Scheduling on Parallel Machines with Single and Multiple Servers[1]

Svetlana A. Kravchenko
Institute of Engineering Cybernetics, Surganov St. 6,
220012 Minsk, Belarus
email: kravch@newman.bas-net.by

Frank Werner
Otto-von-Guericke-Universität, Fakultät für Mathematik, PSF 4120,
39016 Magdeburg, Germany
email: frank.werner@mathematik.uni-magdeburg.de

Keywords: parallel machine problems, setup times, complexity of problems, worst case analysis of heuristics, polynomially solvable cases

1 Introduction

In this paper, we consider the problem of scheduling n jobs on a set of m identical parallel machines available for processing at time zero. The processing of a job must be performed on one of the machines without interruption. Before the processing of a job can start, a setup must be performed for this job by a server, which corresponds to a loading of this job on the corresponding machine. Such a setup is not possible during the processing of another job on the corresponding machine. If a server has performed a setup on a machine, the processing of this job can immediately start, but on the other hand, the server is free to perform another setup on another machine. It is assumed that travel times of a server between the machines are equal to zero. In the general case, a set of k servers is available to perform these required loadings.

So far, parallel machine scheduling problems with a single server have been considered. Classical scheduling objectives in connection with this type of problems have been considered by Hall, Potts & Sriskandarajah [10]. In that paper, a study of algorithmic and computational complexity is performed. In dependence on the objective function, the number of machines, and the structure of the processing and setup times, for the majority of these problems there is either given an efficient algorithm, or it has been proved that the corresponding problem is NP-hard which implies that the existence of such an efficient algorithm is rather unlikely. Some of the efficient algorithms have been obtained by adapting classical scheduling algorithms to the corresponding problem with a single server.

A beam search heuristic for parallel machine scheduling problems with a single server in such a static environment described above has been suggested by Koulamas [12]. Koulamas & Smith [13] give a look-ahead heuristic for the case of continuously arriving jobs.

There exist some problems considered in the literature which are related to parallel machine scheduling with a server. For instance, the machine interference problem involves the determination of an optimal number of machines to be assigned to a single operator such that the operator interference, which occurs when several machines request simultaneously the service of the operator, is minimized (see for instance Aronson [2]).

There also exist several studies of robots and automated guided vehicles in flexible manufacturing. Wilhelm & Sarin [19] investigate some complexity issues of several robot-served systems in flexible manufacturing. Theoretical results on scheduling a robotic cell have been given for instance by Hall, Kamoun & Sriskandarajah [7, 8] and Kamoun, Hall & Sriskandarajah [11]. A more detailed discussion of the literature on robots and automated guided vehicles can be found for instance in [5, 17].

Another related problem is, for example, the problem of concurrent resource scheduling, where the processing of some jobs require the simultaneous use of more than one resource. Dobson & Kamarkar [4] consider such a problem with the objective to minimize the total weighted completion time. Sahney [16] deals with minimizing the total job flow time in a problem with two identical parallel machines and a single server, where each job is preallocated to a machine and the server must attend the machine, but a fixed switching time incurs when the server moves between the machines.

[1] Supported by Deutsche Forschungsgemeinschaft (Project ScheMA) and by INTAS (Project 96-820)

In this paper, we generalize and extend some results from [10] by considering parallel machine problems with k servers, where k may be greater than one. The rest of the paper is organized as follows. The problem formulation and some required notations are given in Section 2. In Section 3 we consider problems with a single server. First we treat the case of minimizing the makespan, and then we handle the minimization of forced idle time, and the minimization of total flow time. Finally, in Section 4 we consider problems with more than one server. We concentrate on results obtained by the authors, but review also some other recent results.

2 Problem Formulation

Let $N = \{1, 2, \ldots, n\}$ be the set of jobs which has to be processed on m identical machines denoted by $M_1, M_2, \ldots, M_m$. For every job $j \in N$, a processing time p_j and a setup or server time s_j are given. The processing may be done on exactly one of the identical machines, and it may only start after a setup or loading of this job on the corresponding machine. Both, the setup and the processing, may not be interrupted. Throughout the paper, we assume that $s_j \geq 1$ and $p_j \geq 0$. Note that zero processing times are not excluded, and a job with a zero processing time can be interpreted as a job with such a small processing time, which can be disregarded. In dependence on the problem type considered, also a nonnegative due date d_j may be given for job $j \in N$.

To denote the problems considered in this paper, we use a classification scheme based on that given by Graham et al. [6], where a scheduling problem is described in the form $\alpha \mid \beta \mid \gamma$. In this scheme, α gives the scheduling environment, β describes some job characteristics, and γ indicates the objective function which has to be minimized. In addition to parallel machine problems without consideration of setups, parameter α includes a symbol of the form Sk, which indicates that k servers are available. For example, $\alpha_1 = Pm, S2$ indicates that we have a problem with m parallel machines, where m is fixed, and two servers. For parameter α_2, we also consider the case of unit processing ($p_j = 1$) and unit setup ($s_j = 1$) times. If the setup times are equal but not necessarily unit, we write $s_j = s$. If $\alpha_2 = \emptyset$, the processing and setup times are arbitrary.

In this paper, we consider the following objective functions in parameter α_3. If $\alpha_3 = C_{max}$, the makespan or schedule length has to be minimized, i.e. we minimize $C_{max} = \max_{j=1,2,\ldots,n}\{C_j\}$, where C_j denotes the completion time of job $j \in N$. Moreover, $L_{max} = \max_{j=1,2,\ldots,n}\{C_j - d_j\}$ denotes the minimization of the maximum lateness. In addition, we consider the minimization of total flow time (or the sum of the completion times $\sum C_j$) and the minimization of the forced idle time. Following [12], we define IT as the amount of time in list scheduling when some machine is idle due to the unavailability of the server for setting up a job when needed, i.e. we disregard the idle time on a machine after all of its processing is completed, but before the other machine completes its processing.

A popular class of heuristics for parallel machine problems are list scheduling algorithms. They try to balance the work load among parallel machines. Such a list schedule is described by a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of the jobs which gives the sequence in which the setups are performed by the server. More precisely, when considering the i-th job π_i, we determine machine M_j which becomes free first, assign job π_i to this machine and perform the setup by a server as early as possible.

3 Problems with a Single Server

In this section we give some complexity results, present some polynomially solvable cases and analyze several list scheduling heuristics for this problem.

3.1 Minimization of Makespan

In [10], it has been shown that problem $P, S1 \mid p_j = 1 \mid C_{max}$ can be solved in constant time. On the other hand, problem $P2, S1 \mid s_j = 1 \mid C_{max}$ was proven to be binary NP-hard, and problem $P2, S1 \mid s_j = s \mid C_{max}$ was proven to be unary NP-hard.

First we present a complexity result for problem $P, S1 \mid s_j = 1 \mid C_{max}$, where the number of machines is arbitrary.

Theorem 1 *Problem $P, S1 \mid s_j = 1 \mid C_{max}$ is unary NP-hard.*

Some polynomially solvable cases of problem $P2, S1 \mid\mid C_{max}$ has been given in [1]. For instance, it has been proved that problem $P2, S1 \mid s_j + p_j = t \mid C_{max}$ (i.e. each job has a constant sum of setup and processing time) can be solved in $O(n \log n)$ time.

Concerning a worst case analysis for list scheduling heuristics, in [9] the following results have been given:

a) *The LPT list scheduling heuristic for problem* $P, S1 \mid\mid C_{max}$ *has a worst-case performance ratio of* $2 - 1/m$.

b) *An arbitrary list scheduling heuristic for problem* $P, S1 \mid\mid C_{max}$ *has a worst-case performance ratio of* $3 - 2/m$.

Clearly, the same is true for problem $P, S1 \mid s_j = 1 \mid C_{max}$, and we show that the above bounds are tight even for the case of unit setup times. First, we consider the LPT heuristic and show that the bound $2 - 1/m$ is tight for problem $P, S1 \mid s_j = 1 \mid C_{max}$.

Consider the following instance of problem $P, S1 \mid s_j = 1 \mid C_{max}$ with $mk - 1$ jobs, where $k > m$. There are $m(m-1)$ jobs with lengths

$$p'_1 = p'_2 = \ldots = p'_{m(m-1)} = k$$

and $m(k-m) + m - 1$ jobs with unit lengths

$$p'_{m(m-1)+1} = \ldots = p'_{mk-1} = 1.$$

Then $\pi^L = (1, 2, \ldots, mk-1)$ is an LPT list schedule (i.e. all jobs are scheduled in the order of non-increasing lengths) with the objective function value

$$C^L_{max} = 2mk - k - m^2 + m - 1.$$

On the other hand, an optimal schedule with $C^*_{max} = mk - 1$ is obtained when all jobs are scheduled by the following list scheduling procedure *LISTA*.

Procedure *LISTA*

1. $i := 1$;
   While $i \leq m$ **Do**
   Begin
2. schedule $m - 1$ jobs with the length k;
3. schedule $k - m$ jobs with the length 1;
4. $i := i + 1$
   End;
5. schedule $m - 1$ jobs with the length 1.

Therefore, we have

$$C^L_{max}/C^*_{max} = (2mk - k - m^2 + m - 1)/(mk - 1)$$

which tends to $2 - 1/m$ for $k \to \infty$.

Now consider an arbitrary heuristic and show that the bound $3 - 2/m$ is tight for problem $P, S1 \mid s_j = 1 \mid C_{max}$. We consider an instance with $(k-m+1)m+(m-2)m+1$ jobs, namely $(k-m+1)m$ jobs numbered as $1, \ldots, (k-m+1)m$ with length 1, $(m-2)m$ jobs numbered as $(k-m+1)m+1, \ldots, (k-m+1)m+(m-2)m$ with length k, where $k > m$, and one job $(k-m+1)m+(m-2)m+1$ with length $mk - 1$.

An optimal schedule with $C^*_{max} = mk - 1$ is obtained when all jobs are scheduled by the following list scheduling procedure *LISTB*.

Procedure *LISTB*

1. $i := 1$;
2. schedule the job with length $mk - 1$;
   While $i \leq m$ **Do**
   Begin
3. schedule $m - 2$ jobs with the length k;
4. schedule $k - m + 1$ jobs with the length 1;
5. $i := i + 1$
   End.

If we schedule the jobs in an arbitrary way, we can schedule at first all unit time jobs, then all jobs with the length k and then the jobs with the length $mk - 1$. As a result, we obtain a schedule with the makespan value $\tilde{C}_{max} = 3km - 2k - 1 - m^2 + m$. So, we get

$$\tilde{C}_{max}/C^*_{max} = (3km - 2k - 1 - m^2 + m)/(mk - 1)$$

which tends to $3 - 2/m$ for $k \to \infty$.

The above tightness proofs of both bounds given in a) and b) show that from the point of a worst case analysis the problem with unit setup times is not easier than the problem with arbitrary setup times in the case of LPT and arbitrary list scheduling.

3.2 Minimization of Forced Idle Time

Next we deal with the IT criterion, where IT is the forced idle time. First of all we show that, unlike problem $P2, S1 \mid s_j = 1 \mid C_{max}$, problem $P2, S1 \mid s_j = 1 \mid IT$ can be solved in $O(n \log n)$ time. For this purpose we divide the set of all jobs into two subsets $A = \{i \mid s_j = 1, p_j = 0\}$ and $B = \{i \mid s_j = 1, p_j \neq 0\}$ and arrange all jobs from B in non-decreasing order of their p_j-values. Then we use the following algorithm $P2, S1 \mid s_j = 1 \mid IT$, where T_1 and T_2 describe the profile of the current partial schedule, i.e. T_j denotes the completion time of the job scheduled last on machine j in the current partial schedule.

Algorithm $P2, S1 \mid s_j = 1 \mid IT$

1. $T_1 := 0$; $T_2 := 1$;
   determine set A and the ordered set B;
   While $A \cup B \neq \emptyset$ **Do**
   Begin
2. determine machine u with $T_u = \min\{T_1, T_2\}$;
3. **If** ($\mid T_1 - T_2 \mid = 1$ and $B \neq \emptyset$) or $A = \emptyset$ **Then**
   schedule the next job from set B on machine u
   Else schedule an arbitrary job from set A
   on machine u;
4. actualize T_u and remove the scheduled job
   from the corresponding set
   End.

Theorem 2 *The above algorithm determines an optimal schedule for problem* $P2, S1 \mid s_j = 1 \mid IT$.

The next result shows that problem $P2, S1 \mid s_j = s \mid IT$ is unary NP-hard like the corresponding problem $P2, S1 \mid s_j = s \mid C_{max}$. This results strengthens a result by Koulamas [12], who proved unary NP-hardness only for the case of arbitrary setup times.

Theorem 3 *Problem* $P2, S1 \mid s_j = s \mid IT$ *is unary NP-hard.*

Now we consider some polynomially solvable cases of problem $P2, S1 \mid s_j = s \mid IT$. First we consider the case when all processing times are larger than the constant setup time ($p_j > s$).

Proposition 1 *Problem* $P2, S1 \mid s_j = s, p_j > s \mid IT$ *can be solved in* $O(n \log n)$ *time by the SPT (shortest processing time) rule.*

Now consider the case with constant setup times and $p_j < s$ for all i.

Proposition 2 *Problem* $P2, S1 \mid s_j = s, p_j < s \mid IT$ *can be solved in* $O(n)$ *time.*

3.3 Minimization of Total Flow Time

In [10], it has been shown that problems $P2, S1 \mid s_j = 1 \mid \sum C_j$ and problem $P, S1 \mid p_j = 1 \mid \sum C_j$ can be solved in $O(n \log n)$ time. Moreover, problem $P, S1 \mid p_j = 1, s_j = 1 \mid \sum C_j$ can be solved in constant time. On the other hand, problem $P2, S1 \mid s_j = s \mid \sum C_j$ is unary NP-hard.

The following results settles the open complexity status of the problem with an arbitrary number of machines and unit setup times [3].

Theorem 4 *Problem* $P, S1 \mid s_j = 1 \mid \sum C_j$ *is unary NP-hard.*

4 Problems with More Than One Server

In [10], it has been shown that problem $P2, S1 \mid s_j = 1 \mid C_{max}$ is binary NP-hard. It is not hard to prove the corresponding result for any fixed number of machines and servers.

Theorem 5 *Problem* $Pm, Sk \mid s_j = 1 \mid C_{max}$ *is binary NP-hard.*

Next, we will show that problem $Pm, S(m-1) \mid s_j = 1 \mid C_{max}$ can be solved optimally for any fixed number of machines in pseudopolynomial time. This result generalizes a result given in [14] for problem $P2, S1 \mid s_j = 1 \mid C_{max}$.

It is known that problem $Pm \mid\mid C_{max}$ can be solved optimally by a well-known pseudopolynomial algorithm (see for instance [18]) even for the case when all machines are available at different times. In the following, we denote this algorithm by PP. Further, we will apply algorithm PP to some partial schedule σ and some set $N' \subset N$ of jobs which means that, beginning from a partial schedule σ, we treat the problem as a parallel machine problem without servers and schedule all jobs from the set N' in such a way that it minimizes the makespan value.

A lower bound for the optimal objective function value of problem $Pm, S(m-1) \mid s_j = 1 \mid C_{max}$ problem is given by $C_{max} \geq \max\{n, \bar{C}_{max}\}$, where $\bar{C}_{max}$ is the optimal makespan value for the problem $Pm \mid\mid C_{max}$ when one machine is available at time 1, all other machines are available at time 0, and the problem includes n jobs with modified processing times $p'_j = p_j + 1$, where p_j is the processing time of job $j \in N$ in problem $Pm, S(m-1) \mid s_j = 1 \mid C_{max}$.

We denote a job $j \in N$ as feasible for some partial schedule σ if after adding job j by list scheduling to this partial schedule σ, in the resulting schedule not all m machines finish their work at the same time.

Before the first step of the algorithm all jobs are unscheduled and the partial schedule σ_0 can be described by the initial situation, where the first $m-1$ machines are available at time 0 and the last one is available at time 1. There are n steps of the algorithm. In each step i, some new job is scheduled and the obtained partial schedule σ_i is considered as a partial schedule for the parallel machine problem without servers but with modified processing times $p'_j = p_j + 1$ of all jobs.

Algorithm 1

Step i, $i = 1, \ldots, n$:

a) Choose the shortest feasible job (i.e. the job with the smallest modified processing time), such that after adding this job to the partial schedule σ_{i-1} by the list scheduling strategy, and applying algorithm PP to the resulting partial schedule σ_i and all unscheduled jobs with modified processing times $p'_j = p_j + 1$, the makespan of the obtained schedule does not exceed the value $C_{max} = \max\{n, \tilde{C}_{max}\}$.

In the case when such a choice is impossible, take any job.

b) Generate a new schedule σ_i from σ_{i-1} by applying list scheduling to the selected job, and delete this job from further consideration.

The time complexity of Algorithm 1 is $O(n^3(\sum_{i=1}^{n} p'_i)^{m-1})$. In order to know the $\tilde{C}_{max}$ value, we need to apply algorithm PP which takes $O(n(\sum_{i=1}^{n} p'_i)^{m-1})$ time. In part a) of step i, we have to consider at most n jobs and to apply algorithm PP to each of the schedules obtained, which takes $O(n(\sum_{i=1}^{n} p'_i)^{m-1})$ time. Therefore, part a) of step i needs $O(n^2(\sum_{i=1}^{n} p'_i)^{m-1})$ time. Since Algorithm 1 consists of n steps, the overall complexity is $O(n^3(\sum_{i=1}^{n} p'_i)^{m-1})$.

Theorem 6 *Algorithm 1 constructs an optimal schedule for problem* $Pm, S(m-1) \mid s_j = 1 \mid C_{max}$.

Algorithm 1 can be applied to problem $P, S \mid s_j = 1 \mid C_{max}$ with an arbitrary number of machines and servers. However, the schedule obtained is not necessarily optimal. Consider the following instance of problem $P3, S1 \mid s_j = 1 \mid C_{max}$. There are nine jobs with $p_1 = p_2 = p_3 = 1$, $p_4 = p_5 = p_6 = p_7 = 2$, $p_8 = 3$, $p_9 = 4$. An optimal schedule with $C_{max} = 10$ can be obtained by applying a list scheduling algorithm to the list $\pi = (4, 5, 6, 9, 1, 2, 8, 7, 3)$. Algorithm 1 schedules the jobs in the order $\pi' = (4, 5, 6, 7, 8, 9, 1, 2, 3)$. Notice that there is one time unit of idle time before the setup for job 3 begins on the corresponding machine. So the obtained C_{max} value is 11.

The existence of a pseudopolynomial algorithm for problem $Pm, Sk \mid s_j = 1 \mid C_{max}$ with $k \leq m-2$ remains open. However, the following result shows that for the L_{max} criterion, the problem with a fixed number of machines and servers cannot be solved in pseudopolynomial time unless P=NP.

Theorem 7 *Problem* $Pm, Sk \mid s_j = 1 \mid L_{max}$ *is unary NP-hard for any fixed number of machines* m *and servers* k *with* $k < m$.

We already know that problem $P, S1 \mid s_j = 1 \mid C_{max}$ is unary NP-hard. From this result we immediately obtain

Theorem 8 *Problem* $P, Sk \mid s_j = 1 \mid C_{max}$ *is unary NP-hard.*

If the number of machines is fixed, we get the following result.

Theorem 9 *Problem* $Pm, Sk \mid\mid C_{max}$ *is unary NP-hard for each* $k < m$.

Now turn to the problem with unit processing times. In [10], it has been shown that problem $P, S1 \mid p_j = 1 \mid C_{max}$ can be solved optimally in polynomial time. If the number of servers can be greater than one, then the problem becomes more complicated.

Theorem 10 *Problem* $P, Sk \mid p_j = 1 \mid C_{max}$ *is binary NP-hard for each fixed* $k > 1$.

Next, we perform a worst case analysis for two list scheduling algorithms for problem $P, Sk \mid\mid C_{max}$, i.e. the problem includes an arbitrary number of identical parallel machines and k servers, where for each job an arbitrary setup and processing times are given and the makespan has to be minimized.

We consider the case $k \geq 2$, and we first give the following worst-case performance bound for the LPT heuristic.

Theorem 11 *For an LPT schedule, the following estimation holds:*

$$C^{LPT}_{max}/C^{*}_{max} \leq 2 + (k-1)/k - k/m,$$

where C^{LPT}_{max} *denotes the makespan value of an LPT schedule and* C^{*}_{max} *denotes an optimal makespan value.*

Now we show that the above bound is tight. Consider the following instance of problem $P, Sk \mid\mid C_{max}$ with $(m-k)m + k(k-1) + 1$ jobs supposing that k is a divisor of m. There are given $m(m-k)$ jobs with $s_j = 0$ and $p_j = 1$, $k(k-1)$ jobs with $s_j = m/k$ and $p_j = 0$, and one job with $s_j = m$ and $p_j = 0$. By the LPT heuristic we can schedule all jobs with $p_j = 1$ first, then all jobs with $s_j = m/k$ and finally the job with $s_j = m$. So the total length of the resulting schedule is $m - k + (k-1)m/k + m$ whereas an optimal schedule has the length m, and the latter one is obtained by scheduling the job with $s_j = m$ first, and then k times we do the following: schedule $k-1$ jobs with $s_j = m/k$ and $(m-k)m/k$ jobs with $p_j = 1$.

We now analyze an arbitrary list of jobs.

Theorem 12 *For an arbitrary list of jobs, the List Scheduling heuristic has a worst-case performance ratio of*

$$C^{LS}_{max}/C^{*}_{max} \leq 3 - (k+1)/m,$$

where C^{LS}_{max} denotes the makespan value of an arbitrary list schedule and C^{*}_{max} denotes an optimal makespan value.

References

[1] ABDEKHODAEE, A.H., Scheduling Jobs with Setup Times on Parallel Machines with a Single Server, PhD Thesis, University of Melbourne, 1999.

[2] ARONSON, J.E., Two Heuristics for the Deterministic, Single Operator, Multiple Machine, Multiple Run Cyclic Scheduling problem, Journal of Operations Management 4, 1984, 159 – 173.

[3] BRUCKER, P., KNUST, S., KRAVCHENKO, S.A., WERNER, F., Complexity Results for Parallel Machine Problems with a Single Server, Working Paper, Universität Osnabrück, 1999.

[4] DOBSON, G., KAMARKAR, U.S., Simultaneous resource Scheduling to Minimize Weighted Flow Times, Operations Research 37, 592 – 600, 1989.

[5] GANESHARAJAH, T., HALL, N.G., SRISKANDARAJAH, C., Design and Operational Issues in AGV-Served Manufacturing Systems, Annals of Operations Research 76, 1997, 109 – 154).

[6] GRAHAM, R.L., LAWLER, E.L., LENSTRA, J.K., RINNOOY KAN, A.H.G., Optimization and Approximation in Deterministic Machine Scheduling: a Survey, Annals of Discrete Mathematics 5, 1979, 287 – 326.

[7] HALL, N.G., KAMOUN, H., SRISKANDARAJAH, C., Scheduling in Robotic Cells: Classification, Two and Three Machine Cells, Operations Research 45, 1997, 421 – 439.

[8] HALL, N.G., KAMOUN, H., SRISKANDARAJAH, C., Scheduling in Robotic Cells, Complexity and Steady State Analysis, European Journal of Operations Research 109, 1998, 43 – 65.

[9] HALL, N.G., POTTS, C.N., SRISKANDARAJAH, C., Parallel Machine Scheduling with a Common Server, In: *The Fifth International Workshop on Project Management and Scheduling*, Abstracts, 1997, pp. 102 – 106.

[10] HALL, N.G., POTTS, C.N., SRISKANDARAJAH, C., Parallel Machine Scheduling with a Common Server, Operations Research, 1998 (revised version submitted).

[11] KAMOUN, H., HALL, N.G., SRISKANDARAJAH, C., Scheduling in Robotic Cells: Heuristics and Cell Design, Operations Research, 1996 (to appear).

[12] KOULAMAS, C.P.: Scheduling on Two Parallel Machines for Minimizing Machine Interference, Working Paper, Department of Decision Sciences and Information Systems, Florida International University, 1993.

[13] KOULAMAS, C.P., SMITH, M.L., Look-Ahead Scheduling for Minimizing Machine Interference, International Journal of Production Research 26, 1988, 1523 – 1533.

[14] KRAVCHENKO, S.A., WERNER, F., Parallel Machine Scheduling Problems with a Single Server, Mathl. Comput. Modelling 26, No. 12, 1997, 1 – 11.

[15] KRAVCHENKO, S.A., WERNER, F., Scheduling on Parallel Machines with a Single Server, Otto-von-Guericke-Universität Magdeburg, FMA, Preprint 30/98, 1998.

[16] SAHNEY, V.K., Single-Server, Two-Machine Sequencing with Switching Time, Operations Research 20, 1972, 24 – 26.

[17] SETHI, S.P., SRISKANDARAJAH, C., SORGER, G., BLAZEWICZ, J., KUBIAK, W., Sequencing of Parts and Robot Moves in a Robotic Cell, International Journal of Flexible Manufacturing Systems 4, 1992, 331 – 358.

[18] TANAEV, V.S., GORDON, V.S., SHAFRANSKY, Y.M, *Scheduling Theory, Single-Stage Systems*, Kluwer Academic, 1994.

[19] WILHELM, W.E., SARIN, S.C., A Structure for Sequencing Robot Activities in Machine Loading Applications, International Journal of Production Research 23, 1985, 47 – 64.