Kolisch, Rainer; Heß, Karsten

Efficient methods for scheduling make-to-order assemblies under resource, assembly area, and part availability constraints

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 474

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 474

Efficient Methods for Scheduling Make-to-Order Assemblies under Resource, Assembly Area, and Part Availability Constraints

R. Kolisch and K. Heß

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 474

Efficient Methods for Scheduling Make-to-Order Assemblies under Resource, Assembly Area, and Part Availability Constraints

R. Kolisch and K. Heß

June 1998

Dr. Rainer Kolisch, Karsten Heß
Lehrstuhl für Produktion und Logistik,
Institut für Betriebswirtschaftslehre,
Christian-Albrechts-Universität zu Kiel,
Olshausenstr. 40, 24098 Kiel, Germany
email: kolisch@bwl.uni-kiel.de
URL: http://www.wiso.uni-kiel.de/bwlinstitute/Prod/kolisch.html
ftp://ftp.wiso.uni-kiel.de/pub/operations-research

Abstract

We consider the problem of scheduling multiple, large–scale, make–to–order assemblies under resource, assembly area, and part availability constraints. Such problems typically occur in the assembly of high volume, discrete make–to–order products. Based on a list scheduling procedure which has been proposed in Kolisch [19] we introduce three efficient heuristic solution methods. Namely, a biased random sampling method and two tabu search–based large–step optimization methods. The two latter methods differ in the employed neighborhood. The first one uses a simple API–neighborhood while the second one uses a more elaborated so–called 'critical neighborhood' which makes use of problem insight. All three procedures are assessed on a systematically generated set of test instances. The results indicate that especially the large–step optimization method with the critical neighborhood gives very good results which are significant better than simple single–pass list scheduling procedures.

1 Introduction

In this paper we consider the problem of scheduling large–scale assemblies in a make–to–order environment. A practical application is a German company which manufactures customized palletising systems for the chemical and food industry. The company has 70 employees, 20 of them working in the final assembly. In 1997, 100 palletising systems with an average revenue of 200,000 German Mark have been delivered to customers. The time between the confirmation of an order and the delivery to the customer is on average 20 weeks and is made of order–specific construction, fabrication, and assembly. The major part of the fabrication is done by outside suppliers. Currently this causes long lead times of up to 12 weeks. The assembly of parts takes on average 3 weeks. A problem formulation and a simple single–pass heuristic for the problem have been proposed in Kolisch [19]. Here, we focus on advanced methods for solving this problem.

The paper is organized as follows. In Section 2 we outline the problem and provide a mixed integer decision problem. Sections 3 and 4 introduce efficient methods for solving the assembly scheduling problem. First, we will show in Section 3 how feasible schedules can be constructed with a list scheduling heuristic. Section 4 is devoted to advanced solution methods, namely biased random sampling and tabu search–based large–step optimization. An extensive computational study is performed in Section 5.

2 Problem Description and Formulation

2.1 Problem Description

The problem of scheduling large–scale make–to–order assemblies can be depicted as follows. A number of $A \geq 0$ customer specific end products have to be assembled and delivered at individual due dates. Product $a \in \{1, \ldots, A\}$ has a due date of $d_a \geq 0$ and a penalty weight of $w_a \geq 0$ for every period the product is delivered after the due date. The assembly of product a consists of a number of assembly operations which are interrelated by technological precedence constraints. Without loss of generality, we assume for each order a a unique start operation s_a and a unique end operation e_a. $\mathcal{S} = \{s_a \mid a = 1, \ldots, A\}$

and $\mathcal{E} = \{e_a \mid a = 1, \ldots, A\}$ denote the set of start and end operations of all orders, respectively. A graphical representation of all operations and their technological interrelations can be made with an integrated assembly graph. Here, the orders are integrated into one network by a dummy start operation which precedes all order start operations and e dummy end operation which succeeds all order end operations. Figure 1 gives an integrated assembly network with $A = 3$ orders.

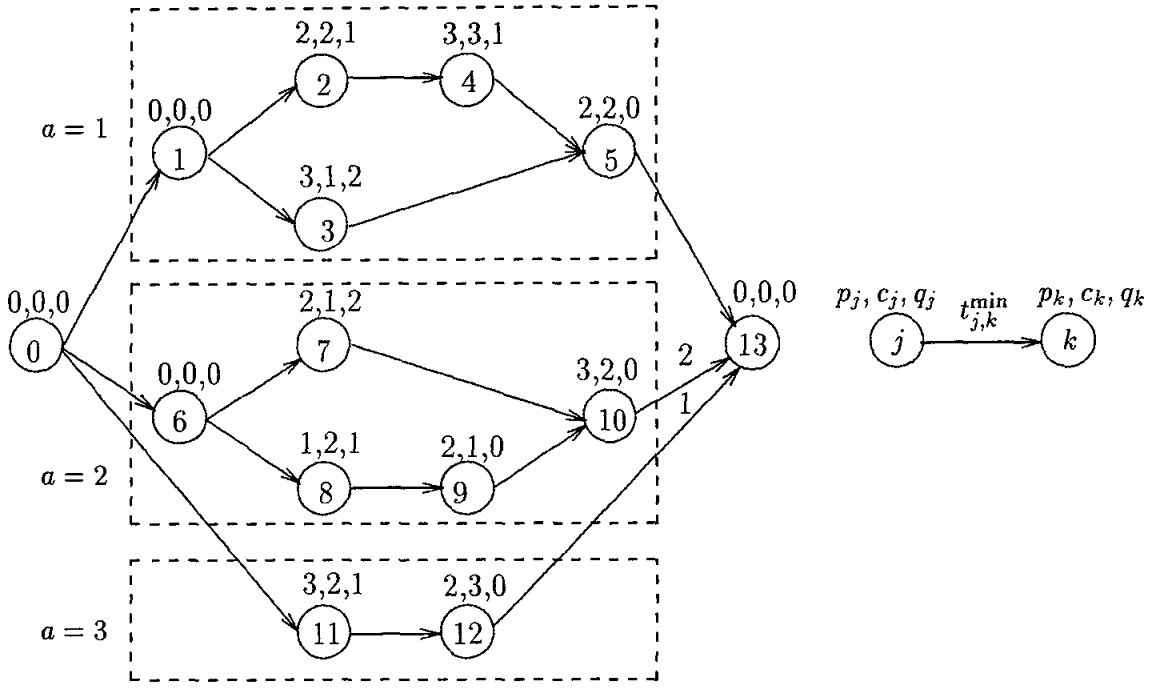

Figure 1: Integrated Assembly Graph

With $j = 0, \ldots, J+1$ we denote the operations of the assembly network. The operations 0 and $J + 1$ are the dummy start and end operation of the integrated assembly network, respectively. All operations of one order are consecutively labelled, i.e. order a consists of the operations $s_a, \ldots, e_a$. Additionally, the labelling considers the topological order of operations, i.e. $h < j$ holds for each predecessor operation h of operation j. With $\mathcal{J} = \{0, \ldots, J + 1\}$ we denote the set of all assembly operations; $\mathcal{P}_j$ and $\tilde{\mathcal{P}}_j$ denote the set of immediate and the set of all predecessors of operation j, respectively. Between each operation j and each of its immediate predecessors $h \in \mathcal{P}_j$ we have an arc $h \to j$ with weight $t^{\min}_{h,j} = 0$; between all order end operations e_a $(a = 1, \ldots, A)$ and the dummy sink $J + 1$ we introduce arcs with weight $t^{\min}_{e_a,J+1} = d^{\max} - d_a$ where $d^{\max} = \max\{d_a \mid a = 1, \ldots, A\}$ denotes the maximal due date of all orders.

The entire assembly of one order takes place on an assembly area, which is typically the shop floor. Since the shop floor area is limited, no more than $C \geq 0$ orders can be assembled at the same time. The assembly of an order $a \in \{1, \ldots, A\}$ begins as soon as the start operation s_a of the order is started and it commences until the end operation e_a is finished. During the entire time interval the order occupies the shop floor. Order

preemption, i.e. the removal of a partly assembled product from the assembly area, is not allowed because of high set up costs and the risk of damage.

Each non–dummy assembly operation $j \in \{1, \ldots, J\}$ is characterized by a triple $(p_j, \vec{c}_j, \vec{q}_j)$ with processing time $p_j \geq 0$, capacity demand $\vec{c}_j$, and part demand $\vec{q}_j$.

There are $R \geq 0$ types of assembly resources such as mechanical assemblers, electrical assemblers, and power tools. Resource type $r \in \{1, \ldots, R\}$ has a capacity of $C_{r,t} \geq 0$ units at time instant t. The time varying capacity can be caused either from the demand side or from the supply side. On the demand side there might be operations which have been started in the past and are still in process. Capacity is reserved until their planned finish times. On the supply side planned off–time of workers (vacation, fluctuation) and planned down–times of machines (inspection, repair) might reduce the available capacity. Each operation j requires $c_{j,r} \geq 0$ units of resource type r while being processed.

There are $I \geq 0$ types of A–parts which have been fabricated or procured specifically for the planned orders. Delayed A–parts will disrupt or delay the assembly and hence need special management attention (cf . Vaart et al. [30] and Nof et al. [26]). For each part type $i \in \{1, \ldots, I\}$ the quantity on hand is $n_{i,0} \geq 0$. Additionally, the number of parts which will become available at time instant t is $n_{i,t} \geq 0$. The cumulated number of parts which will become available until time instant t is $N_{i,t} = \sum_{\tau=0}^{t} n_{i,\tau}$. The amount and timing of incoming parts are known from vendor contracts and production schedules of in–house part fabrication. Operation j requires $q_{j,i} \geq 0$ units of part type i. Before an assembly operation can be initiated, all required parts must be kitted. As noted by Nof et al. [26], kitting plays a central role in assembly.

The objective of the assembly scheduling problem is to place orders on assembly areas, to assign parts to operations, and to schedule operations subject to precedence and resource constraints such that the sum of the weighted tardiness of the orders is minimized. Note that in contrast to, e.g., Agrawal et al. [2], Chen and Wilhelm [6], and Faaland and Schmitt [10] we do not consider any earliness costs. This is due to the short–term character of our problem where holding costs of parts and availability costs of resource levels have already been determined.

Figure 1 and Table 1 give an example with $A = 3$ orders, $J = 12$ operations, $R = 1$ resource type, and $I = 1$ part type. The capacity of the single resource type is $C_{1,t} = 4$ for each time instant t. The quantity on hand for part type 1 is $n_{1,0} = 2$. Furthermore, 2 part units will become available at time instants 3, 6, and 9, respectively, i.e. $n_{1,3} = n_{1,6} = n_{1,9} = 2$. The capacity of the shop floor is $C = 2$.

2.2 Problem Formulation

We assume that events, i.e. the delivery of parts and the change of available capacity, occur at discrete multiples of a standard period length, e.g., a shift or half shift, and that the processing times of the operations are discrete multiples of the standard period length. In this case, all operation start times will also be discrete multiples of the standard period length. We employ two types of decision variables. First, the binary decision variable $x_{j,t} = 1$, if operation j is started at time instant t, and 0, otherwise (cf. Pritsker et al. [27]). That is, $x_{j,0} = 1$ denotes that operation j starts at time instant $t = 0$, ends at time instant $t = p_j$ and is processed during periods $t = 1, \ldots, p_j$. In order to lower the

a	d_a	w_a	j	p_j	$c_{j,1}$	$q_{j,1}$
1	8	2	2	2	2	1
			3	3	1	2
			4	3	3	1
			5	2	2	
2	6	3	7	2	1	2
			8	1	2	1
			9	2	1	
			10	3	2	
3	7	4	11	3	2	1
			12	2	3	

Table 1: Order and Operation Data

number of $x_{j,t}$ variables we calculate ES_j, earliest operation start times, and LS_j, latest operation start times, by classical network-based forward and backward recursion (cf. Elmaghraby [9]). Forward recursion starts with $ES_0 = 0$ while backward recursion starts from $LS_{J+1} = T$ where T denotes the latest time for all orders to be finished. Note that to assure feasibility, in general, we have to set $T > d^{\max}$. The second decision variable is $T_a \geq 0$, the time span order a is tardy. The short-term assembly scheduling problem (STASP) can now be modelled as follows (cf. Kolisch [19]).

$$\text{Min} \quad Z = \sum_{a=1}^{A} w_a T_a \tag{1}$$

s.t.

$$\sum_{t=ES_j}^{LS_j} x_{j,t} = 1 \qquad (j = 1, \ldots, J) \tag{2}$$

$$\sum_{t=ES_h}^{LS_h} (t + p_h)\, x_{h,t} - \sum_{t=ES_j}^{LS_j} t x_{j,t} \leq -t_{h,j}^{\min} \qquad (j = 1, \ldots, J;\ h \in \mathcal{P}_j) \tag{3}$$

$$\sum_{j=1}^{J} \sum_{\tau=\max\{0,t-p_j+1\}}^{t} c_{j,r} x_{j,\tau} \leq C_{r,t} \qquad (r = 1, \ldots, R;\ t = 0, \ldots, T) \tag{4}$$

$$\sum_{a=1}^{A} \sum_{\tau=0}^{t} \left(x_{s_a,\tau} - x_{e_a,\max\{0,\tau-p_{e_a}\}} \right) \leq C \qquad (t = 0, \ldots, T) \tag{5}$$

$$\sum_{\tau=0}^{t} \sum_{j=1}^{J} q_{j,i} x_{j,\tau} \leq N_{i,t} \qquad (i = 1, \ldots, I;\ t = 0, \ldots, T) \tag{6}$$

$$\sum_{t=ES_{e_a}}^{LS_{e_a}} (t + p_{e_a})\, x_{e_a,t} - T_a \leq d_a \qquad (a = 1, \ldots, A) \qquad (7)$$

$$x_{j,t} \in \{0,1\} \qquad (j = 1, \ldots, J;\ t = ES_j, \ldots, LS_j) \qquad (8)$$

$$T_a \geq 0 \qquad (a = 1, \ldots, A) \qquad (9)$$

The objective function (1) minimizes the sum of the weighted tardiness. (2) forces each operation to be processed. (3) stipulates the technological precedence constraints between operations. (4) guarantees that the capacity of each type of assembly resource is respected at every time instant. Note that at time instant t, capacity is required by all operations which do start or are in process in t but not by operations which finish in t. (5) ensures for every time instant the spatial capacity constraints imposed by the limited availability of the assembly area. (6) models the constraints imposed by the fact that for each part type i and each time instant t the sum of assembled units has to be less equal the sum of the parts which have become available until t. (7) links the continuous tardiness variable with the binary start variables of the order sink. Finally, (8) and (9) define the binary and the continuous decision variables, respectively. Note, that one can model the STASP with the $x_{j,t}$ variables solely. But employing the T_a variables makes the objective function more handy.

It is shown in Kolisch [19] that the STASP is an $\mathcal{NP}$-hard optimization problem. Hence, heuristic algorithms are required when solving large industrial problem instances.

Similar problems as the STASP appear in the assembly of machine tools (cf. Drexl and Kolisch [8]), ships (cf. Lee et al. [22]), and plants (cf. Moder et al. [25]). A review of the assembly scheduling and the multi-project scheduling literature provided in Kolisch [19] reveals that none of the papers presented so far simultaneously considers multiple assembly orders under spatial, resource, and part availability constraints.

3 Solution Procedures — List Scheduling

A very popular approach to solve scheduling problems is list scheduling (cf. Schutten [29]) which works as follows: First, the operations of the scheduling problem are sequentially ordered in a list. Second, in the order given by the list, the operations are scheduled at their earliest feasible start times. Kolisch [19] has devised a list scheduling algorithm especially suited for the STASP. Since it is an integral part of the advanced solution methods which will be discussed in Section 4 we have to revise it in the next section. We will first show how a list can be transformed into a schedule. Afterwards we turn to the generation of so-called feasible lists, i.e. lists which will bring forth a feasible schedule. An elaborated example can be found in Kolisch [19].

3.1 Schedule Generation

Let $\pi = (j_1, j_2, \ldots, j_J]$ be a list of the J non-dummy operations belonging to STASP. j_g is the operation at position g and $\pi(j)$ is the list position of operation j. Let us assume for now that we have such a list π and want to transform it into a feasible schedule $S(\pi)$. A

feasible schedule gives a start time S_j for each operation $j = 1, \ldots, J$ such that precedence relations, part availability, assembly resource, and spatial resource constraints are obeyed. In order to schedule the g–th operation on the list, we need to know the material availability $\tilde{N}_{i,t}(g)$, the assembly capacity $\tilde{C}_{r,t}(g)$, and the spatial resource capacity $\tilde{C}_t(g)$ after the first $g-1$ operations have been scheduled. This can be calculated with the following recursions for $g = 2, \ldots, J$:

For all $i = 1, \ldots, I$ and $t = 0, \ldots, T$ we first initialize $\tilde{N}_{i,t}(1) = N_{i,t}$ and set

$$\tilde{N}_{i,t}(g) = N_{i,t}(g-1) - \begin{cases} q_{j_{g-1},i} & \text{, if } t \in \left\{S_{j_{g-1}}, \ldots, T\right\} \\ 0 & \text{, else} \end{cases} \tag{10}$$

afterwards.

Similar, for the capacity of assembly resources we set $\tilde{C}_{r,t}(1) = C_{r,t}$ for $r = 1, \ldots, R$, $t = 0, \ldots, T$ and update the available capacity

$$\tilde{C}_{r,t}(g) = \tilde{C}_{r,t}(g-1) - \begin{cases} c_{j_{g-1},r} & \text{, if } t \in \left\{S_{j_{g-1}}, \ldots, S_{j_{g-1}} + p_{j_{g-1}} - 1\right\} \\ 0 & \text{, else} \end{cases} \tag{11}$$

for $r = 1, \ldots, R$ and $t = 0, \ldots, T$.

Finally, for the capacity of spatial resources, we set $\tilde{C}_t(1) = C$ for $t = 0, \ldots, T$. Updating can for each $t = 0, \ldots, T$ be done by

$$\tilde{C}_t(g) = \tilde{C}_t(g-1) + \begin{cases} 1 & \text{, if } j_{g-1} \in \mathcal{E} \text{ and } t \in \left\{S_{j_{g-1}} + p_{j_{g-1}}, \ldots, T\right\} \\ -1 & \text{, if } j_{g-1} \in \mathcal{S} \text{ and } t \in \left\{S_{j_{g-1}}, \ldots, T\right\} \end{cases} \tag{12}$$

where $\mathcal{S}$ and $\mathcal{E}$ denote the set of start and end operations as introduced in Section 2.1.

Now, assume that all list predecessors $j_1, \ldots, j_{g-1}$ of operation $j = j_g$ have been scheduled and we want to start j as early as possible. If we only take into account precedence constraints (PC) we obtain S_j^{PC}, the precedence–constrained start time.

$$S_j^{PC} = \max_{h \in \mathcal{P}_j} \left\{S_h + p_h + t_{h,j}^{\min}\right\} \tag{13}$$

With respect to the part availability (PA) we get S_j^{PA}, the part availability–constrained start time.

$$S_j^{PA} = \min_{t=ES_j}^{LS_j} \left\{t \mid \tilde{N}_{i,\tau}(g) \geq q_{j,i} \text{ for all } i = 1, \ldots, I;\ \tau = t, \ldots, T\right\} \tag{14}$$

Considering only the assembly capacity (AC) and the dynamic earliest start time ES_j', we have S_j^{AC}, the assembly resource–constrained start time.

$$S_j^{AC}(ES_j') = \min_{t=ES_j'}^{LS_j} \left\{t \mid \tilde{C}_{r,\tau}(g) \geq c_{j,r} \text{ for all } r = 1, \ldots, R;\ \tau = t, \ldots, t + p_j - 1\right\} \tag{15}$$

Taking only the spatial capacity (SC) into account, we have for order start operations S_j^{SC}, the spatial–constrained start time.

$$S_j^{SC} = \min_{t=ES_j}^{LS_j} \left\{t \mid \tilde{C}_t(g) \geq 1\right\} \tag{16}$$

We now can proceed to the schedule generation algorithm.

Schedule Generation

Initialization: $S_0 = 0$.
For $g = 1$ to J do

(1) Calculate $\tilde{N}_{i,t}(g)$, $\tilde{C}_{r,t}(g)$, and $\tilde{C}_t(g)$

(2) Take the next job from the list: $j = j_g$

(3) Determine the dynamic earliest start time ES'_j of j w.r.t. precedence, part availability, and spatial constraints:
If $j \in S$ then $ES'_j = \max\left\{S_j^{PC}, S_j^{PA}, S_j^{SC}\right\}$, else $ES'_j = \max\left\{S_j^{PC}, S_j^{PA}\right\}$

(4) Determine the earliest resource feasible start time of j which is $\geq ES'_j$:
$S_j = S_j^{AC}(ES'_j)$.

Step (1) updates the part availability and the resource availability of assembly and spatial resources according to (10), (11), and (12). After Step (2) has selected the next operation from the list, its dynamic earliest start time w.r.t. precedence constraints, part availability constraints, and spatial constraints is calculated in Step (3). Note, that for non-start operations no spatial constraints have to be taken into account. Step (4) determines the earliest resource feasible start time within the time window $\left[ES'_j, \ldots, LS_j\right]$. Note, that S_0, the start time of the dummy source of the integrated assembly graph, is initialized although it does not belong to the schedule because this start time is required in Steps (1) and (3).

Applying the schedule generation algorithm to the problem instance given in Section 2.1 while using the list $\pi = \langle 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ we obtain the schedule $S = (0, 0, 3, 6, 9, 9, 9, 11, 12, 14, 0, 3)$ with an objective function value of $Z = 39$. This schedule is given as Gantt-chart in Figure 2. Note that only operations with non-zero processing times are depicted.

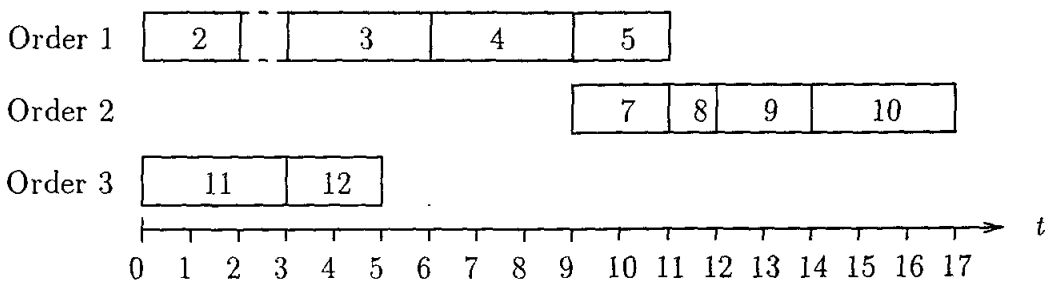

Figure 2: Initial Solution

It has been shown in Kolisch [19] that the schedule generation algorithm generates feasible schedules which are active in the case of $C \geq A$ and which might be non-active for $C < A$.

3.2 List Generation

We now turn to the problem of generating a list $\pi = [j_1, j_2, \ldots, j_J]$. In order to transform a list π into a feasible schedule $S = (S_1, \ldots, S_J)$, two properties corresponding with constraints (3) and (5) have to be met. The first property corresponds to the precedence constraints in (3). It requires that the list position of an operation j must be greater than the list position of each of its predecessors (cf., e.g., Hartmann [16])

$$\tilde{\mathcal{P}}_{j_g} \subseteq \{j_1, \ldots, j_{g-1}\} \quad (g = 1, \ldots, J). \tag{17}$$

The second property concerns the spatial resource constraints given in (5). Consider the available spatial capacity given in (12). Whenever an order start operation s_a $(a = 1, \ldots, A)$ is scheduled, the available spatial capacity is reduced by 1 from the start of that operation to the end of the planning horizon. Scheduling an order end operation e_a $(a = 1, \ldots, A)$ adds 1 capacity unit from the finish time of the operation to the end of the planning horizon. Hence, starting with an available capacity of C at iteration 1 of the schedule generation algorithm, no more than C order sources can be on list positions $1, \ldots, g$ without an order sink in–between. Let $\mathcal{L}(g) = \{j_1, \ldots, j_g\}$ denote the set of operations which have been assigned to list positions $1, \ldots, g$. $\tilde{C}(g)$, the spatial capacity available at the g–th position of the list, is

$$\tilde{C}(g) = C - \mid \mathcal{L}(g-1) \cap \mathcal{S} \mid + \mid \mathcal{L}(g-1) \cap \mathcal{E} \mid . \tag{18}$$

Now, $\tilde{C}(g) \geq 0$ has to hold for each list position $g = 1, \ldots, J$.

Let further $\mathcal{A}(g)$ be the set of all available operations which can be put at list position g. $\mathcal{A}(g)$ is the union of two disjoint sets of operations. $\mathcal{A}^S(g)$, available start (S) operations of orders, $\mathcal{A}^S(g) = \{j \in \mathcal{S} \mid j \notin \mathcal{L}(g-1)\}$ and $\mathcal{A}^N(g)$, available non–start (N) operations, $\mathcal{A}^N(g) = \{j \in \mathcal{J} \setminus \mathcal{S} \mid j \notin \mathcal{L}(g-1),\ \mathcal{P}_j \subseteq \mathcal{L}(g-1)\}$. Both, start and non–start operations respect the precedence constraint property (17). The former because they do not have any (non–dummy) predecessors, the latter by definition. Denoting with $v(j)$ a priority value associated with operation j, we can give the list generation algorithm as follows:

List Generation

Initialization: $\mathcal{L}(0) = \emptyset$.
`For` $g = 1$ `to` J `do`

(1) Update $\tilde{C}(g)$, $\mathcal{A}^N(g)$, and $\mathcal{A}^S(g)$

(2) `If` $\tilde{C}(g) \geq 1$ `then` $\mathcal{A}(g) = \mathcal{A}^S(g) \cup \mathcal{A}^N(g)$ `else` $\mathcal{A}(g) = \mathcal{A}^N(g)$

(3) Choose $j_g \in \mathcal{A}(g)$ with $v(j_g) = \min_{i \in \mathcal{A}(g)} v(i)$

(4) Update $\mathcal{L}(g)$

The Initialization assigns the set of operation which are in the list to be empty. Step (1) updates the spatial capacity and the set of start and non–start operations, respectively.

Step (2) defines the set of available operations. If there is spatial capacity, i.e. $\tilde{C}(g) \geq 1$, then the set comprises start and non–start operations, otherwise, i.e. $\tilde{C}(g) = 0$, the set comprises non–start operations only. In Step (3) we select one operation j from the set of available operations with smallest priority value $v(j)$. Finally, Step (4) updates the set of operations assigned to list positions. Applying the list generation algorithm to the problem instance introduced in Section 2.1 while employing the priority values given in Table 2 we obtain the list $\pi = (11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$.

j	1	2	3	4	5	6	7	8	9	10	11	12
v_j	3	4	5	6	7	8	9	10	11	12	1	2

Table 2: Priority Values

4 Solution Procedures — Advanced Methods

Building upon the list scheduling algorithm given in the last section, we can now turn our attention to more advanced solution methodologies, namely biased random sampling and two different tabu search procedures. All of them employ the fact that each solution can be represented by a feasible list π. Hence, we will consider from now on only lists. The mapping into a schedule is then straightforward as given in Section 3.1.

4.1 Biased Random Sampling

Biased random sampling methods have been mainly used to solve resource–constrained project scheduling problems (cf. Kolisch [17]) and job shop scheduling problems (cf. Baker [4]). A recent survey of biased random sampling methods for the resource–constrained project scheduling problem is given in Kolisch and Hartmann [20]. New biased random sampling approaches are developed in Schirmer and Riesenberg [28]. Feo et al. [11] have introduced a similar solution method called Greedy Randomized Adaptive Search Procedure (GRASP) which has been used to solve, amongst other optimization problems, different type of scheduling problems including the scheduling of printed wiring board assembly (cf. Feo et al. [12]).

The general idea of biased random sampling is to employ a schedule generation scheme n times in a probabilistic way in order to construct at most n different schedules. More precisely, biased random sampling for the STASP works as follows. We use n times the list generation algorithm in order to generate n lists. At iteration $k = 1, \ldots, n$, the objective function value $Z(S(\pi_k))$ associated with list π_k is compared with the incumbent best objective function value $\overline{Z}$. Whenever an improved objective function value has been obtained, the corresponding list is saved as incumbent best solution. The generation of different lists is achieved by employing a slightly modified list generation algorithm. Instead of selecting at Step (3) the available operation with minimum priority value, we select the operation probabilistically. The selection probability of operation $j \in \mathcal{A}(g)$

is calculated as follows. First, we determine for each operation in the available set the regret value $r(j)$ which is the absolute difference between the priority value $v(j)$ of the operation under consideration and the worst priority value of all operations in the decision set, i.e. $r(j) = \max_{i \in \mathcal{A}(g)} v(i) - v(j)$. Afterwards, we calculate the probability $p(j) = r'(j) / \left(\sum_{i \in \mathcal{A}(g)} r'(i)\right)$. The modified regret value $r'(j) = r(j) + 1$ assures two things. First, the denominator cannot be zero and hence for each operation j the selection probability $p(j)$ is always defined. Second, each operation j in the available set $\mathcal{A}(g)$ has a selection probability of $p(j) > 0$. Hence, each list π can be generated and the search space includes the optimal solution.

4.2 Tabu Search Based Large–Step Optimization

In what follows we will introduce two different tabu search procedures. Both are embedded in a large–step optimization method. Large–step optimization methods have been first introduced by Martin et al. [24]; an application of this method to the job shop scheduling problem is given by Lourenço [23]. The general idea is as follows. One starts with an initial solution and tries to improve this solution by employing a local search algorithm. From the obtained local optimal solution it is then proceeded by a large–step to a new solution which again is the starting point for a new local search. That is, the large–step method visits only local optimal solutions which reduces the solution space. The application of large steps secures that the method does not get trapped in local optimal solutions. There are three elements of a large–step optimization algorithm which have to be detailed: *i*) the representation of a solution, *ii*) a specification of 'large steps', and *iii*) the specification of the local search procedure. We will sketch out these elements before detailing them below.

Solution Representation As solution representation we employ the list of assembly operations π. From π we can additionally derive the information of the order sequence λ. $\lambda = \langle a_1, \ldots, a_A]$ gives the sequence orders are placed on the assembly area. E.g., in Figure 2 we have the order sequence $\lambda = \langle 3, 1, 2]$. The order sequence λ can be directly obtained from the operations sequence π because whenever we have $\pi(s_h) < \pi(s_p)$, then $\lambda(h) < \lambda(p)$ holds.

Local Search Procedure The local search method performs a tabu search on the operations sequence π where the order sequence λ is fixed. E.g., in Figure 3 the solution space for order sequence λ_B is B. The dotted arrow $b \rightarrow b'$ shows the path from the initial solution b to the local optimal solution b'.

Large Steps The large steps perform a tabu search on the order sequence λ. In Figure 3 the solid arrow $b' \rightarrow c$ shows the large step from the local optimal solution b' within the solution space B defined by order sequence λ_B to the new solution c within solution space C defined by the new order sequence λ_C.

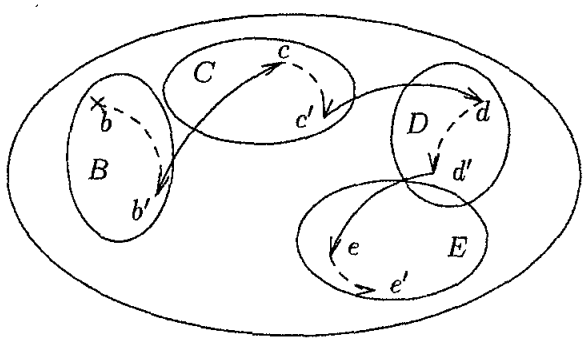

Figure 3: Large–Step Optimization

4.2.1 Principles of Tabu Search

Tabu search (TS) has been originally developed by Glover [14, 15]. It is essentially a steepest descent/mildest ascent method. That is, it evaluates all solutions of the neighborhood and chooses the best one, from which it proceeds further. This concept, however, bears the possibility of cycling, that is, one may always move back to the same local optimum one has just left. In order to avoid this problem, a tabu list is set up as a form of memory for the search process. Usually, the tabu list is employed to forbid those neighborhood moves that might cancel the effect of lately performed moves and might thus lead back to a recently visited solution. Typically, such a tabu status is overrun if the corresponding neighborhood move would lead to a new overall best solution (aspiration criterion). It is obvious that TS extends the simple steepest descent search, often called best fit strategy, which scans the neighborhood and then accepts the best neighbor solution, until none of the neighbors improves the current objective function value.

Key issues in the design of tabu search procedure are the selection of a suitable solution representation, the definition of a neighborhood, and the definition of a tabu list.

4.2.2 Local Search of Operation Sequences

We will now detail the tabu search procedure for finding local optimal operation sequences. We use two different neighborhoods, a simple adjacent pairwise neighborhood and a more elaborated so–called 'critical neighborhood'.

API–Neighborhood The first neighborhood is an adjacent pairwise interchange (API) neighborhood (cf., e.g., Della Groce [7]). A neighbor of sequence π is each precedence feasible sequence π' with $\lambda' = \lambda$, $j'_h = j_{h+1}$ and $j'_{h+1} = j_h$ for one $h = 1, \ldots, J-1$. A precedence feasible list π' is only derived, if operation j_h is no predecessor of operation j_{h+1}, i.e. $j_h \notin \mathcal{P}_{j_{h+1}}$. Figure 4 illustrates the API–neighborhood. The new neighbor π' is derived by the move $M(j_{h+1}, j_h)$ where operation j_{h+1} is shifted in front of operation j_h. The backward move $M(j_h, j_{h+1})$ is set tabu.

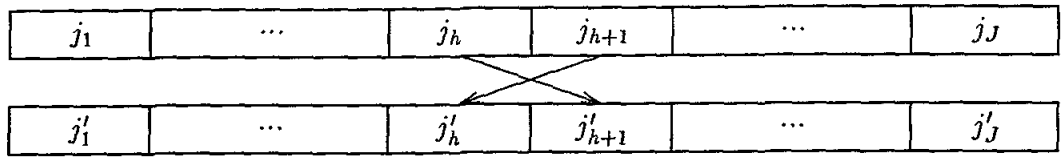

Figure 4: API–Neighborhood

Critical Neighborhood The drawback of the API–neighborhood is the fact that it does not employ any knowledge which is given in the schedule $S(\pi)$ associated with the incumbent list π. In contrast, the so–called 'critical neighborhood' exploits problem specific knowledge given in the incumbent schedule $S(\pi)$. The general idea is to find for an operation j a set of blocking operations $\mathcal{B}_j$ which hampers j from being started one period earlier. A similar idea for the resource–constrained project scheduling problem has been proposed by Baar et al. [3]. The neighbor of π w.r.t. j is obtained by shifting j (and all its predecessors) in front of the operation $h \in \mathcal{B}_j$ which has the smallest list position. More detailed, the neighborhood is as follows. For operation j we define ES_j^{PA}, the earliest feasible start time of operation j w.r.t. part availability.

$$ES_j^{PA} = \min_{t=ES_j}^{LS_j} \left\{ t \mid \forall i \in \{1,\ldots,I\} : N_{i,t} \geq q_{j,i} + \sum_{h \in \tilde{P}_j} q_{h,i} \right\}. \tag{19}$$

An operation j is considered to be left shiftable, if $S_j > \max\left\{S_j^{PC}, ES_j^{PA}\right\}$ holds. S_j^{PC} is the earliest precedence feasible start time as defined in (13). A left shiftable operation does neither start at its precedence nor part availability–based earliest start time. Hence, there must be a set of operations containing at least one operation which prevents j from starting earlier than S_j.

Denoting with $\tilde{C}_{r,S_j-1}$ and $\tilde{N}_{i,S_j-1}$ the available assembly capacity and the material availability w.r.t. the current schedule, we can define the capacity shortage $c_{j,r}^{\Delta}$ and the part availability shortage $q_{j,i}^{\Delta}$, respectively, which hampers operation j from being started at $S_j - 1$.

$$c_{j,r}^{\Delta} = \max\left\{0, c_{j,r} - \tilde{C}_{r,S_j-1}\right\} \tag{20}$$

$$q_{j,i}^{\Delta} = \max\left\{0, q_{j,i} - \tilde{N}_{i,S_j-1}\right\} \tag{21}$$

The set of resource and part types for which there is a capacity or part availability shortage for operation j at $S_j - 1$ is defined as set of critical resources $\mathcal{R}_j^c$ and critical parts $\mathcal{I}_j^c$, respectively.

$$\mathcal{R}_j^c = \left\{r \mid 1 \leq r \leq R, c_{j,r}^{\Delta} > 0\right\} \tag{22}$$

$$\mathcal{I}_j^c = \left\{i \mid 1 \leq i \leq I, q_{j,i}^{\Delta} > 0\right\} \tag{23}$$

The set of resource critical operations $\mathcal{C}_j^R$ contains all operations which are causing some, not necessary all, capacity shortage at $S_j - 1$.

$$\mathcal{C}_j^R = \left\{ h \in \mathcal{J} \mid h \notin \tilde{\mathcal{P}}_j \text{ and } \pi(h) < \pi(j) \text{ and } S_h < S_j \leq S_h + p_h \text{ and } \exists\, r \in \mathcal{R}_j^c :\ c_{h,r} > 0 \right\} \tag{24}$$

The set of part critical operations $\mathcal{C}_j^I$ contains all operations which are causing some, not necessarily all, part availability shortage at $S_j - 1$.

$$\mathcal{C}_j^I = \left\{ h \in \mathcal{J} \mid h \notin \tilde{\mathcal{P}}_j \text{ and } \pi(h) < \pi(j) \text{ and } S_h < S_j \text{ and } \exists\, i \in \mathcal{I}_j^c :\ q_{h,i} > 0 \right\} \tag{25}$$

The set of critical operations $\mathcal{C}_j = \mathcal{C}_j^I \cup \mathcal{C}_j^R$ is a set of operations where each operation is causing some, not necessary all, capacity or part shortage at $S_j - t$.

Let a blocking set $\mathcal{B}_j$ be a mimimal subset of the critical operations $\mathcal{C}_j$ for which the following holds:

$$\forall r \in \mathcal{R}_j^c : \sum_{h \in \mathcal{B}_j} c_{h,r} \geq c_{j,r}^{\Delta} \text{ and } \forall i \in \mathcal{I}_j^c : \sum_{h \in \mathcal{B}_j} q_{h,i} \geq q_{j,i}^{\Delta} \tag{26}$$

'Minimal' means that a blocking set must not contain any other subset for which condition (26) holds. In particular a blocking set does not contain any other blocking set as a subset.

Depending on the set of critical operations, there is more than one blocking set. E.g., consider the left shiftable operation 4 within the schedule given in Figure 2 (cf. also Table 3). Associated with operation 4 are the set of critical operations $\mathcal{C}_4 = \{3, 11\}$ and the two blocking sets $\mathcal{B}_4^1 = \{3\}$ and $\mathcal{B}_4^2 = \{11\}$, respectively. We determine for each left shiftable operation j exactly one blocking set. Thereby we are seeking for the blocking set with lowest cardinality. Starting with cardinality 1 we are looking for a blocking set with only a single operation $h \in \mathcal{C}_j$. In case of more than one possible operation h, we select the operation with the latest start time S_h, in case of ties we choose the operation with the highest list position $\pi(h)$. Is there no blocking set with cardinality one, we search for a blocking set with cardinality two. We start with the operation $h \in \mathcal{C}_j$, with latest start time (highest list position) and search for a second operation $g \neq h, g \in \mathcal{C}_j$ with latest start time (highest list position). Table 3 gives the blocking sets which have been determined for the schedule given in Figure 2.

A list π' where in the corresponding schedule S' operation j shall start earlier than S_j has to put operation j at a list position π'_j which is smaller than the list positions π'_h of all operations h in a blocking set $\mathcal{B}_j$.

The corresponding move $M(j, h)$ is to shift j and all its predecessors which are at list positions between $\pi(h) + 1$ and $\pi(j) - 1$ between operation $j_{\pi(h)-1}$, the immediate list predecessor of h, and h itself. Figure 5 illustrates the move $M(j_i, j_b)$ with j_f being predecessor of j_i. Table 4 provides the four moves associated with the blocking sets of Table 3 and the corresponding neighbor lists with their objective function values.

For all operations k which have been in list π in front of j and are in list π' behind j, i.e. $\pi(k) < \pi(j)$ and $\pi'(j) < \pi'(k)$, we set the move $M(k, j)$ tabu. That is, k is not allowed to be shifted in front of j. Table 4 shows the moves and the associated tabu moves for the example. A move leads always to a precedence feasible list but the new list might

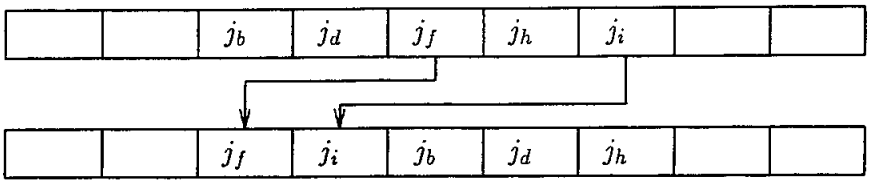

Figure 5: Critical Neighborhood

not be feasible w.r.t. the spatial capacity. Spatial infeasible lists are discarded. In order to obey the order sequence λ we set for all order pairs a and b the move $M(s_b, s_a)$ tabu if $\lambda(a) < \lambda(b)$ holds.

j	1	2	3	4	5	6	7	8	9	10	11	12
$c^{\Delta}_{j,1}$	–	–	0	0	–	–	0	1	–	–	–	–
$q^{\Delta}_{j,1}$	–	–	2	1	–	–	1	0	–	–	–	–
$\mathcal{C}_j$	–	–	{2,11}	{3,11}	–	–	{2,3,4,11}	{5,7}	–	–	–	–
$\mathcal{B}_j$	–	–	{2,11}	{3}	–	–	{3}	{7}	–	–	–	–
Move	–	–	M(3,11)	M(4,3)	–	–	M(7,3)	M(8,7)	–	–	–	–

Table 3: Blocking Sets

Figure 2 gives the schedule of the list $\pi = \langle 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ with objective function value $Z(S(\pi)) = 39$ and Figure 6 gives the schedule of the best neighbor $\pi' = \langle 11, 12, 1, 2, 4, 3, 5, 6, 7, 8, 9, 10]$ with objective function value $Z(S(\pi')) = 30$.

Move	π	Z	Tabu
	$\langle 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$	39	
M(3,11)	$\langle 1, 3, 11, 12, 2, 4, 5, 6, 7, 8, 9, 10]$	47	M(2,3), M(11,3), M(12,3)
M(4,3)	$\langle 11, 12, 1, 2, 4, 3, 5, 6, 7, 8, 9, 10]$	30	M(3,4)
M(7,3)	$\langle 11, 12, 1, 2, 6, 7, 3, 4, 5, 8, 9, 10]$	48	M(3,7), M(4,7), M(5,7)
M(8,7)	$\langle 11, 12, 1, 2, 3, 4, 5, 6, 8, 7, 9, 10]$	33	M(7,8)

Table 4: Moves

4.2.3 Large–Step Optimization of Order Sequences

We have employed the same large–step method for the API–neighborhood and the critical neighborhood. The method can be given as follows.

Let for the incumbent local optimal solution a^* be the order with the highest objective function contribution which is not on the first position of the order sequence. The new

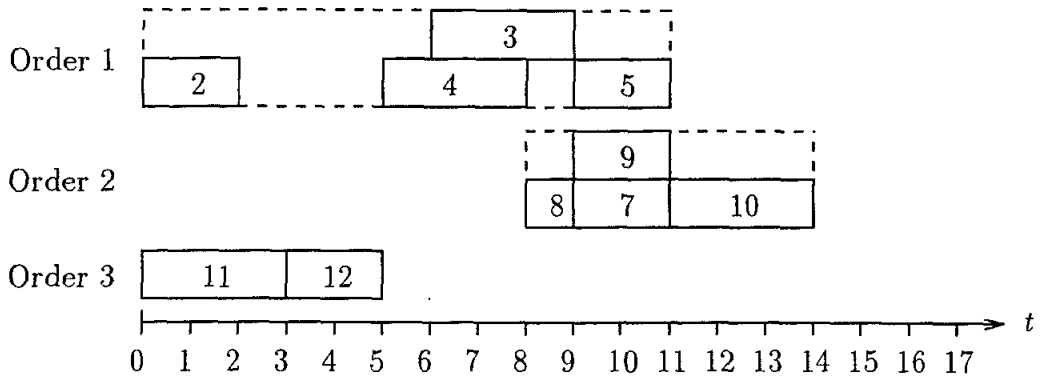

Figure 6: Solution after Move $M(4,3)$

order sequence is obtained by moving a^* to the first position of the sequence, i.e. $a'_1 = a^*$. Any order sequence which has been evaluated is set tabu for the rest of the procedure.

Let us illustrate this large step with the help of the example which has been employed for the critical neighborhood. Consider the solution given in Figure 6. The order with the highest objective value contribution is $a = 2$ with $w_2 \cdot T_2 = 3 \cdot 8 = 24$. We are now changing the order sequence $\lambda = \langle 3,1,2]$ to the new order sequence $\lambda' = \langle 2,3,1]$ by moving order 2 to the first list position. A solution for the new order sequence is obtained by first constructing a feasible operation sequence which is then mapped into a schedule. The construction of the operation list is straight forward. In the order given by λ' we sequence the operations of each order by ascending operation number, i.e. $\pi = \langle s_{a_1}, \ldots, e_{a_1}, \ldots, s_{a_A}, \ldots, e_{a_A}]$. This way we obtain for the order sequence $\lambda' = \langle 2,3,1]$ the operation sequence $\pi = \langle 6,7,8,9,10,11,12,1,2,3,4,5]$. The associated schedule with an objective function value of $Z = 50$ is pictured in Figure 7.

After the large step has been performed, the local search of operation sequences starts anew. Employing the critical neighborhood search, we have the possible moves $M(2,12)$, $M(8,7)$, $M(11,7)$, and $M(12,10)$. The move which leads to a solution with smallest objective function value is $M(11,7)$. The new operation sequence is $\pi = \langle 6,11,7,8,9,10,12,1,2,3,4,5]$. Applying the schedule generation algorithm this sequence is mapped into the schedule given in Figure 8 with an optimal objective function value of 18.

5 Experimental Evaluation

5.1 Test Instances

For evaluation purposes, a set of 270 problem instances was generated with a parameter controlled instance generator for assembly type problems which builds upon ProGen (cf. Kolisch et al. [21]). The instance set can be divided w.r.t. the size into small and large instances. A full factorial experimental design with three independent problem parameters was employed for both sets. Each of the independent problem parameters corresponds with one of the 'hard' constraints (4), (5), and (6) of STASP, respectively. All other parameters

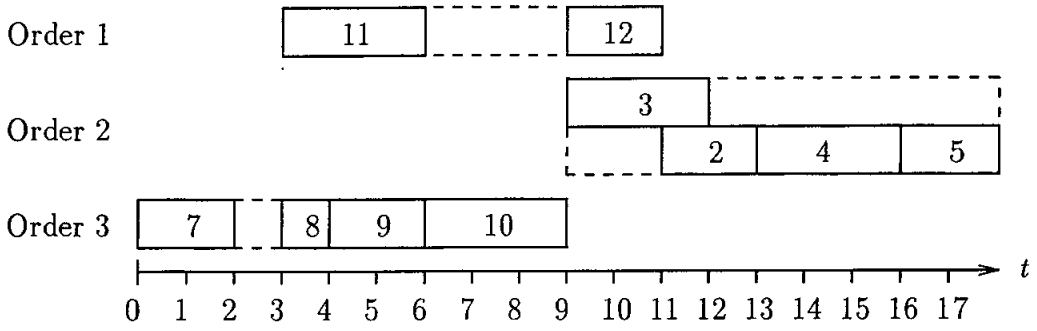

Figure 7: Solution after Application of a Large Step

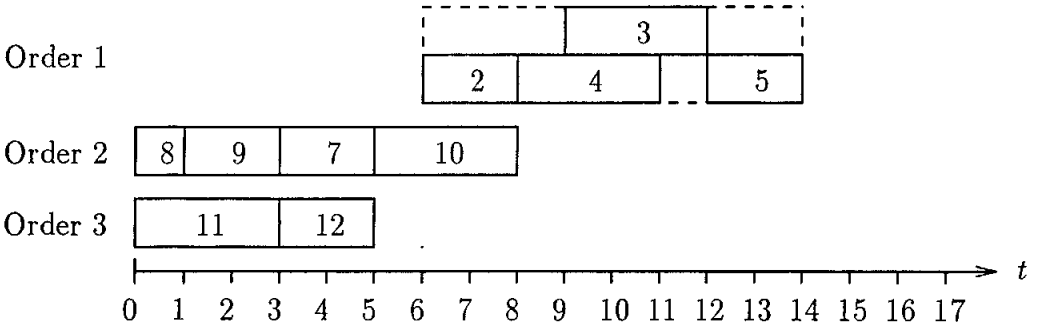

Figure 8: Optimal Solution

were randomly drawn from the given intervals.

For the small test instances, there is $A = 3$, $J_a \in [3,5]$, $R = 1$, $RF = 1$, $I = 1$, $PF = 0.7$, and $r_a \in [0,5]$. J_a denotes the count of operations belonging to an order, RF denotes the resource factor of all operations. It measures the density of constraint type (4) (cf. Kolisch et al. [21]). Correspondingly, PF denotes the part factor which measures the density of constraint type (6). $RF = 1$ and $PF = 0.7$ express the fact that for all non–dummy operations there is $c_{j,r} > 0$ and for 7 out of 10 non–dummy operations we have $q_{j,r} > 0$. The following parameters were randomly drawn from the specified intervals: $p_j \in [1,3]$, $c_{j,r} \in [1,3]$, $w_a \in [1,5]$, $q_{j,i} \in [1,2]$. The due date d_a was calculated as follows: $d_a = r_a + ES_{e_a}$ with r_a, the release date of order a.

The following parameter values were set differently for the large test instances: $A = 10$, $J_a \in [5,10]$, $R = 2$, $I = 2$, and $r_a \in [0,20]$.

The three independent problem parameters are the assembly resource strength, the spatial resource strength, and the part strength. The assembly resource strength (RS^A) measures the scarcity of the assembly resource capacity given in (4). For $RS^A = 0$, the capacity for each resource type $r = 1,\ldots,R$ equals the minimum capacity needed when no two operations are processed in parallel. For $RS^A = 1$, the capacity for each resource type r suffices to realize, w.r.t. constraint (4), the schedule $S = (ES_1,\ldots,ES_J)$. RS was set to 0.1, 0.3, and 0.5 for the small instances and to 0.1, 0.2, and 0.3 for the large instances,

respectively. The spatial resource strength (RS^S) measures the available capacity of the assembly area. For $RS^S = 0$ there is just one assemble area available, while for $RS^S = 1$, there is enough capacity to assemble all orders in parallel. RS^S was set to 0.1. 0.5, and 1.0 for the small instances and to 0.3, 0.5, and 1.0 for the large instances, respectively. The part strength (PS) measures the timing, parts are made available for assembly. For $PS = 1$, each part type is made available such that, w.r.t. constraint (6), the schedule $S = (ES_1, \ldots, ES_J)$ can be realized while for $PS = 0$, operations cannot be started earlier than given in the schedule $S = (LS_1, \ldots, LS_J)$. Latest start times LS_j are calculated by backward recursion from the upper bound $T = \sum_{j=1}^{J} p_j$. For the small instances, PS was set to 0.7, 0.8, and 0.9, while for the large instances, it was set to 0.8, 0.9, and 1, respectively. Additionally, some variance was added to the part arrival times by using the part variability (PV). For $PV = 0$ there is no variation and parts arrive as calculated by the part strength. For $PV = 1$ and $PS = 0.5$ part arrival is randomly drawn out of the interval defined by the earliest and latest start times of the part requesting operations. PV was set to 0.3 for the small instances and to 0.4 for the large instances, respectively. Realizing a full factorial design with 5 replications for each combination of the independent parameter levels, $5 \cdot 3^3 = 135$ instances were generated for the small and the large instance set, respectively.

5.2 Results

5.2.1 Parameter Adjustments

Within the experimental investigation we have applied the following four methods: Random sampling, biased random sampling based on the weighted earliest due date priority rule (WEDD), large–step optimization with the API–neighborhood, and large–step optimization with the critical neighborhood.

Random Sampling Random sampling is as detailed in Section 4.1. In Step (3) of each list generation run the selection probability for each available operation is set equal, i.e. $p(j) = 1/ \mid \mathcal{A}(g) \mid$. We use random sampling as a benchmark.

Biased Random Sampling In Kolisch [17, 18] it has been experimentally shown that the performance of biased random sampling methods relies significantly on the quality of the employed priority rule. When used with the best deterministic priority rules, biased random sampling will also show the best results. Hence, we employed for biased random sampling the weighted earliest due date (WEDD) rule which is the best performing priority rule for solving the STASP (cf. Kolisch [19]). The calculation of the deterministic priority value $v(j)$ for WEDD is simply by dividing the due date of the order associated with operation j by the weight of the order associated with j. If we denote with $a(j)$ the order a operation j belongs to, the priority value $v(j)$ is as follows: $v(j) = d_{a(j)}/w_{a(j)}$.

Large–Step Optimization with API–Neighborhood The length of the tabu list has been set according to preliminary experiments to 20. The local search for a given order list is stopped whenever 20 consecutive moves have not given an improved solution. Each

visited order list is set tabu until the entire solution procedure commences. The aspiration level has been used. The number of evaluated operation lists is set to a maximum of 5000.

Large–Step Optimization with Critical Neighborhood For the critical neighborhood we set the length of the tabu list to 30, the entire number of evaluated solutions is set to a maximum of 5000. The aspiration level has been used. The local search for a given order list λ is stopped whenever Z^*_λ, the incumbent best objective function value associated with order list λ, exceeds after *iter* accepted operation lists the so far best objective function value $\overline{Z}$ by at least ϵ %, i.e. $\left(Z^*_\lambda - \overline{Z}\right)/\overline{Z} \cdot 100 > \epsilon$ %. According to preliminary computational results we have chosen $\epsilon = 0$ and $iter = 5$. That is, for a given order sequence λ, the tabu search of operation sequences is stopped after 5 iterations if one has not found a solution which is as least as good as the so far best found objective function value $\overline{Z}$.

5.2.2 Computational Results

Table 5 reports the average deviation for random sampling (RS), biased random sampling (BRS), large–step optimization with the API–neighborhood (API–LSO), and large–step optimization with the critical neighborhood (CN–LSO) from the lower and the upper bound of the objective function value, respectively. Additionally, the average number of evaluated solutions are given. A distinction is made w.r.t. the problem size for all data. The bounds have been obtained as follows. For the lower bound we solved for each small instance the MIP–model (1) - (9) and for each large instance the LP–relaxation of (1) - (9) with CPLEX (cf. Bixby and Boyd [5]). Modelling was done with AMPL (cf. Fourer et al. [13]). Note that for the small instances the term 'lower bound' coincides with the optimum. The upper bounds were calculated for each instance by taking the best objective function value derived with the 4 heuristics RS, BRS, API–LSO, and CN–LSO. Note that the upper bounds employed in Kolisch [19] are different than the ones employed here; on average they are larger than the upper bounds obtained in this study.

The sampling heuristics give better results for the small problem instances whereas the large–step optimization methods show superior results for the large problems. Note, that the superior results of the sampling heuristics on the small instances is solely caused by the far greater number of visited solutions. Based on the large instances there is clear order of performance of the four methods: CN–LSO performs best, followed by API–LSO and then BRS. RS gives the poorest results. The number of visited solutions is much smaller for the LSO–methods. The critical neighborhood–based large–step optimization method clearly outperforms the API–based one. As it is the case for solving many hard optimization problems, we see that using problem insight pays in terms of the goodness of the obtained solutions. Using the $iter/\epsilon$–stopping criterion pays for the critical neighborhood–based large–step optimization. The number of visited solutions is significantly smaller than for API–LSO.

Table 6 reports the effect of the independent problem parameters on the average deviation and the number of visited solutions of the four methods. The value in parenthesis gives the level of confidence when testing with the nonparametric Kruskal–Wallis test whether the problem parameter under consideration has a significant influence.

Method	avg. dev. lb small	avg. dev. lb large	avg. dev. ub small	avg. dev. ub large	visited solutions small	visited solutions large
CN-LSO	1.43	43.63	1.43	3.80	86	2411
API-LSO	2.18	49.40	2.18	7.81	264	3832
BRS	0.10	65.98	0.10	19.42	5000	5000
RS	0.00	87.15	0.00	32.32	5000	5000

Table 5: Average Deviation and Number of Visited Solutions

The effect of the assembly resource strength RS^A on the average percentage deviation from the lower bound of all three heuristics is significant at the 0% level of confidence. The solution quality of all three solution procedures deteriorates when the RS^A-level is lowered, i.e. capacity becomes scarce. The critical neighborhood-based large-step optimization method is the only heuristic which shows a significant decrease of the number of visited solutions when capacity becomes abundant.

Both sampling methods show a significant decrease in the average deviation from the lower bound when the spatial resource strength is increased, i.e. the shop floor capacity becomes abundant. The explanation is that a larger spatial capacity allows to process more orders in parallel which, in turn, increases the number of available operations at Step (3) of the list generation algorithm. This lowers the selection probability of the operations which would lead to the best schedule. The two LSO-methods show for increasing RS^S-values also a decreasing but no significant decrease in the solution quality; the number of visited schedules decreases significantly and sharply when $RS^S = 1.0$, i.e. the spatial resource constraints are not binding any more.

A low level of the part strength, i.e. a late delivery of parts, gives way to better results for all four heuristics. But the extent of the effect is different. For both sampling methods there is a significant influence while the influence on the LSO-methods is not significant.

6 Summary and Conclusions

We have considered the problem of scheduling multiple, large-scale, make-to-order assemblies under resource, assembly area, and part availability constraints. Based on a list scheduling procedure proposed in the literature we introduced three efficient heuristic solution methods, namely, biased random sampling, API-based large-step optimization, and critical neighborhood-based large-step optimization. We assessed all three methods on a systematically generated set of test instances. Especially the critical neighborhood-based large-step optimization heuristic shows favourably results. On the large instances it generates significantly better solutions than the other methods while, at the same time, it requires less visited solutions. Hence, this method should be employed in Leitstand-systems (cf., e.g., Adelsberger and Kanet [1] and Drexl and Kolisch [8]) for the short-term scheduling of make-to-order assemblies.

Acknowledgement. We thank Andreas Drexl for his continuous support and the Deutsche

		RS		BRS		API-LSO		CN-LSO	
Parameter	Level	%	#	%	#	%	#	%	#
	0.1	114.91	5000	87.98	5000	66.33	3827	58.98	2914
RS^A	0.2	78.34	5000	58.83	5000	44.75	3849	39.68	2184
	0.3	68.20	5000	51.11	5000	37.10	3821	32.22	2133
		(0.00)	(1.00)	(0.00)	(1.00)	(0.00)	(0.71)	(0.00)	(0.01)
	0.3	59.62	5000	48.96	5000	48.65	5000	42.15	2822
RS^S	0.5	77.40	5000	57.59	5000	48.07	5000	41.86	3155
	1.0	124.43	5000	91.38	5000	51.46	1449	46.87	1255
		(0.00)	(1.00)	(0.00)	(1.00)	(0.49)	(0.00)	(0.32)	(0.00)
	0.8	65.02	5000	48.16	5000	33.83	3839	29.66	2584
PS	0.9	79.89	5000	60.23	5000	43.24	3839	37.99	2151
	1.0	116.54	5000	89.54	5000	71.12	3819	63.23	2498
		(0.00)	(1.00)	(0.00)	(1.00)	(0.00)	(0.14)	(0.00)	(0.54)

Table 6: Effect of the Problem Parameters — Average Deviation from the Lower Bound and Number of Visited Solutions for the Large Instances

Forschungsgemeinschaft for grant Ko 1680/1-1.

References

[1] H.H. Adelsberger and J.J. Kanet. The leitstand — A new tool for computer-integrated manufacturing. *Production and Inventory Management Journal*, 1:43–48, 1991.

[2] A. Agrawal, G. Harhalakis, I. Minis, and R. Nagi. "Just-in-time" production of large assemblies. *IIE Transactions*, 28:653–667, 1996.

[3] T. Baar, P. Brucker, and S. Knust. Tabu-search algorithms for the resource-constrained project scheduling problem. Technical report, Osnabrücker Schriften zur Mathematik, Fachbereich Mathematik/Informatik, Osnabrück, 1997.

[4] K.R. Baker. *Introduction to sequencing and scheduling.* Wiley, New York, 1974.

[5] N. Bixby and E. Boyd. *Using the CPLEX callable library.* CPLEX Optimization Inc., Houston, 1996.

[6] J.F. Chen and W.E. Wilhelm. Optimizing the allocation of components to kits in small-lot, multi-echelon assembly systems. *Naval Research Logistics*, 41:229–256, 1994.

[7] F. Della Croce. Generalized pairwise interchanges and machine scheduling. *European Journal of Operational Research*, 83:310–319, 1995.

[8] A. Drexl and R. Kolisch. Assembly management in machine tool manufacturing and the PRISMA–Leitstand. *Production and Inventory Management Journal*, 37(4):55–57, 1996.

[9] S.E. Elmaghraby. *Activity networks: Project planning and control by network models.* Wiley, New York, 1977.

[10] B. Faaland and T. Schmitt. Scheduling tasks with due dates in a fabrication/assembly process. *Operations Research*, 35(3):378–381, 1987.

[11] T.A. Feo, J.F. Bard, and R.F. Claflin. An overwie of GRASP methodology and applications. Technical report, Department of Mechanical Engineering, University of Texas at Austin, 1991.

[12] T.A. Feo, J.F. Bard, and S.D. Holland. Facility–wide planning and scheduling of printed wiring board assembly. *Operations Research*, 43(2):219–230, 1995.

[13] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL — A modeling language for mathematical programming.* boyd & fraser, Danvers, 1993.

[14] F. Glover. Tabu search — Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[15] F. Glover. Tabu search — Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[16] S. Hartmann. A competitive genetic algorithm for resource–constrained project scheduling. Technical Report 451, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997. Naval Research Logistics, to appear.

[17] R. Kolisch. *Project scheduling under resource constraints — Efficient heuristics for several problem classes.* Physica, Heidelberg, 1995.

[18] R. Kolisch. Serial and parallel resource–constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90:320–333, 1996.

[19] R. Kolisch. Integrated scheduling, assembly area–, and part–assignment for large scale make–to–order assemblies. Technical Report 468, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, Kiel, 1997.

[20] R. Kolisch and S. Hartmann. Heuristic algorithms for the resource–constrained project scheduling problem: Classification and computational analysis. In J. Weglarz, editor, *Handbook on recent advances in project scheduling.* Kluwer, Amsterdam, 1998. to apppear.

[21] R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource–constrained project scheduling problems. *Management Science*, 41(10):1693–1703, 1995.

[22] J.K. Lee, K.J. Lee, H.K. Park, J.S. Hong, and J.S. Lee. Developing scheduling systems for Daewoo shipbuilding: DAS project. *European Journal of Operational Research*, 97:380–395, 1997.

[23] H.R. Lourenço. Job–shop scheduling: computational study of local search and large–step optimization methods. *European Journal of Operational Research*, 83:347–364, 1995.

[24] O. Martin, S.W. Otto, and E.W. Felten. Large–step markov chains for TSP incorporating local search heuristics. *Operations Research Letters*, 11:219–224, 1992.

[25] J.J. Moder, C.R. Phillips, and E.W. Davis. *Project management with CPM, PERT and precedence diagramming.* Van Nostrand Reinhold, New York, 3. edition, 1983.

[26] S.Y. Nof, W.E. Wilhelm, and H.-J. Warnecke. *Industrial Assembly.* Chapman & Hall, London, 1997.

[27] A.A.B. Pritsker, L.J. Watters, and P.M. Wolfe. Multiproject scheduling with limited resources: A zero–one programming approach. *Management Science*, 16:93–107, 1969.

[28] A. Schirmer and S. Riesenberg. Parameterized heuristics for project scheduling — Biased random sampling methods. Technical Report 456, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 1997.

[29] J.M.J. Schutten. List scheduling revisited. *Operations Research Letters*, 18:167–170, 1996.

[30] J.T. van der Vaart, J. de Vries, and J. Wijngaard. Complexity and uncertainty of materials procurement in assembly situations. *International Journal of Production Economics*, 46–47:137–152, 1996.