Kolisch, Rainer

Working Paper

Just-in-time production of large-scale assemblies: An integrative perspective

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 437

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Kolisch, Rainer (1997) : Just-in-time production of large-scale assemblies: An integrative perspective, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 437, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
http://hdl.handle.net/10419/149057

Terms of use:
Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.
'Just-in-time' production of large-scale assemblies: An integrative perspective

R. Kolisch

March 1997
'Just-in-time' production of large-scale assemblies: An integrative perspective

R. Kolisch

March 1997

This work has been supported by the Deutsche Forschungsgemeinschaft DFG
Abstract

Since its advent, the 'just-in-time'-paradigm has considerable influence on production planning and control. While originally employed for high-volume production, 'just-in-time'-concepts are meanwhile documented for low-volume, make-to-order production as well. An example is the assembly of multiple customer-specific products as recently considered by Agrawal et al. The problem has been inspired by the production planning of Westinghouse ESG, Maryland. Agrawal et al. modelled the problem as MILP and proposed a suited heuristic with acronym LETSA which gives near-optimal solutions. We show that the problem can be modelled as simple single resource-constrained project scheduling problem (RCPSP). This has several advantages: First, it allows an integrated perspective on 'just-in-time'-steered low-volume, make-to-order assemblies. Second, it makes available the rich supply of models, insights, and methods of resource-constrained project scheduling. The latter includes the well-known serial scheduling algorithm which brings forth the same schedule as LETSA with a computational effort which is only \( O(n^2) \) instead of \( O(n^4) \), where \( n \) is the number of assembly operations.

1 Introduction

Since the advent of 'just-in-time'-driven production planning and control at the Toyota manufacturing plants, the 'just-in-time'-paradigm has considered wide-spread consideration within production and operations management (cf., e.g., Schniederjans [14] and Cheng and Podolski [4]). While it was first employed for the high-volume-production of goods only, recently there has been considerable research in the area of low-volume, make-to-order manufacturing (cf., e.g., Baker and Scudder [3] and Rachamadugu [13]). Agrawal et al. [2] give an application of 'just-in-time'-production planning for large-scale, make-to-order assemblies. They considered a practical scheduling problem at Westinghouse ESG, where a number of customer-specific products have to be assembled subject to technologically precedence and capacity constraints. The authors developed a suited MILP-formulation and – in the face of the \( \mathcal{NP} \)-hardness of the problem – a "lead time evaluation an scheduling algorithm" with acronym LETSA. The algorithm has a computational complexity of \( O(n^4) \) where \( n \) is the number of assembly operations.
operations. Computational experiments performed by the authors indicate that the heuristic derives near optimal results.

In what follows we will show that the type of problems as considered by Agrawal et al. can be modelled as classical resource-constrained project scheduling problem (RCPSP) and that each assembly problem can be polynomially transformed into an RCPSP. This is important for several reasons.

First, it allows an integrated view on 'just-in-time'-steered low-volume, make-to-order assembly. The RCPSP is well established, which facilitates the classification of the assembly-scheduling problem w.r.t. to existing scheduling problems.

Second, a lot of research has been and is performed for the RCPSP. A recent overview is given, e.g., in Özdamar and Ulusoy [11]. Hence, there are both, optimal and heuristic, algorithms with theoretical or experimental performance guarantees available. Experimental investigations on test-sets with well-defined problem parameters, as given, e.g., in Kolisch et al. [9], allow a very precise choice of algorithms suited for different type of RCPSP-problems. This knowledge can be employed in modern decision support systems for production planning such as documented in Drexel and Kolisch [5] and Nowicki and Smutniki [10].

Finally, we will show that solving the RCPSP with one of the heuristics, the well known serial scheduling algorithm, generates for any transformed assembly instance the same schedule as LETSA, but with a computational effort which is only $O(n^2)$.

The remainder of the paper is organised as follows: In Section 2 we introduce the assembly scheduling problem and the lead time evaluation and scheduling algorithm proposed in Agrawal et al. Section 3 provides the resource-constrained project scheduling problem and outlines the serial scheduling algorithm. Then, in Section 4, we show that the assembly scheduling problem can be polynomially transformed into an RCPSP and that both scheduling algorithms will generate equivalent schedules. Conclusions are given in Section 5.
The assembly scheduling problem

2 The assembly scheduling problem

The assembly scheduling problem (ASP) can be depicted as follows. There are \( e = 1, \ldots, n_j \) customer-specific products. Each product \( e \) has to be assembled by its due date \( D_e \) at the latest. The assembly-structure of each product \( e \) is depicted by its bill of material (BOM). Figure 1 shows the BOM of two products. Product \( e = 1 \) with due date \( D_1 = 14 \) comprises operations \( O_1, \ldots, O_6 \), product \( e = 2 \) with due date \( D_2 = 10 \) comprises operation \( O_7 \). Each rectangle depicts a make part which is manufactured by a sequence of operations. Overall, there are \( i, \ldots, I \) operations. Each product \( e \) has one final-assembly operation \( f(e) \). All other assembly operations \( O_i \) of product \( e \) have exactly one downstream operation \( d(i) \). This gives for each product an assembly structure of the operations. In the assembly shop there are \( m \) different work-centres. In each work-centre \( W_k \), \( K = 1, \ldots, m \), there are \( f_k \) functional identical machines. \( I_k \) is the set of operations which have to be processed by one of the machines in work-centre \( W_k \). The processing of operation \( O_i \) takes \( t_i \) periods time. Once started, an operation cannot be preempted. When processed, operation \( O_i \) occupies one of the functional identical machines of the work-centre where it has to be processed. Table 1 and Figure 1 give a two product example which has been derived by adding product \( e = 2 \) to the example originally proposed in Agrawal et al.

In order to model the ASP, Agrawal et al. introduce the following decision variables:

\[
\delta_{i,j} = \begin{cases} 
1, & \text{if operation } O_j \text{ precedes operation } O_i \\
0, & \text{otherwise}
\end{cases}
\]

![Table 1: Data of the assembly scheduling problem](image)

2 We use, with some minor modifications, the original notation proposed by Agrawal et al.
\[ \Delta_{i,k} = \begin{cases} 
1, & \text{if operation } O_i \text{ is performed on the } k\text{-th functionally identical machine of 'its' work-centre} \\
0, & \text{otherwise} 
\end{cases} \]

\[ S_i \geq 0 \quad \text{start time of operation } O_i. \]
\[ C_i \geq 0 \quad \text{completion time of operation } O_i. \]

A MILP for the ASP develops then as follows:

\[ \text{Max } Z = \min \{ S_i \mid i = 1, \ldots, n \} \quad (1) \]
\[ \text{s.t.} \]
\[ S_{d(i)} \geq C_i \quad i = 1, \ldots, n \quad (2) \]
\[ C_i = S_i + t_i \quad i = 1, \ldots, n \quad (3) \]
\[ C_{f(e)} \leq D_e \quad e = 1, \ldots, n_f \quad (4) \]
\[ \delta_{i,j} + \delta_{j,i} = 1 \quad i, j \in I_K, i \neq j, K = 1, \ldots, m \quad (5) \]
\[ S_i - C_j \geq M (\delta_{i,j} + \Delta_{i,k} + \Delta_{j,k} - 3) \]
\[ i, j \in I_K, i \neq j, K = 1, \ldots, m, k = 1, \ldots, f_K \quad (6) \]
(1) maximises the start time of the first-starting operation. Together with constraints (4), which secure that each product is delivered by its due date, this enforces schedules which start operations as late as possible with the aim to obtain solutions with low tied-up capital. This is the classical pull-strategy of 'just-in-time' manufacturing planning. The constraints (2) enforce the technological precedence relations. (3) simply links the start and finish time for each operation. Constraints as enforced by scarce capacities of the machines are given by (5) – (7). (5) selects for each pair of operations \((O_i, O_j)\) which have to be processed at the same work-centre the orientation \(O_i \rightarrow O_j\) or \(O_j \rightarrow O_i\). (7) assigns each operation to be processed at a work-centre one of its machines. Finally, (6) enforces the orientation between the two operations \(O_i\) and \(O_j\) if they are processed on the same machine.

Agrawal et al. [1] prove that the ASP is an \(\text{NP}\)-hard optimisation problem. In order to solve large real-world instances, they propose a "lead time evaluation and scheduling algorithm" (LETSA). Let denote with \(F\) the set of precedence feasible operations, i.e. operations that can be scheduled in the current iteration since all their upstream operations have already been scheduled. Then, LETSA can be outlined as follows.

**Initialisation** Set \(F = \{O_{f(e)} \mid e = 1, \ldots, n_f\}\)

While \(F \neq \emptyset\) do

1. Calculate for each path from an operation in \(F\) to an operation which is not downstream of any other operation the path-length, i.e. the sum of the duration of all operations on this path.

2. Determine the path with the longest duration (critical path) and its associated operation \(O_c\) in \(F\).

3. Set the tentative completion time \(C_c\) of operation \(O_c\) equal the start time of the downstream operation, or, in case \(O_c\) is a final-assembly operation, equal the due date of the associated product.

\[
\sum_{s=1}^{f_K} \Delta_{i,s} = 1 \quad K = 1, \ldots, m, \ i \in I_K
\]  
\[
S_i \geq 0 \quad i = 1, \ldots, n
\]  
\[
C_i \geq 0 \quad i = 1, \ldots, n
\]  
\[
\delta_{i,j} \in \{0, 1\} \quad i, j \in I_K, i \neq j, K = 1, \ldots, m
\]  
\[
\Delta_{i,k} \in \{0, 1\} \quad K = 1, \ldots, m, i \in I_K, k = 1, \ldots, f_K
\]
4. Calculate for each machine \( k = 1, \ldots, f_c \) of the work-centre where operation \( O_c \) has to be processed, the latest resource-feasible start-time \( S_c(k) \) such that \( S_c(k) \leq C_c - t_c \) holds.

5. Select the start time \( S_c = \max \{ S_c(k) \mid k = 1, \ldots, f_c \} \), set \( \Delta_c, k = 1 \) for the associate machine, set the final completion time \( C_c = S_c + t_c \).

Update \( F \) by removing operation \( O_c \) and adding the operations which are upstream-operations of \( O_c \).

The computational effort of LETSA is as follows. Obviously, steps (2), (3), and (5) are linear. For each machine \( k \), step (4) can be performed with linear effort of \( n \), which will be shown in Section 3. The computationally most demanding step is (1). Here, we have at most \( \frac{1}{4}n^2 \) paths.\(^3\) Calculating the length of one path can be done in \( n \) which gives an effort of \( O(n^3) \). Performing the while-loop \( n \) times results in an overall computational effort of \( O(n^4) \).

Solving the example problem with LETSA, we obtain the schedule given in Figure 2 with an objective function value of 14.

![Figure 2: Schedule generated with LETSA](image)

3 The resource-constrained project scheduling problem

The resource-constrained project scheduling problem (RCPSP) can be depicted as follows.\(^4\) A single project consists of \( j = 1, \ldots, J \) jobs

\(^3\)Let \( x = | F | \), then the maximum number of paths, \( N \), is \( N(x) = x(n - x) \). The extremum of the function \( N \) is obtained by setting \( N'(x) = 2x - n = 0 \) which gives \( x = \frac{1}{2}n \). Since \( N''(x = \frac{1}{2}n) > 0 \), \( N(x = \frac{1}{2}n) = \frac{1}{4}n^2 \) is the maximal number of paths.

\(^4\)As it is the case in project scheduling, we depict jobs and resources by their index \( j \) and \( r \), respectively.
which are interrelated by precedence and resource constraints. Acyclic precedence constraints impose precedence relations with minimal time lags between pairs of jobs such that job \( j \) has a set of immediate predecessors \( P_j \). The minimal time lag between the completion time of each predecessor \( h \in P_j \) and the start of job \( j \) is \( l_{h,j} \). There are \( r = 1, \ldots, R \) different type of resources where resource type \( r \) has a period capacity of \( K_r \) units per period. Job \( j \) uses \( k_{j,r} \) units of resource type \( r \) when being processed. The duration of job \( j \) is \( d_j \). W.l.o.g. we assume that job \( j = 1 \) is the unique dummy-source and that job \( j = J \) is the unique dummy-sink of the network, i.e. we have \( d_1 = d_J = 0 \) and \( k_{1,r} = k_{J,r} = 0, r = 1, \ldots, R \). The RCPSP-model stems from the formulation of Pritsker et al. [12] which employs the following binary variables:

\[
x_{j,t} = \begin{cases} 
1, & \text{if job } j \text{ is finished at the end of period } t \\
0, & \text{otherwise}
\end{cases}
\]

The number of variables can be drastically reduced if one considers for each job \( j \) only the time window \([EF_j, LF_j]\) of precedence-feasible earliest and latest finish times. Earliest finish times are calculated by setting \( EF_1 = 0 \) and performing a forward recursion for \( j = 2, \ldots, J \); latest finish times are derived by setting \( LF_j = T = \sum_{j=1}^{J} d_j \) and performing a backward recursion for \( j = J - 1, \ldots, 1 \). Both recursions can be done in linear time effort \( J \) (cf. Elmaghraby [6]). Given the time window for all jobs, we can model the RCPSP as follows (cf., e.g., Sprecher et al. [15]):

\[
\text{Min } Z = \sum_{t=EF_j}^{LF_j} t \cdot x_{j,t} \\
\text{s.t.} \\
\sum_{t=EF_j}^{LF_j} x_{j,t} = 1 \quad j = 1, \ldots, J \\
\sum_{t=EF_h}^{LF_h} x_{h,t} \left( t + l_{h,j} \right) \leq \sum_{t=EF_j}^{LF_j} \left( t - d_j \right) x_{j,t} \quad j = 2, \ldots, J, \quad h \in P_j
\]
\[
\sum_{j=1}^{J} \min\{T,j+\delta_j-1\} \sum_{\tau=t}^{k_j} x_{j,\tau} \leq K_r \quad r = 1, \ldots, R; t = 1, \ldots, T \quad (15)
\]

\[
x_{j,t} = \begin{cases} 0,1 \end{cases} \quad (16)
\]

(12) minimises the finish time of the sink and thus the makespan of the entire project. (13) enforces each job to be processed within its time-window. (14) ensures the precedence constraints with minimal time lags between jobs while (15) makes sure that for each time period, the resource demand of the processed jobs does not exceed the available capacity. Finally, (16) defines the binary decision variables.

One of the first heuristics proposed for solving the RCPSP is the serial scheduling algorithm which has been originally given by Kelley [7]. An actual overview about this method which includes recent research results can be found in Kolisch [8]. The serial scheduling algorithm uses two job-sets. The scheduled set \( S \) consists of all jobs which have been scheduled and the decision set \( D \) is made up of all jobs which are schedulable in a iteration, since all their predecessors are in \( S \). An algorithmic description of the serial scheduling algorithm can be given as follows:

**Initialisation** Set \( S = \emptyset \) and \( D = \{1\} \), calculate for each job \( j \) the latest start time \( LS_j = LF_j - \delta_j \) by backward recursion from \( T \).

While \( D \neq \emptyset \) do

1. Select the job \( j^{*} \in D \) with the smallest \( LS_j \)-value
2. Set the start \( S_{j^{*}} \) of job \( j^{*} \) to the earliest period within the interval \([\max\{S_h + \delta_h + t_h,j^{*} \mid h \in P_{j^{*}}\}, LF_{j^{*}} - \delta_{j^{*}}\] which is resource-feasible for the periods \([S_{j^{*}}, S_{j^{*}} + \delta_{j^{*}} - 1]\).
3. Update resource profiles and job-sets, i.e. \( S = S \cup j^{*}, D = D \setminus j^{*} \cup \{l \mid j^{*} \in P_l, P_l \in S\}\).

The computational complexity of the serial scheduling algorithm is as follows. Performing the backward recursion of the initialisation needs effort \( J \). Within the while-loop, step (2) needs the most computational effort. For one operation, at most each period of the horizon has to be checked for resource-feasibility w.r.t. each of the \( R \) resource-types. Since we have \( T = \sum_{j=1}^{J} \delta_j \) and for a transformed ASP, at most one resource-type is used by each job, we have a linear effort within \( J \).
Performing the while-loop \( J \) times, the overall complexity of the serial method is \( \mathcal{O}(J^2) \). Note, that the serial scheduling algorithm can be applied with any priority rule and is not, as outlined above, restricted to the latest-start-time-rule.

The serial scheduling algorithm has the following nice properties (cf. Kolisch [8]). It derives feasible solutions for any RCPSP. Furthermore, the generated schedules belong to the set of active schedules. Active schedules are characterised by the fact that no activity can be started earlier without starting some other activity later. It has been shown in Sprecher et al. [15] that, when a regular measure of performance such as the makespan is considered, the set of active schedules does always contain an optimal schedule.

Kolisch [8] tested different combinations of scheduling algorithms and priority rules on 'typical' RCPSP-instances with 30 jobs and 4 resources. The results show that the combination of serial scheduling algorithm and latest start time rule is, with an average percent deviation from the optimum of approximately 5 \%, one of the best priority-rule based heuristics.

4 Equivalence of the assembly and the project scheduling problem

Any assembly scheduling problem (1) – (11) can be transformed into an RCPSP (12) – (16). The basic idea is to 'reverse' the assembly-structure of each final product and to combine all structures into a single super-project where due dates of final assemblies are modelled by minimal time lags between the project source and the job representing the final assembly operations of each final product, respectively. More formally, a polynomial transformation of the ASP to the RCPSP can be given as follows:

1. **Initialisation**
   Set the number of jobs: \( J = n + 2 \)
   Set the number of resources: \( R = m \)
   Set the capacity of resource \( r \): \( K_r = f_{K_{err}}, r = 1, \ldots, R \)
   Set the dummy-source: \( d_1 = 0, k_1, r = 0, \ r = 1, \ldots, R, \ \mathcal{P}_1 = \emptyset \)
   Set the dummy sink: \( d_2 = 0, k_j, r = 0, \ r = 1, \ldots, R, \)
\[ \mathcal{P}_f = \{ h \mid h \neq d(i), i = 1, \ldots, n \}, \lambda_{h,f} = 0, \quad h \in \mathcal{P}_f. \]

2. **Transformation of the operations**
For \( i = 1, \ldots, n \) do:
\[ j(i) = J - i, \quad \mathcal{P}_{j(i)} = d(i), \quad l_{h,j(i)} = 0, \quad h \in \mathcal{P}_{j(i)} \]

\[
k_{j(i),e} = \begin{cases} 
1, & \text{if } i \in I_{(k=\epsilon)} \\
0, & \text{otherwise} 
\end{cases}
\]

3. **Set due date-based minimal time lags from the dummy-source to the final assembly jobs**
For \( e = 1, \ldots, n_f \) do:
\[ \mathcal{P}_{j(f(e))} = \{ 1 \}, \quad l_{1,j(f(e))} = \max \{ D_e \mid e = 1, \ldots, n_f \} - D_e. \]

If we apply the transformation steps (1) - (3) to the problem as depicted in Figure 1 and Table 1 we obtain the RCPSP as given in Figure 3 and Table 2. The solution obtained by the serial scheduling algorithm is the schedule given in Figure 4 with an objective function value of 14.

![Figure 3: Precedence network of the example problem](image)

**Theorem.** The serial scheduling algorithm generates a schedule for the RCPSP which is equivalent to the schedule generated by LETSA for the ASP.

**Proof.** The serial scheduling algorithm starts selected jobs as early as possible, while LETSA starts selected operations as late as possible. Hence, it suffices to show that the sequence of jobs selected by the serial
scheduling algorithm is the same as the associated operation–sequence selected by LETSA. This will be the case if the following holds: For the second iteration of the serial scheduling algorithm the set of jobs $\mathcal{D}$ equals the set of associate operations $F$ within the first iteration of LETSA \textit{(Equivalence of selection sets)}. For each iteration $2, \ldots, J-1$ of the serial scheduling algorithm the selected job $j^*$ equals the associate operation $O_c$ selected by LETSA in iteration $1, \ldots, n$ \textit{(Equivalence of selected entity)}.

\textit{Equivalence of selection sets.} In the second iteration of the serial scheduling algorithm the dummy-source has already been scheduled and we have $\mathcal{D} = \{ j \mid P_j = 1 \}$. The jobs in $\mathcal{D}$ represent exactly the final-assembly operations being in the set $F$ at the first iteration of LETSA. This can bee seen by looking at the transformation step (3).

\textit{Equivalence of selected entity.} W.l.o.g. we assume that the data is such that we do not need a tie-breaker for the two heuristics. First, we consider LETSA. Obviously, we can add the artificial operation $O_0$ to LETSA. $O_0$ has all the operations as downstream operations, which originally do not have an upstream operation. This way we have to calculate for each operation $O_i \in F$ only the longest path $P_{i,0}$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j(i)$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>$d_j(i)$</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$k_j(i),1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$k_j(i),2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: RCPSP-data

Figure 4: Schedule generated by the serial scheduling algorithm
The operation which will be selected is then $O_c = \max \{ P_{i,0} \mid i \in F \}$. The serial scheduling algorithm selects the job as follows: $j^* = \min \{ LS_j \mid j \in D \}$. Denoting with $P_{j,J}$ the longest path from job $j$ to the sink $J$ we can write instead $j^* = \min \{ T - P_{j,J} \mid j \in D \}$. Since $T$ is constant for all $j \in D$, this is $j^* = \min \{ -P_{j,J} \mid j \in D \}$ which equals $j^* = \max \{ P_{j,J} \mid j \in D \}$. □

5 Conclusions

We have shown that the problem of scheduling assemblies in a 'just-in-time' fashion as treated by Agrawal et al. can be modelled as classical RCPSP. This allows an integrated perspective with the following implications. Solving the RCPSP with the well-known serial scheduling algorithm generates the same solutions as the heuristic proposed by Agrawal et al., but with a smaller computational effort. Furthermore, many other exact and heuristic procedures have been developed for the RCPSP. They are now available as a backbone for powerful decision support systems for solving low-volume, make-to-order assemblies.

References


\footnote{Note that this reduces the effort for step (1) of LETSA from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.}


