, we developed a customized lean scientific scheduling and optimization framework named PyScOp (Python-based Scheduling and Optimization Framework, Kühn et al. 2017) which is based on the programming language Python (Python Software Foundation 2017). The basic function of the platform is as follows (Fig. 5):

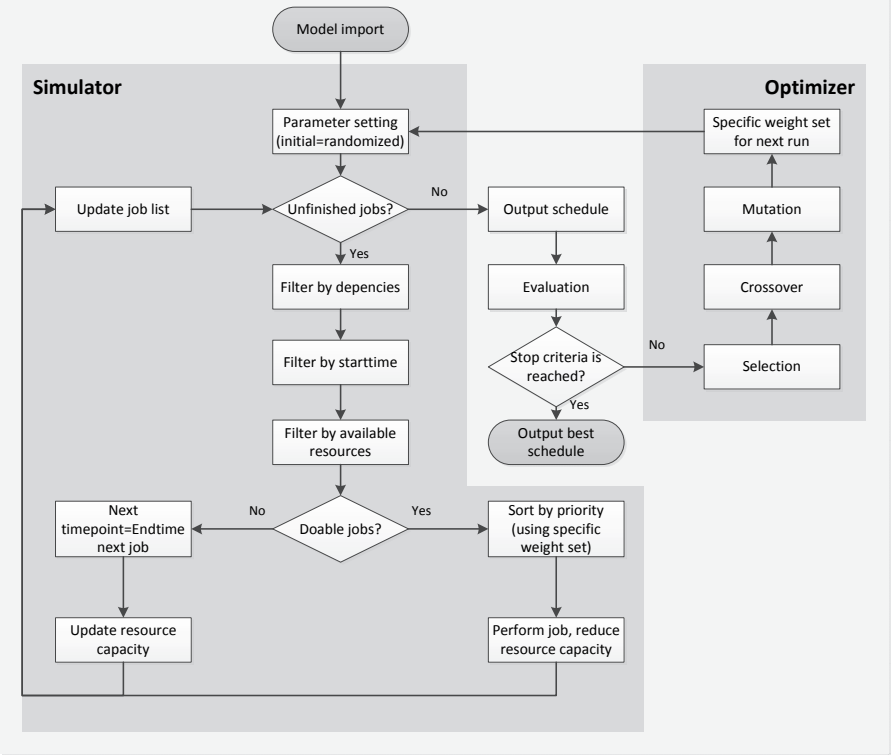

Figure 5: *Simulation-based optimization with proposed PyScOp-Framework*

5.2 Benchmark Tests

The performance of the proposed algorithm was benchmarked using the famous MPSPLIB (Homberger 2017) and compared to SPRs and the library's best results (Tab. 4). The objective of each optimization run was to minimize the total makespan (Eq. 1).

For the experiments we used the following GA settings:

Maximum population size: 100, maximum generation size: 500, copy rate 0.1, random rate 0.1, mutation rate 0.6, crossover rate 0.6 and tournament rate 0.05. If the objective value does not improves after 50 generations, the algorithm stops.

Table 4: *Problem character and results*

Problem	mp_j30_a10_nr1	mp_j30_a10_nr3	mp_j30_a10_nr5	mp_j90_a10_nr1	mp_j90_a10_nr3	mp_j90_a10_nr5	mp_j120_a10_nr2	mp_j120_a10_nr5	mp_j120_a20_nr3	mp_j120_a20_nr4	mpj_120_a10_nr5_AC1	mpj_120_a10_nr5_AC2	mp_j120_a20_nr5_AC1	mp_j120_a20_nr5_AC9
Results														
FIFO	196	266	222	228	249	270	307	540	338	297	806	432	413	342
EDD	196	284	209	228	240	282	316	540	275	232	839	440	433	346
SPT	204	263	218	187	246	320	307	551	290	220	823	417	406	328
CR	201	265	220	177	245	248	277	547	280	228	831	422	441	338
MSLK	191	256	192	166	226	270	287	490	280	212	773	384	384	302
MWRK	205	261	211	226	281	309	350	511	303	229	817	424	421	362
Best LIB*	**188**	**243**	**186**	**158**	**213**	**230**	**248**	**481**	**235**	**203**	**763**	**378**	**380**	**301**
Our WS**	**187**	**242**	**184**	**157**	**213**	**227**	**242**	**480**	**230**	**201**	**760**	**376**	**374**	**295**

*Best solution of MPSPLIB as of 05.05.2017 (Homberger 2017)

**Result of proposed weighted sum approach is submitted on MPSPLIB (Homberger 2017).

6 Conclusions and Further Research

In this paper, we presented an approach for calculating project individual job priorities for solving the MRCMPSP by using a customized genetic algorithm. For verifying the effectiveness, we developed a python-based scheduling framework and evaluated it with the famous benchmark library MPSPLIB. Our results are equal or better to existing solutions with respect to our chosen objective function TMS. However, there is much more research potential.

Our next step is the implementation of a multi-objective algorithm (e.g. a non-dominated sorting algorithm such as NSGA-III (Deb and Jain 2014)) to investigate the behaviour of differentiated objectives. Further, we will investigate the stochastic influence of process time fluctuation and want to answer the question, if it is possible to generate robust calculation models.

Acknowledgement

The research was funded by the Deutsche Forschungsgemeinschaft (DFG), “Simulationsbasierte dynamische Heuristik zur verteilten Optimierung komplexer Mehrziel-Multiprojekt-Multiressourcen-Produktionsprozesse” (SCHM 2689/5-1 RO 2126/3-1).

References

Blickle, T.; Thiele, L.: A Comparison of Selection Schemes Used in Evolutionary Algorithms. Evolutionary Computation 4 (1996) 4, pp. 361–394.

Bloech, J.: Einführung in die Produktion. Berlin, Heidelberg: Springer 2014.

Branke, J.; Nguyen, S.; Pickardt, C.W.; Zhang, M.: Automated Design of Production Scheduling Heuristics. IEEE Transactions on Evolutionary Computation 20 (2016) 1, pp. 110–124.

Brucker, P.; Knust, S.: Complex scheduling. New York: Springer 2012.

Deb, K.; Jain, H.: An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I. IEEE Transactions on Evolutionary Computation 18 (2014) 4, pp. 577–601.

Hildebrandt, T.; Freitag, M.: Bessere Prioritätsregeln für komplexe Produktionssysteme mittels multi-kriterieller simulationsbasierter Optimierung. In: Rabe, M.; Clausen, U. (eds.): 16. ASIM-Fachtagung Simulation in Production and Logistics 2015, Dortmund: Fraunhofer Verlag, pp. 309–318.

Homberger, J.: A multi-agent system for the decentralized resource-constrained multi-project scheduling problem. International Transactions in Operational Research 14 (2007) 6, pp. 565–589.

Homberger, J.: MPSPLIB. With cooperation of Christian Tausch und Christian Wiedemann. Hg. v. Hochschule für Technik Stuttgart. Online available www.mpsplib.com. Last access 07.07.2017.

Ingimundardottir, H.; Runarsson, T.P.: Evolutionary Learning of Linear Composite Dispatching Rules for Scheduling. In: Merelo, J.Julian; Rosa, A.; Cadenas, J.M.; Dourado, A.; Madani, K.; Filipe, J. (eds.): Computational Intelligence. Cham: Springer International Publishing 2016, pp. 49–62.

Kaban, A.K.; Othman, Z.; Rohmah, D.S.: Comparison of dispatching rules in job-shop scheduling problem using simulation. International Journal of Simulation Modelling 11 (2012) 3, pp. 129–140.

Kora, P.; Yadlapalli, P.: Crossover Operators in Genetic Algorithms. International Journal of Computer Applications 162 (2017) 10, pp. 34–36.

Kuczapski, A.M.; Micea, M.V.; Maniu, L.A.; Cretu, V.I., 2010: Efficient generation of near optimal initial populations to enhance genetic algorithms for job-shop scheduling. Information Technology and Control 39 (2010) 1, pp. 32–37.

Kühn, M.; Schmidt, T.; Genßler, P., 2017: PyScOp: TU Dresden, Professur für Technische Logistik. https://tlscm.mw.tu-dresden.de/scm/git/PyScOp. Last access 09.07.2017

Ma, L.M.; Li, J.Y.; Xu, W.S.; Kong, L.J.: Multi-Objective Scheduling Based on Weighted Combination of Heuristic Rules and the Simulation Method. Advanced Materials Research 424-425 (2012), pp. 1132–1138.

Panwalkar, S.S.; Iskander, W.: A Survey of Scheduling Rules. Operations Research 25 (1977) 1, pp. 45–61.

Pinedo, M.L.: Planning and scheduling in manufacturing and services. Dordrecht: Springer 2009.

Python Software Foundation, 2017: Python. www.python.org.Last access 10.02.2017.

Tay, J.C.; Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers & Industrial Engineering 54 (2008) 3, pp. 453–473.