Jena Research Papers in Business and Economics

Sequencing Mixed-Model Assembly Lines to Minimize the Number of Work Overload Situations

Nils Boysen, Mirko Kiel, Armin Scholl

05/2010

Jenaer Schriften zur Wirtschaftswissenschaft

Working and Discussion Paper Series
School of Economics and Business Administration
Friedrich-Schiller-University Jena

ISSN 1864-3108

Publisher:

Wirtschaftswissenschaftliche Fakultät
Friedrich-Schiller-Universität Jena
Carl-Zeiß-Str. 3, D-07743 Jena

www.jbe.uni-jena.de

Editor:

Prof. Dr. Hans-Walter Lorenz
h.w.lorenz@wiwi.uni-jena.de

Prof. Dr. Armin Scholl
armin.scholl@wiwi.uni-jena.de

www.jbe.uni-jena.de

Sequencing Mixed-Model Assembly Lines to Minimize the Number of Work Overload Situations

Nils Boysen[a,*], Mirko Kiel[b], Armin Scholl[b]

[a] *Friedrich-Schiller-University Jena, Chair of Operations Management, Carl-Zeiß-Straße 3, D-07743 Jena, Germany,*
nils.boysen@uni-jena.de

[*] *Corresponding author, phone +49 3641 943100.*

[b] *Friedrich-Schiller-University Jena, Chair of Management Science, Carl-Zeiß-Straße 3, D-07743 Jena, Germany,*
mirko-kiel@web.de, armin.scholl@uni-jena.de

The mixed-model sequencing problem is to sequence different product models launched down an assembly line, so that work overload at the stations induced by direct succession of multiple labor-intensive models is avoided. As a concept of clearing overload situations, especially applied by Western automobile producers, a team of cross-trained utility workers stands by to support regular workforce. Existing research assumes that regular and utility worker assemble side-by-side in an overload situation, so that processing speed is doubled and the workpiece can be finished inside a station's boundaries. However, in many real-world assembly lines the application of utility workers is organized completely different. Whenever it is foreseeable that a work overload will occur in a production cycle, a utility worker takes over to exclusively execute work whereas the regular worker omits the respective cycle and starts processing at the successive workpiece as soon as possible. The paper investigates this more realistic sequencing problem and presents a binary linear program along with a complexity proof. Then, different exact and heuristic solution procedures are introduced and tested. Additional experiments show that the new model is preferable from an economic point of view whenever utility work causes considerable setup activities, e.g., walking to the respective station.

Keywords: Mixed-Model Assembly Lines; Sequencing; Utility Work; Branch and Bound; Tabu Search

1 Introduction

In today's mixed-model assembly lines, e.g., of automobile industry, billions of models (for detailed figures see, e.g., Boysen et al., 2009a) of a common base product can be manufactured in facultative production sequence (lot size one). However, in spite of (nearly) any succession of models being technically possible, the actual sequence very well influences different economic targets. For instance, depending on their specification, models require diverging processing times at the stations of the line. Whenever multiple labor-intensive models, e.g., all having an elaborate option, follow each other in direct succession at a specific station, a work overload situation occurs, which means that the worker is not able to finish his work prior to reaching the right-hand (down-stream) station border. To reduce such overload situations (and a cost-intensive compensation, e.g., by applying utility workers or line stoppage), the mixed-model sequencing problem was introduced. Within this sequencing approach, a detailed scheduling is applied, which explicitly takes operation times, worker movements, station borders and other operational characteristics of the line into account. This way, (total) work overload can exactly be quantified and, thus, be minimized.

Depending on the characteristics of the assembly system, a magnitude of different problem types of mixed-model sequencing were introduced in the literature. For a recent survey paper see Boysen et al. (2009b). A major distinctive criterion between these approaches is the assumption with regard to the compensation of imminent work overload. One line of research (e.g., Celano et al., 2004; Xiaobo and Ohno, 1994, 1997, 2000; Yoo et al., 2005) investigates the kind of compensation promoted by the famous Toyota Production System, which is to stop the line via a so-called "Andon Cord" until the overloaded station has finished work. Other researchers (e.g., Bolat, 1997; Bolat and Yano, 1992a,b; Scholl et al., 1998; Sumichrast et al., 2000; Yano and Bolat, 1989; Yano and Rachamadugu, 1991) present multiple solution procedures being based on the application of cross-trained utility workers (floaters). For these utility workers, it is assumed that they support regular workforce at an overloaded station side-by-side, so that processing speed is doubled and the workpiece can be finished right in time when reaching the right-hand border of the respective station (see Boysen et al., 2009b). However, it is our observation at different major European car manufacturers that the application of utility workers is organized completely different in many real-world assembly lines. Whenever it is foreseeable that regular workforce (with normal processing speed) cannot complete work at the next workpiece before reaching the station's border, the respective worker simply skips the respective workpiece. Instead, work overload is compensated by a utility worker, who completely takes over work on the workpiece in the overloaded production cycle, whereas the regular worker proceeds with the succeeding workpiece as soon as it enters the station.

The paper on hand is the first to investigate this kind of organizing utility work. The resulting sequencing problem is described and analyzed in detail within Section 2 and formalized by a binary linear program in Section 3. Then, an exact branch-and-bound procedure and heuristic procedures, i.e., a greedy algorithm and a tabu search approach, are presented in Section 4. The computational study of Section 5 tests the performance of the solution procedures. Furthermore, it is investigated which policy of organizing utility work requires a longer overall duration of utility operations and results in more overload situations, so that the practitioner gets decision support which kind of organizing utility work will be preferable in a specific problem setting. The final Section 6 gives a summary and discusses future research questions.

2 Problem description and analysis

Existing research on assembly lines considering floaters explicitly (e.g., Scholl et al., 1998; Boysen et al., 2009b) or implicitly relies on the following assumptions with regard to organizing utility work, which we label the **side-by-side policy**:

Whenever a regular assembly worker (with normal processing velocity) cannot finish the current workpiece inside his/her station's boundaries work overload threatens. Then, to avoid an obstruction of the subsequent station and a delay in starting work at succeeding product units at the same station, a utility worker is called to support regular workforce. This support is assumed to double processing speed, i.e., to halve required (residual) processing time, and to start exactly at that point in time which allows to finish

work just at the right-hand border of the station (cf. Yano and Bolat, 1989). Note that w.l.o.g. assembly lines are assumed to flow from left to right.

The assumptions of the side-by-side policy are quite restrictive, so that it might be complicated or even impossible to implement it in the real-world for the following reasons:

- A part to be assembled (and even a workpiece) might be comparatively small, so that space is restricted and both workers cannot work simultaneously at the same workplace (see Becker and Scholl, 2009).
- At least, both workers might obstruct each other, so that, typically, processing speed is not exactly doubled but raises sub-proportionally and, moreover, varies from model to model and station to station.
- Because of the aforementioned reason it might also be difficult to exactly compute the point in time from that on the utility worker is required.
- In many real-world settings, it is rather unrealistic to assume that a utility worker arrives just-in-time at a station, is engaged there for some seconds, then immediately returns to his/her regular work, before he/she walks to another station for some other seconds of utility work. Calling a utility worker breaks his/her normal work and leads to a longer engagement often (at least) throughout a complete cycle even if only little utility work is required.

Thus, the side-by-side policy can seldom be found in the real-world. Instead, utility work is often (e.g., by most major European car manufacturers) organized as follows, which we label the **skip policy**:

To each segment of the final assembly line (each consisting of multiple stations) a group of workers is assigned. The assignment of group members to stations, where they constitute the regular workforce, is decided by the group leader. The leader himself represents the team in inter-group processes and is mainly employed with intra-group planning tasks. Furthermore, the group leader gets cross-trained for all tasks of the respective line segment and acts as a utility worker for his group. The group leader is called up by a regular worker, as soon as it is foreseeable that the current workpiece cannot be finished before reaching the right-hand border of the station. Then, the utility worker exclusively takes over work and completely processes the active workpiece, whereas the regular worker *skips* this (overload causing) workpiece and starts with the subsequent workpiece as early as possible.

Example: Consider an assembly line with merely a single station. The length of the station is 13 time units (TU), i.e., each workpiece is in the station area for 13 TU, and cycle time amounts to 10 TU, i.e., every 10 TU another workpiece is launched down the line. Note that throughout this paper, w.l.o.g., conveyor velocity is normalized to 1 LU/TU, so that length units (LU) and time units can be used interchangeably. Furthermore, assume two models, where the first (M_1) needs an processing time of 12 TU and has a demand of four units, while the second model (M_2) needs 7 TU and is required only once. A model sequence π let be given as follows: $\pi =< M_1, M_2, M_1, M_1, M_1]$.

Figure 1 represents the workers' movements in the station for the side-by-side policy. Diagonal lines represent the movement of workers accompanying a workpiece during assembly, where a bold line is used to indicate additional utility work. Dashed horizonal lines depict the return movement to the next workpiece assumed to consume zero time and bold vertical lines mark idle times.
Figures 2 and 3 show the independent movements of both operator types for the skip policy. The figures show, that organizing utility work side-by-side results in a total work overload (utility work) of 3 TU in two overload situations for the given sequence, whereas the skip policy requires 12 TU of utility work in only one overload situation. Note that a detailed investigation of differences in duration and number of overload situations between both policies will be part of our computational study in Section 5.

To formalize the resulting mixed-model sequencing problem applying the skip policy, the following additional assumptions, which are frequently fulfilled in real-world lines, are introduced (for the notation used see Table 1):

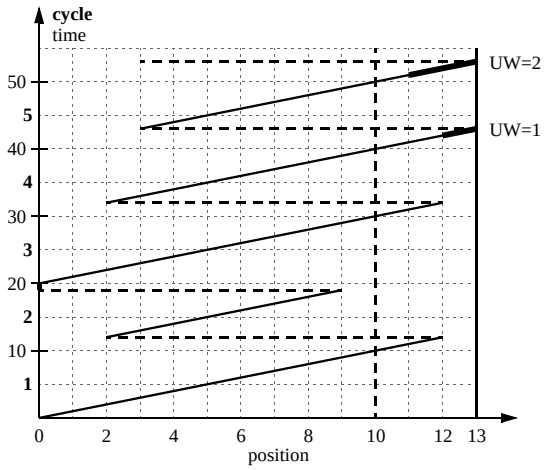

Figure 1: Side-by-side policy: Movements of regular & utility worker

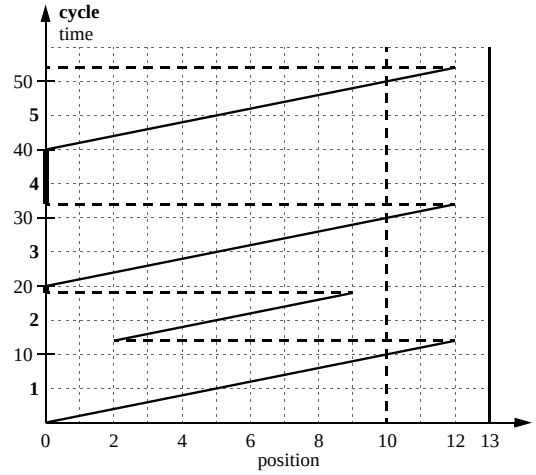

Figure 2: Skip policy: Regular worker movements

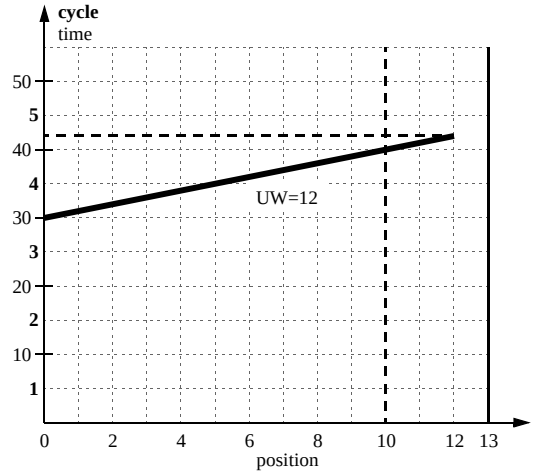

Figure 3: Skip policy: Utility worker movements

- The model-mix, i.e., the demand for models throughout the planning horizon (usually a shift or day), and all processing times per model and station are known with certainty (static and deterministic problem).
- Working across the stations' boundaries is not possible (closed stations), which in the real-world is often caused by a restricted availability of machinery and prohibitive distances to material boxes. Furthermore, the line continuously moves with a constant velocity and there are no buffers between stations.
- The operators return with infinite velocity to the next workpiece. This is an adequate simplification whenever the conveyor speed is much slower than the walking speed of workers. If this was not the case, the available cycle time could be reduced by the (average) return time. Furthermore, it is assumed that processing starts as soon as worker and workpiece meet inside a station.
- Fixed rate launching is applied, so that consecutive units are placed on the line at equidistant intervals equal to cycle time c.
- Furthermore, it is assumed that all models can be processed inside a station when starting work at the left-hand border: $p_{mk} \leq l_k \; \forall \; m \in M; \; k \in K$. Otherwise, sequence-independent work overload exists and a utility worker would automatically be required with any occurrence of the respective model.

M	set of models (index m)
T	number of production cycles (index t)
K	set of stations (index k)
d_m	demand for copies of model m
l_k	length of station k
c	cycle time
p_{mk}	processing time of model m in station k
b_{kt}	continuous variable: processing time of the model assembled in cycle t at station k
s_{kt}	continuous variable: starting position of regular worker in station k (relative to left-hand border) when cycle t begins
x_{mt}	binary variable: $\begin{cases} 1, & \text{if a unit of model } m \text{ is assigned to cycle } t \\ 0, & \text{otherwise} \end{cases}$
y_{kt}	binary variable: $\begin{cases} 1, & \text{if a utility worker is required at station } k \text{ in cycle } t \\ 0, & \text{otherwise} \end{cases}$

Table 1: Notation

- Whenever regular and utility worker simultaneously work on different workpieces of successive production cycles, it is assumed that no obstructions, e.g., with regard to material fetching and resource availability, exist.
- The station length is not greater than twice the cycle time, i.e., $l_k \leq 2c \; \forall \; k \in K$. This assumption ensures that at most two workers are working inside a station at a time and that never two utility workers are simultaneously required within a single station as this would come along with considerable organizational difficulties due to space requirements and staff management. As already mentioned, a simultaneous engagement of two utility workers at the same station is not possible when only the team leader acts as a floater which is typical in Western automobile plants.
- It is assumed that each utility worker has a regular work content, e.g., planning tasks of the group leader, from which he/she must be released while acting as a floater. In such a setting, minimizing the overall number of overload situations is a suited objective function, so that the number of breaks pulling utility workers out of their regular work is minimized.

Given these assumptions, the following theorem shows that a premature application of a utility worker prior to the overloaded cycle cannot be advantageous.

Theorem 1: There exists at least one optimal solution with utility workers only assigned to overloaded cycles.

Proof: We prove this optimality condition by comparing time and movement calculations for two situations, one with utility work in an overloaded cycle and the other with premature utility work. As a reallocation of floater assignments within one station does not have any effect on other stations (closed stations), it is sufficient to consider a single station k:

Given an overload situation occurring in cycle $t < T$, i.e., $s_{kt} + b_{kt} > l_k$ (condition (a)). Then, cycle $t-1$ cannot be overloaded, i.e., $s_{k,t-1} + b_{k,t-1} \leq l_k$ (condition (b)). Condition (b) must hold, because otherwise utility work would have been necessary in $t-1$ (with an overload in cycle t not being possible). Given a closed station, the starting time s_{kt} (in relation to the left station border) of the regular worker at station k in cycle t is either determined by the left station border $s_{kt} = 0$ or by the delayed return from the previous workpiece, so that $s_{kt} = \max\{s_{k,t-1} + b_{k,t-1} - c, 0\}$ holds. Thus, by insertion into overload condition (a) of station t, we get $\max\{s_{k,t-1} + b_{k,t-1} - c, 0\} + b_{kt} > l_k$. Clearly, to fulfill condition (b) for cycle $t-1$ of not being overloaded, $b_{kt} > c$ must hold. This allows for the following comparison.

Algorithm 1: Calculation of a given sequence

```
/* Initialize starting positions in t=1
for k := 1 to |K| do
    s[k,1] := 0;
/* Calculate sequence positions in forward direction
for t := 1 to T do
    /* Calculate each station separately
    for k := 1 to |K| do
        /* Enough time to finish workpiece of cycle t?
        if s[k,t] + b[k,t] <= l[k] then
            s[k,t+1] := max(s[k,t] + b[k,t] - c, 0);   /* Compute start position for next workpiece
            y[k,t] := 0;   /* Mark that no work overload is required
        else
            s[k,t+1] := max(s[k,t] - c, 0);   /* Regular worker omits workpiece of cycle t
            y[k,t] := 1;   /* Work overload requires a utility worker
for k := 1 to |K| do
    if s[k,T+1] > 0 then
        y[k,T] := 1;   /* Use utility work to finally return to left-hand borders, if necessary
```

If utility work is assigned to the overloaded cycle t, the following starting positions are calculated. Note that cycle t is omitted by the regular worker and the floater starts at the left-hand border. So, s_{kt} only defines the point where the regular worker *would* start work in cycle t if he did.

$$s_{kt} := \max\{s_{k,t-1} + b_{k,t-1} - c, 0\}$$
$$s_{k,t+1} := 0$$

If utility work is assigned earlier, e.g., to cycle $t-1$, the starting positions $\hat{s}_{kt}$ are as follows:

$$\hat{s}_{k,t-1} := s_{k,t-1}$$
$$\hat{s}_{kt} := 0$$
$$\hat{s}_{k,t+1} := \max\{\hat{s}_{kt} + b_{kt} - c, 0\} = b_{kt} - c > 0 \text{ (due to } b_{kt} > c)$$

Therefore, the relation $\hat{s}_{k\tau} \geq s_{k\tau}$ holds for $\tau = t+1, \ldots, T+1$ such that using utility work in cycle $t-1$ instead of t cannot reduce the number of overload situations. The same argument can be made for bringing forward utility work to previous cycle $t-2$, $t-3$ and so on, which completes the proof. □

Theorem 1 states that utility work needs to be performed only in an overloaded cycle, whereas a premature engagement of utility workers to prevent future work overload situations is not preferable. Thus, a simple one-to-one mapping between a work overload situation and an application of a floater exists. This property considerably eases calculating starting times of regular workers and floaters, so that a simple time calculation (similar to the forwards path of project planning (cf., e.g., Krüger and Scholl, 2009)) can be applied.

Evaluation of a given sequence: Let a model sequence be given with b_{kt} denoting the current processing times, then the relevant variables s_{kt} and y_{kt} (1, if a utility worker is applied in cycle t at station k, 0 otherwise) can be computed by the algorithm 1 which is presented in a Pascal-like pseudo-code.

3 Mathematical model

With the assumptions and the optimality condition (Theorem 1) stated above as well as the notation summarized in Table 1 the mixed-model sequencing problem to minimize the number of overload situations

(MM-OS) can be formalized as a binary linear program consisting of objective function (1) and constraints (2) to (11) as follows:

$$\text{(MM-OS) Minimize} \quad Z(X,Y,S) = \sum_{t=1}^{T} \sum_{k \in K} y_{kt} \quad \text{subject to} \tag{1}$$

$$\sum_{m \in M} x_{mt} = 1 \qquad \forall t = 1,...,T \tag{2}$$

$$\sum_{t=1}^{T} x_{mt} = d_m \qquad \forall m = 1,...,M \tag{3}$$

$$b_{kt} = \sum_{m \in M} p_{mk} \cdot x_{mt} \qquad \forall t = 1,...,T; \quad k \in K \tag{4}$$

$$s_{kt} + b_{kt} - l_k \cdot y_{kt} \leq l_k \qquad \forall t = 1,...,T; \quad k \in K \tag{5}$$

$$s_{k,t+1} \geq s_{kt} + b_{kt} - l_k \cdot y_{kt} - c \qquad \forall t = 1,...,T; \quad k \in K \tag{6}$$

$$s_{kt} \geq 0 \qquad \forall t = 2,...,T; \quad k \in K \tag{7}$$

$$s_{k1} = 0 \qquad \forall k \in K \tag{8}$$

$$s_{k,T+1} = 0 \qquad \forall k \in K \tag{9}$$

$$x_{mt} \in \{0,1\} \qquad \forall t = 1,...,t; \quad m \in M \tag{10}$$

$$y_{kt} \in \{0,1\} \qquad \forall t = 1,...,T; \quad k \in K \tag{11}$$

The objective function (1) minimizes the number of overload situations, where utility workers are required, over all production cycles t and stations k. Constraints (2) ensure the assignment of exactly one model to every cycle, while (3) enforces the production of the demanded number of units for any model during the planning horizon. Occurrences of overload situations (marked by $y_{kt} = 1$) are computed in constraints (5). Here, a utility worker is to be applied, whenever the remaining space (and working time) is not sufficient to produce the current workpiece, whose processing time is computed within equations (4). Available working time per station and cycle is calculated with regard to the sequence-dependent starting position s_{kt} for the present workpiece as well as the given station borders, which must not be crossed. Starting positions s_{kt} are calculated in constraints (6) and (7), which represent two different cases limiting the start of work. The limitations of the left station borders are enforced by constraints (7), while constraints (6) model the dependence of the regular operator's starting position on finishing the previous workpiece and a possible application of a utility worker. If a utility worker is active in station k in cycle t, i.e., $y_{kt} = 1$, subtracting the term $l_k \cdot y_{kt}$ makes (6) redundant which reflects that the regular worker can start with his next workpiece in cycle $t+1$ at the left-hand station border. The initial states of starting positions are defined in (8). Constraints (9) enforce the regeneration after completing the sequence, which means that all operators are able to start processing directly at the left-hand station border in a subsequent planning run.

Theorem 2: MM-OS is NP-hard in the strong sense.

Proof: See Appendix.

4 Solution procedures

Since problem MM-OS is an NP-hard optimization problem, instances of real-world size cannot be solved to optimality in a reasonable amount of computational time. However, it is useful to have on hand an exact solution procedure which is able to solve small- and medium-sized instances to optimality, so that benchmark solutions can be gained. Thus, on the one hand, we present a branch-and-bound procedure, which is based on a useful lower bound argument. On the other hand, two heuristic procedures are defined – a fast greedy algorithm and a tabu search procedure. In order to explain the procedures in a comprehensive manner, an example instance with $|K| = 3$ stations, $|M| = 3$ different models, $T = d_1 + d_2 + d_3 = 5$ cycles, and the following parameters is defined as follows:

$$(p_{mk}) = \begin{pmatrix} 105 & 90 & 108 \\ 92 & 110 & 90 \\ 74 & 91 & 110 \end{pmatrix},\ (d_m) = \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix},\ (l_k) = \begin{pmatrix} 110 \\ 110 \\ 110 \end{pmatrix},\ c = 90$$

4.1 Lower bound

In order to restrict enumeration and to evaluate heuristic solution procedures, it is useful to compute a lower bound on the objective function value, i.e., on the number of overload situations. This bound argument relies on a comparison of required assembly time and available (time) capacity at each station. Consider a certain station $k \in K$ and define:

$rt_k = \sum_{m \in M} d_m \cdot p_{mk}$	*required* total time to produce d_m units of all models $m \in M$ at station k
$at = T \cdot c$	total time *available* at each station k during the complete planning horizon T

Obviously, if $rt_k > at$, then it is not possible to produce model units by only applying regular work at station k. At least the *excess time* $et_k = \max\{0, rt_k - at\}$ must be compensated by utility work. Taking into account the logic of the skip policy, it becomes clear that each overload situation has the following characteristics as already utilized in the calculation scheme of Section 2:

- The utility worker takes over complete work on the workpiece in overloaded cycle t. He/She starts at the left-hand station border and is able to finish work as $p_{mk} \leq l_k$ holds.
- As the regular worker needs not be active in cycle t and his/her work of cycle $t-1$ is finished not later than by the end of cycle t due to $l_k \leq 2 \cdot c$, he/she is able to start work at the workpiece of cycle $t+1$ as early as possible at the left-hand station border. Thus, he/she is also able to finish work on this workpiece even if its time b_{kt} is equal to the station length l_k.

As a consequence, the available production (time) capacity is increased by (at most) $it_k = 2 \cdot (l_k - c)$ in each overload situation at a station k as in both cycles $t-1$ and t capacity is increased from c to (at most) l_k. Based on this upper bound on the capacity increase, we can compute a lower bound LB on the number of overload situations necessary to compensate the excess times of the stations as follows:

$$LB := \sum_{k \in K} \left\lceil \frac{et_k}{it_k} \right\rceil \quad \text{with} \quad et_k = \max\{0, \textstyle\sum_{m \in M} d_m \cdot p_{mk} - T \cdot c\} \text{ and } it_k = 2 \cdot (l_k - c) \tag{12}$$

Dividing the excess time by the capacity increase (and rounding the value up, if necessary) provides the minimum number of overload situations to achieve sufficient capacity at station k. Summing up these lower bounds over all stations, provides a lower bound on the total number of overload situations.

In our *example*, we get the available regular capacity $at = 5 \cdot 90 = 450$ TU for each station. As required capacity, we compute $rt_1 = 2 \cdot 105 + 1 \cdot 92 + 2 \cdot 74 = 450$ TU for station 1. Due to $rt_1 = at$ and, thus, $et_1 = 0$, total regular capacity of station 1 is sufficient and no overload situation is mandatory (but could arise in an optimal solution, nonetheless). With $rt_2 = 2 \cdot 90 + 1 \cdot 110 + 2 \cdot 91 = 472$ TU, we get excess capacity $et_2 = 472 - 450 = 22$ TU for station 2. With maximum increase $it_2 = 2 \cdot (110 - 90) = 40$ TU, at least $\lceil \frac{22}{40} \rceil = 1$ overload situations are necessary for station 2. For station 3, we compute $et_3 = 2 \cdot 108 + 1 \cdot 90 + 2 \cdot 110 - 450 = 76$ TU and $it_3 = 2 \cdot (110 - 90) = 40$ TU such that at least $\lceil \frac{76}{40} \rceil = 2$ overload situations are necessary. This leads to the overall bound value $LB = 0 + 1 + 2 = 3$.

4.2 Branch-and-bound procedure

We develop a branch-and-bound procedure for exactly solving MM-OS which is based on a sequence-oriented branching scheme and the lower bound presented in Section 4.1.

4.2.1 Branching scheme

Branching, i.e., constructing the enumeration tree, is organized as follows: The enumeration tree consists of levels $0, 1, \ldots, T-1$. In any node (subproblem) at level t of the tree, a copy of a model is already assigned to each of the first t positions of the sequence defining a residual problem for the sequence positions $t+1, \ldots, T$ with remaining demands d'_m of the models $m \in M$ and (earliest) starting positions $s_{k,t+1}$ for all stations k. The source node at level 0 represents the overall problem with $d'_m = d_m$ for all m and $s_{k,1} = 0$ for all k. The (leaf) nodes at level $T-1$ represent a complete sequence as setting a model copy to position $T-1$ also defines the model to be assigned to the last position T. The same is true at a level $t < T-1$ whenever only one particular model m' has positive residual demand $d'_{m'}$ as all remaining sequence positions are to be filled with copies of m'. A node at level t is branched (developed) in up to $|M|$ different subproblems by assigning a copy of every model $m \in M$ with $d'_m > 0$ to position $t+1$ of the sequence.

The tree is expanded following the *depth-first-search strategy* (with complete node development; cf. Scholl, 1999, p. 116): A single branch of the tree is developed until a leaf node is reached. On its way back to the root, the search follows the first possible alternative branch, i.e., each node is completely developed before its ancestor nodes are revisited. In order to follow most-promising branches first, all descending nodes of the current one are generated first and sorted according to non-decreasing (local) lower bound values (cf. Section 4.2.2). At each revisit of a node P_i, the first (highest-priority) entry of its local list of descending nodes is removed from the list and taken as current node. If the local list of a node P_i is empty, the search traces back to the parent node of P_i.

In case of identical local lower bound values, several *tie breaking rules* are applied to decide on the relative order of sibling nodes. These rules are based on comparing the models just assigned to the latest position t (as all sibling nodes are based on an identical subsequence $1, \ldots, t-1$ defined by their parent node):

1. Ties are broken in favor of the model which has the higher total production time, i.e., the model which maximizes $\sum_{k \in K} p_{mk}$.
2. When a second level tie is present, the model with the largest production time for a single station, i.e., the model which maximizes $\max \{p_{mk} \mid k \in K\}$, is preferred.
3. Finally, third-level ties are broken in favor of the model with lower index m.

Whenever a node at level $T-1$ is reached, a feasible solution (complete sequence) is generated. If the objective function value is smaller than the one of the incumbent solution, the (global) upper bound UB will be set to this improved value and the just found sequence will be stored as the new incumbent (upper bound) solution. Initially, the upper bound is set to infinity or derived by some heuristic procedure.

4.2.2 Bounding

Due to the combinatorial nature of MM-OS, the tree generated as described above will have an excessive number of nodes to be enumerated (at least in real-world problem instances). Thus, we try to restrict its size by fathoming nodes based on the lower bound argument presented in Section 4.1. The bound value $LB(P_0) = LB$ computed by applying (12) serves as a *(global) lower bound* of the initial problem (root node) P_0.

Any other node (subproblem) P_i with $i > 0$ at a level t of the tree, originates from extending the partial sequence of the ancestor node with length $t-1$ by assigning a workpiece to position t. Thus, a partial sequence covering positions $1, 2, \ldots, t$ is already fixed and the residual problem is defined by remaining demands d'_m for all models $m \in M$ to be assigned to the remaining $T-t$ positions. To apply the logic of LB to the residual problem, some modifications are necessary: The *remaining required time* is simply calculated as $rt'_k = \sum_{m \in M} d'_m \cdot p_{mk}$. The *remaining available time* becomes station-dependent as earliest starting times of cycle $t+1$ are to be considered, i.e., $at'_k = (T-t) \cdot c - s_{k,t+1}$, because cycle $t+1$ has a reduced length of $c - s_{k,t+1}$. Thus, we get *remaining excess times* $et_k(P_i) = \max \{0, rt'_k - at'_k\} \forall k \in K$. Let $Z(P_i)$ denote the number of overload situations arising in the partial sequence (including position t) which can be calculated

by applying the procedure of Section 2 to the partial sequence. Then, the local lower bound of subproblem P_i can be computed as follows:

$$LB(P_i) := Z(P_i) + \sum_{k \in K} \left\lceil \frac{et_k(P_i)}{it_k} \right\rceil \tag{13}$$

Besides being utilized for sorting subproblems in the branching scheme, local lower bounds allow for *fathoming*: If $LB(P_i) \geq UB$, the optimal solution of node P_i cannot improve on the incumbent solution and P_i is fathomed (deleted from the list of candidate problems).

4.2.3 Dominance rule

We propose a simple dominance rule which may help to fathom nodes even if bounding is not successful. It is based on the following dominance relationship. A partial sequence π_i (defining a node P_i) is said to *dominate* another partial sequence π_j of node P_j with the same length (at the same level) t, if the following conditions hold:

1. Both partial sequences contain the same set of model copies, i.e., $d'_m(P_i) = d'_m(P_j) \forall m \in M$.
2. The starting positions fulfill $s_{k,t+1}(P_i) \leq s_{k,t+1}(P_j) \forall k \in K$.
3. Partial sequence π_i does not cause more overload situations than π_j, i.e., $Z(P_i) \leq Z(P_j)$.

As both partial sequences define the same residual problem with respect to model demands (condition 1) and partial sequence π_i has better (or equivalent) conditions concerning residual available capacity (condition 2) as well as fixed capacity requirements (condition 3), the dominated partial sequence π_j cannot be part of an overall solution that is better than the best overall solution containing the dominating sequence π_i. Thus, subproblems P_j with dominated partial sequences π_j can be fathomed without risk of missing the optimal solution. Note that if both conditions 2 and 3 are fulfilled as equations, there is a two-way dominance relationship. In order not to miss the optimal solution, only one of both subproblems is fathomed (we choose the one which is constructed later).

4.2.4 Example Computation

We illustrate the branch-and-bound procedure by means of our example presented at the beginning of Section 4. Figure 4 repeats problem data and shows the resulting enumeration tree. Subproblems (nodes) P_i are numbered in the order of generating them. As node weights, the local lower bounds $LB(P_i)$ are denoted. Each arc symbolizes a branching operation and the assigned model is given as arc weight. We start without an initial heuristic solution, so $UB = \infty$.

At first, P_0 (with lower bound $LB = LB(P_0) = 3$ as computed in Section 4.1) is subdivided into the three subproblems P_1 to P_3 by fixing model 1, 2, and 3, respectively, to sequence position 1. Applying (13) leads to $LB(P_1) = LB(P_2) = 0+3 = 3$ and $LB(P_3) = 0+4 = 4$. The latter value originates from the fact that station 1 receives a positive excess time, namely, $et_1(P_3) = 2 \cdot 105 + 1 \cdot 92 + 1 \cdot 74 - 4 \cdot 90 + 0 = 16$. Applying the branching rule, the just generated subproblems are ordered into the local candidate list $\langle P_1, P_2, P_3]$, because model 1 has a larger total production time ($105 + 90 + 108 = 303$) compared to model 2 ($92 + 110 + 90 = 292$).

Thus, P_1 is removed from the list and branched into three subproblems P_4 to P_6. Only in P_5 the lower bound of P_1 remains valid, whereas the other two subproblems get increased bound values. In P_4, overload situations take place in current cycle 2 at stations 1 and 3, i.e., $Z(P_4) = 2$. Since the residual problem with $d' = (0,1,2)$ and $s_{k3} = 0$ for all k requires (at least) two further overload situations to match capacity demands in stations 2 and 3, we get $LB(P_4) = 2+2 = 4$. In P_6, an overload situation is to be compensated in cycle 2 of station 3, and the residual problem with $d' = (1,1,1)$ and $s_{k3} = 0$ for all k shows at least one overload situation in each station. Thus, we have $LB(P_6) = 1+3 = 4$. Due to the bound values and the larger total time of model 1, we get the local candidate list $\langle P_5, P_4, P_6]$.

$$(p_{mk}) = \begin{pmatrix} 105 & 90 & 108 \\ 92 & 110 & 90 \\ 74 & 91 & 110 \end{pmatrix},\ (d_m) = \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix},\ (l_k) = \begin{pmatrix} 110 \\ 110 \\ 110 \end{pmatrix},\ c = 90$$

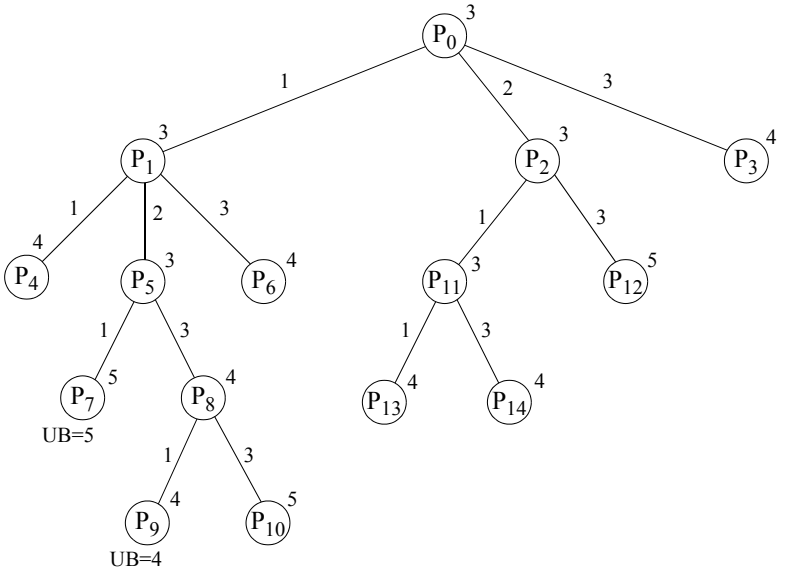

Figure 4: Branch-and-bound tree for the example instance

By branching P_5, we receive subproblems P_7 and P_8. In P_7, the residual problem with $d' = (0,0,2)$ is trivial such that the complete sequence $\langle 1,2,1,3,3]$ is found with objective value $Z(P_7) = 5$. Thus, this sequence is stored as first incumbent solution, UB is set to 5 and P_7 is marked as fathomed. Afterwards, P_8 is branched into P_9 and P_{10} each of which corresponds to a feasible sequence. In P_9, we find the sequence $\langle 1,2,3,1,3]$ with $Z(P_9) = 4$ such that UB is set to 4 and the incumbent solution is updated. Then, P_{10} is fathomed due to $Z(P_{10}) = 5 > UB$ and the search traces back to nodes P_4 and P_6 which are also fathomed due to their bounds being not smaller than UB.

Now, the search proceeds with branching P_2 with $LB(P_2) < UB$. Subproblem P_{11} with bound value 3 is branched whereas P_{12} is fathomed due to $LB(P_{12}) > UB$. The remaining subproblems P_{13} and P_{14} as well as P_3 are fathomed due to their bound values and the search terminates with incumbent solution $\langle 1,2,3,1,3]$ and $Z^* = 4$ overload situations being proven to be optimal.

If we additionally apply the dominance rule of Section 4.2.3, sequence $\pi_{11} = \langle 2,1]$ (subproblem P_{11}) is found to be dominated by $\pi_5 = \langle 1,2]$ (P_5) because both partial sequences contain the same model copies, show the same starting times $s_{\cdot 3} = (17,20,18)$, and do not contain active overload situations. Consequently, nodes P_{13} and P_{14} need not be built anymore.

4.3 Heuristic procedures

Due to the complexity of MM-OS, it is necessary to develop suited heuristic procedures. We propose a fast and easy-to-implement construction heuristic and an exchange-based improvement heuristic controlled by meta-strategy tabu search.

4.3.1 A greedy construction heuristic

In order to compute a first feasible solution for MM-OS quickly, we propose a greedy algorithm based on a time-oriented dynamic priority rule. The sequence is constructed position-by-position starting with an empty sequence. In each iteration $t = 1,\ldots,T$, the model with highest priority value and remaining demand

$d'_m > 0$ is selected and assigned to position t of the sequence. The priority rule is equivalent to the sorting criterion utilized for ordering subproblems in the branch-and-bound procedure: It selects a still required model m which causes the least number of overload situations in cycle t. Ties are broken by the rules 1 to 3 given as described in Section 4.2.1. Nevertheless, the greedy procedure need not necessarily compute the same sequence as the first complete branch of the branch-and-bound procedure, because the local lower bound includes a look-ahead to future assignments in contrast to the myopic approach of the heuristic priority rule.

Example: Applying the greedy heuristic to our example instance works as follows: In the first iteration, of course, no model causes an overload such that model 1 is selected due to its largest total production time. In the second iteration, only model 2 does without an overload and, thus, is assigned to position 2. In the third iteration, model 1 causes an overload in stations 1 and 3, whereas model 3 overloads stations 2 and 3 (no copy of model 2 is left). Due to its larger total time, model 1 is assigned to position 3. Since only model 3 has residual demand, the remaining positions are filled with copies of model 3 thereby inducing three overload situations (two in station 2 and 1 in station 3). To summarize, we receive suboptimal sequence $\langle 1,2,1,3,3]$ with five overload situations.

4.3.2 Tabu Search

To improve on a given feasible solution, we apply an exchange-based improvement heuristic which is guided by a rather simple but effective tabu search approach (see, e.g., Glover and Laguna, 1997). After having tested several versatile neighborhood definitions, which try to find out at which position a model copy should be removed or exchanged to systematically reduce active overload situations, we found out that best results are obtained by a full-exchange neighborhood: As a feasible move, we consider any position exchange of some model copy $m \in M$ with another model copy $m' \in M - \{m\}$. Thus, the neighborhood $N(\pi)$ of the current sequence π consists of all sequences which differ from π in exactly two positions.

In each iteration, all feasible, i.e., tabu-inactive (see below), moves are evaluated by calculating worker movements and by deriving the number of overload situations (both being determined applying the calculation procedure of Section 2). The move which leads to the best objective value is performed and the incumbent solution is updated if an improved solution is identified. Ties are broken randomly, because this way a diversification of the search process is enabled. Clearly, it is not necessary to re-evaluate the entire sequence. In front of the first swap position there is no modification of movements and times so that these overloads need not to be re-evaluated. Furthermore, it can be considered that the starting position of a worker in a station is always reset to the left station border after an overload situation has occurred irrespective of the overload extent. Thus, evaluating the neighborhood can often be done in moderate time although many potential moves are to be evaluated (up to $|T| \cdot |T-1|/2$ neighboring sequences in case of $d_m = 1$ for all m).

In order not to get stuck in a local optimum, a move, i.e., the least increasing one, is also performed if the objective function value does not decrease. As this holds the risk of repeated cycling between a set of already visited solutions, a dynamic version of a tabu list is applied (see, e.g., Glover and Laguna, 1997, chap. 2.5.2) which has been selected after some preliminary tests including different approaches. After having performed any move, it is forbidden for a number of iterations (called tabu tenure TT), to involve the positions just changed in further moves. Initially, TT is set to $\lceil \frac{T}{16} \rceil$. In order to diversify search after a number of unsuccessful moves, the length of the list is increased by 1 every 50,000 iterations after having found the incumbent solution. The increase of TT is controlled such that at least one move remains feasible. Directly after updating the incumbent, TT is reset to its initial value $\lceil \frac{T}{16} \rceil$.

The procedure stops after a pre-specified number of iterations, when a certain time limit is reached or as soon as the current best solution can be proven to be optimal. The latter condition is fulfilled when the objective value of the incumbent solution (upper bound) is equal to the global lower bound LB (cf. Section 4.1).

Example: Let us consider our example instance defined at the beginning of Section 4. We start with randomly generated sequence $\pi = \langle 1,1,2,3,3]$, which shows 5 overload situations and is saved as incumbent. This solution has 8 neighbor solutions to be evaluated: $\langle 2,1,1,3,3]$, $\langle 3,1,2,1,3]$, $\langle 3,1,2,3,1]$, $\langle 1,2,1,3,3]$,

$\langle 1,3,2,1,3]$, $\langle 1,3,2,3,1]$, $\langle 1,1,3,2,3]$ as well as $\langle 1,1,3,3,2]$ each of which causing 5 overload situations. Among these neighbors, let $\langle 1,3,2,3,1]$ be randomly chosen as new current solution π. The involved positions 2 and 5 are set tabu for the next iteration ($TT = 1$). Then, only three neighbor solutions exist: $\langle 2,3,1,3,1]$, $\langle 3,3,2,1,1]$, $\langle 1,3,3,2,1]$ with objective values 5, 4, and 4, respectively. Among the latter two improving neighbors, we select $\langle 3,3,2,1,1]$ by chance and store it as incumbent solution. In the next iteration, changes at positions 1 and 4 are forbidden. As neighbors, we get $\langle 3,2,3,1,1]$, $\langle 3,1,2,1,3]$, $\langle 3,3,1,1,2]$ each with 5 overload situations. After having performed a pre-specified number of iterations (or reaching the time limit), the search stops with $\langle 3,3,2,1,1]$ as best solution found. Due to $LB = 3$, it cannot be proven as optimal solution (though it is).

5 Computational experiments

In this section, we report on different computational experiments in order to examine algorithmic performance of our solution procedures presented in Section 4. Furthermore, we compare both policies of organizing utility work (side-by-side vs. skip policy) in different assembly line settings. This way, the resulting overall duration (cost) of utility work can be compared, so that the preferable policy for a specific problem setting can be identified.

5.1 Data sets and computing environment

As an established test bed is not available, we generated a new data set with 1,080 instances. Due to the problem's complexity, the test bed is subdivided into a subset of small instances and another one containing large instances. Each subset is generated within a full factorial design of different parameters relevant to the problem. Table 2 surveys the different problem parameters and their alternative settings. In both cases, three values for the number of models, the number of stations, and the sequence length are combined with four settings concerning station lengths. For each combination, 5 instances are generated such that each subset contains $3 \cdot 3 \cdot 3 \cdot 4 \cdot 5 = 540$ instances.

Symbol	Description	Values	
		Small	Large
$\|M\|$	Number of models	5, 10, 15	20, 25, 30
$\|K\|$	Number of stations	5, 10, 15	20, 25, 30
T	Number of production cycles	15, 20, 25	100, 200, 300
l_k	Station-lengths constant	110, 150	
	Station-lengths random	[85,125], [85,145]	
c	Cycle Time	90	

Table 2: Parameter settings for randomly generated instances

The cycle time is set to $c = 90$ for all instances, which is a typical value (in seconds) in automobile industry. To determine station lengths l_k, we use two different approaches: First, we assume constant lengths for all stations (short stations, compared to the cycle time $c = 90$, with a length of 110 or long stations with a length of 150). Second, we consider heterogenous stations with different lengths that are randomly chosen (following an uniform distribution) from [85,125] in the short-station case and [85,145] in the long-station case, respectively.

Demand values and processing times are randomly determined by applying uniformly distributed pseudo random numbers. The demand values d_m are chosen from interval [0.5·T/|M|, 1.2·T/|M|] with regard to given number T of production cycles such that $\sum_{m \in M} d_m = T$ is fulfilled. To generate processing times, we first define the average processing time $\bar{t}_m$ of model m by randomly choosing from $[0.75 \cdot c, c]$. This average time is taken as a target for the individual times p_{mk} of model m at station k, each of which is randomly drawn from interval $[0.5 \cdot t_m, min\{l_k, 1.5 \cdot t_m\}]$.

All procedures were implemented using programming language Java and experiments were performed on a personal computer having an AMD Sempron CPU with clock-speed 2.0 GHz and 2 GB of memory.

5.2 Comparing different procedures for MM-OS

We compare the procedures described in Section 4 with respect to their ability of finding (near-) optimal solutions for MM-OS. The branch-and-bound procedure (B&B) is applied as described in Section 4.2. As a time limit, we restrict the computation time to 300 seconds per instance. The tabu search procedure (TS for short) gets a time limit of only 60 seconds as this approach is not intended to solve instances to proven optimality and should obtain good-quality solutions in reasonable time. The greedy procedure is utilized to compute the start solutions for TS as well as B&B and as a stand-alone procedure (denoted as Greedy).

Within this setting, optimal solutions could be found and proven for 436 (410 small and 26 larger instances) out of 1,080 instances. Thus, we subdivide the entire data set into two subsets, one of which contains all instances for which optimal solutions are known (OptSet) and another subset, which contains the instances with unknown optimal solution (NOptSet). For evaluating solution procedures, we utilize the following measures if available:

rel.opt	average relative deviation from the optimal objective value (for OptSet) [in %]
rel.best	average relative deviation from the best upper bound found in the test (for NOptSet) [in %]
maxrel.opt	maximal relative deviation from the optimal objective value (for OptSet) [in %]
maxrel.best	maximal relative deviation from the best upper bound found in the test (for NOptSet) [in %]
totrel.opt	relative deviation of total objective value of all instances from total value of optima (for OptSet) [in %]
totrel.best	relative deviation of total objective value of all instances from the sum of the best upper bounds found in the test (for NOptSet) [in %]
#opt	the number of proven optima, i.e., procedure itself proves optimality due to $UB = LB$ (for OptSet)
#best	the number of best solutions found (among the approaches within a test for NOptSet)
av.time	average computation time in seconds

Let $f^h(i)$ be the objective value computed by a certain method h for some instance i and $f^*(i)$ be the optimal value for this very instance, then $r^h(i) := (f^h(i) - f^*(i))/f^*(i)$ defines the relative deviation from optimality that is caused by procedure h for instance i. The average relative deviation (rel.opt) of procedure h is calculated as the mean of the $r^h(i)$ over all instances i considered in an experiment. If the optimal value remains unknown, we compute the average relative deviation from the best solution value found by any procedure (rel.best) to allow for a cross-procedure comparison.

Besides not having optimal objective values for all instances, we have to face another difficulty which is that instances may have an optimal objective value of 0. In such cases, relative deviations per instance cannot be computed. To overcome this problem, we adopt the approach proposed by Scholl et al. (1998) which compares procedures by calculating cumulated values for all instances of a data set before computing relative deviations. That is, we build the sum of $f^h(i)$ over all instances i (the total number of overload situations procedure h causes in all instances i) and relate this value to the sum of $f^*(i)$ (sum of minima over all instances) to define the measure totrel.opt. Analogously, totrel.best is computed by taking the sum of best solution values as a surrogate for total $f^*(i)$. Consequently, we further subdivide OptSet into OptSet0 (213 instances with optimal value 0), and OptSet1 (223 instances with optimal values > 0). The results are listed in Table 3.

Clearly, B&B outperforms the other procedures for all three subsets as it solves 424 of 436 instances (97.25%) to optimality and reaches low deviations from the (known) optima in acceptable computational times. As a heuristic, TS achieves reasonable results in less time but shows larger deviations of about 14%. The results of Greedy are considerably worse but come at negligible computational time. However, one should not overrate these relatively large relative deviations as it has to be considered that absolute objective values are rather small (near to zero) in many instances of MM-OS. Thus, relative deviations are high even in cases where absolute deviations are small. For example, if the minimum number of overload situations is 1 and a procedure determines a solution with value 2, the relative deviation amounts to 100%.

	OptSet0 (213 inst.)			OptSet1 (223 inst.)			OptSet (436 inst.)		
	Greedy	TS	B&B	Greedy	TS	B&B	Greedy	TS	B&B
rel.opt	-	-	-	102.07	13.85	0.45	-	-	-
max.rel.opt	-	-	-	600	300	100	-	-	-
totrel.opt	-	-	-	79.12	10.22	0.14	113.14	14.16	4.82
#opt	118	189	202	41	169	222	159	358	424
av.time	0.00	31.43	80.12	0.00	44.25	65.34	0.00	37.99	72.56

Table 3: Results for instances solved to optimality

Table 4 summarizes the results obtained for NOptSet (644 instances 514 of which are large). Since lower bound values are 0 or very small for most instances, we do without measuring deviations from lower bounds as these will get values of several hundred percent even for small absolute deviations. Instead, we compare the procedures directly by computing deviations from the best upper bounds determined by one of the procedures in the experiment. Additionally, we summarize the (ascertainable) results for all 1,080 instances of the entire data set.

	NOptSet (644 inst.)			Entire data set (1080 inst.)		
	Greedy	TS	B&B	Greedy	TS	B&B
rel.best	174.04	10.36	53.10	-	-	-
max.rel.best	3,900	200	2,500	-	-	-
totrel.best	83.65	5.40	27.10	85.26	5.88	25.88
#best	42	515	298	201	873	722
av.time	0.00	56.85	266.78	0.00	49.23	188.37

Table 4: Results for instances with still unknown optima and for entire data set

This comparison reveals that TS is the most robust procedure in the test as it achieves total deviations from the best solutions of about 5% (and best solutions in about 80% of the instances) whereas B&B causes deviations of about 26% not to mention Greedy with more than 80%. B&B frequently fails in determining near-optimal solutions within the given time span with the worst case deviations being 2,500%. Clearly, the timeout is reached to early for computing reasonable solutions in these cases. Greedy is not a recommendable stand-alone procedure but suitable as a start point for TS and B&B.

5.3 Side-by-side policy versus skip policy

Up to now, we have introduced and evaluated a new model for mixed-model sequencing being based on the skip policy, i.e., an overloaded cycle is completely taken over by a utility worker. Traditional approaches are based on the side-by-side policy, where a floater accelerates work by working in parallel with the regular worker. Besides the fact that the latter policy is impractical and even unrealistic in many cases (see Section 2), it is an open question which approach is more efficient from an economic point of view. As a precise cost function incorporating any cost type possibly influenced by utility work (e.g. via quality defects) is difficult to derive, we compare both policies with regard to the total time of utility work. Assuming that utility workers get fixed wage rates (which are typically greater than those of regular work) allows for calculating an acceptable estimate for total utility cost.

When measuring the total time of utility work, traditional mixed-model sequencing approaches representing the side-by-side policy (abbreviated by MM-WO from now on, see, e.g., Scholl et al., 1998; Boysen et al., 2009b) do only account for the pure operation times. However, a utility worker will always have some setup effort to disrupt his regular work (e.g., as a team leader), to walk to the station where support is required and to get informed about the tasks to be done. Thus, if a very small amount of utility work, say, some seconds, is considered in the objective function of MM-WO (minimize total work overload) this engagement of a utility worker will usually require much more time.

In order to evaluate and compare solutions in a realistic manner, we incorporate a fixed setup time st for each deployment of a utility worker. By doing so, we can compute the total time effort of utility work by multiplying the number of overload situations (NOS) with st and adding the sum of pure operation times of utility workers (TOT) at the line. Thus, the overall cost function of utility work reads as $NOS \cdot st + TOT$ provided that the wage rate is normalized to 1. This allows for comparing solutions computed by MM-OS and MM-WO on a fair and realistic basis. Clearly, depending on the value of st, the number of overload situations (minimized by MM-OS) or the total utility operation times (minimized by MM-WO) are more or less important.

To compare both approaches adequately, we utilize the same data set as described in Section 5.1. Both approaches differ only with respect to their objective function and the way of calculating times. MM-OS is solved by the proposed B&B procedure (see Section 4.2) and MM-WO by a similar procedure described in Scholl et al. (1998). By doing so, we have very similar conditions for both approaches. The sequences computed by any approach are evaluated by counting the number of overload situations (NOS) each causing a setup of duration st and adding all operation times of utility workers (TOT) as defined by the overall cost function above. The values of TOT are quite different in both approaches. While MM-WO only considers the net excess times which cannot be provided by regular work, the operation times of utility workers are constituted by the processing times of models assigned to an overloaded cycle in MM-OS. This difference is visualized in Figures 1 to 3 of Section 2. For MM-WO (side-by-side policy), the given sequence causes $NOS = 2$ overload situations and utility operation times of $TOT = 3$ TU, i.e., a total cost of $2 \cdot st + 3$. By the way of contrast, the same sequence shows a single overload and a utility operation time of 12 TU in MM-OS (skip policy), i.e., a total cost of $1 \cdot st + 12$. By comparing these total cost values, we can conclude that the side-by-side policy is more efficient if $st < 9$ holds (and the skip policy if $st > 9$).

On this basis, we can deduce which approach deals with overload situations more economically, i.e., by testing which policy is superior for certain values of setup time st. This allows for recommending the best-suited policy in a particular real-world setting. Figure 6 compares both approaches based on the entire data set. The setup time parameter st is varied between 0 (no setup necessary) and $c = 90$ (setup takes a complete cycle). As the measure for comparison, we take the total cost of utility work averaged over all instances of the respective data set.

	side-by-side	skip
NOS	18.77	8.92
TOT	8.72	106.61

Figure 5: Average results for different policies

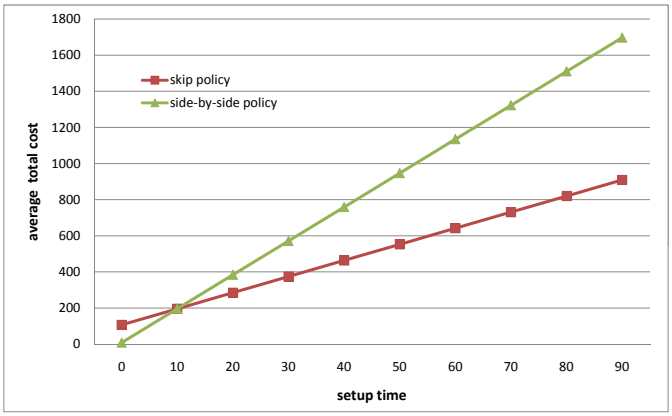

Figure 6: Total cost of different overload policies

Figure 5 shows the apparent trade-off between number of overload situations NOS and their total length TOT. While the skip policy requires only about half of the number of utility worker deployments on average, its mean total deployment time is twelve times higher than caused by the side-by-side policy. The large difference in total time is due to the fact that about 60% of the utility worker deployments caused by the side-by-side policy last less than $0.1 \cdot c$ (about 9 or 10 seconds) while most deployment times are larger than the cycle time in case of the skip policy.

In Figure 6, we display the average total cost functions for both policies. It reads as $8.92 \cdot st + 106.61$ for the skip policy and $18.77 \cdot st + 8.72$ for the side-by-side policy. Thus, $st = 9.94$ (11% of the cycle time

$c = 90$) is the "break-even point" where both policies cause the same total cost (time). That means, if (average) setup time consumes more than 11% of the cycle time (9.94 seconds), one should prefer the skip policy, if not, the side-by-side policy will be more efficient. In automobile industry, we typically find setup times (interruption of regular work, walking, getting to know which tasks are to be done) which consume about 20 to 40% of the cycle time. In such cases, the skip policy is to be preferred. If we take into account that parallel working of regular and utility worker will usually require additional setup as they have to communicate about what has to be done by whom, this advantage of the skip policy further increases.

Similar results are obtained when only taking into account instances which are solved to optimality by *both* approaches. This subset contains 344 (out of 1,080) instances, mostly small instances. In these small instances, the average number of overload situations is much lower than in the entire data set such that the influence of the setup time declines. As a consequence, the break-even point now amounts to about 30% of the cycle time. The other way round, we can conclude that the superiority of the skip policy increases for larger (more realistic) problem instances.

Another possible measure for comparing both approaches is the number of utility workers required (during a shift). This number is determined by the maximum number of simultaneous overload situations as a single utility worker can only handle one overload at a time. On average of the entire data set, the side-by-side policy requires 3.13 utility workers, whereas the skip policy gets by with 2.15 additional workers. This result is due to the larger number of deployments caused by the side-by-side policy. Typically, there are several (short) deployments at different stations at the same time, whereas the skip policy concentrates on fewer (longer) deployments such that the staff of utility workers can be reduced. In the real-world, this is a great advantage as the skip policy contributes to reducing long-time labor costs.

To summarize, the skip policy can be identified the better strategy to cope with overload situations in a majority of real-world mixed-model assembly line settings. Although at a first glance, the side-by-side policy seems superior due to its much shorter total operation time (of actual utility work). But, as supported by our experiments, the opposite is true. Longer deployments force the sequencing model MM-OS to build subsequences with high capacity requirements for which a utility worker deployment really pays off, whereas MM-WO does not account for the setup effort connected with each deployment event.

6 Conclusions and future research

In this paper, we establish a new approach for compensating work overload in mixed-model assembly lines by utility work. The so-called skip policy is often applied in real-world assembly systems but so far no decision support for computing optimal or near-optimal sequences existed. Thus, we describe the modified problem, formalize it as a mathematical program, analyze it thoroughly including an NP-hardness proof and develop suited solution procedures. Computational experiments show that small- and medium-sized problem instances can be solved by a branch-and-bound procedure, whereas larger instances require a heuristic tabu search approach. Comparing the skip policy to the traditional side-by-side policy, we find the new approach being superior concerning times and cost if considerably setup times for utility workers exist.

Further research could extend the new problem in different directions, e.g., by considering other line settings with parallel stations or by directly considering cost effects of utility work. Furthermore, additional solution procedures improving the algorithmic performance of our approaches would be a valuable contribution.

Appendix

We will now prove Theorem 2, i.e., NP-hardness for MM-OS, by a transformation from the 3-Partition Problem, which is well known to be NP-hard in the strong sense (cf. Garey and Johnson, 1979).

3-Partition Problem: Given $3q$ positive integers a_m ($m = 1,\ldots,3q$) and a positive integer B with $B/4 < a_m < B/2\ \forall m \in M$ and $\sum_{m=1}^{3q} a_m = q \cdot B$, does there exist a partition of the set $\{1,2,\ldots,3q\}$ into q sets $\{A_1,A_2,\ldots,A_q\}$ such that $\sum_{m\in A_j} a_m = B\ \forall\, j = 1,\ldots,q$?

Transformation of 3-Partition into MM-OS: Consider $3q+q$ models which are to be assembled on an assembly line consisting of only one station: $|K| = 1$. The further characteristics of the line are as follows: cycle time amounts to $c = B$ and the length of the station is $l_1 = c + B = 2c$. The $3q+q$ models are subdivided into $3q$ models of type 1, which build the counterparts to integer values a_m of 3-Partition. Processing times of type 1 models are to be fixed as follows: $p_{m1} = c + a_m\ \forall m = 1,\ldots,3q$. Furthermore, consider processing times of the remaining q models of type 2 as follows: $p_{m1} = 0\ \forall m = 3q+1,\ldots,3q+q$. Thus, models of type 2 have no processing time and are introduced to reset a sequence, so that work may continue at the left-hand border in the next production cycle. We call each partial sequence of three type 1 models with a concluding type 2 model a "reset cycle". The question we ask is whether we can find a solution for MM-OS without utility work, so that objective value amounts to $Z = 0$.

On the one hand, a feasible solution for an instance of 3-Partition can be directly transformed into a feasible solution of the corresponding MM-OS-instance. For each set A_j we just schedule the corresponding type 1 models (in facultative order) at the first three production cycles of a reset cycle and additionally schedule a type 2 model into the last position of the reset cycle. As the integer values of each set amount to B, obviously in any reset cycle the right-hand border of the station will exactly be reached just before the final type 2 model is processed. Then, this model exactly allows for a reset to the startup position at the left-hand border of the station. Thus, no work overload can occur when we concatenate all reset cycles and $Z = 0$ holds for MM-OS.

On the other hand, we can also prove that each YES-instance of MM-OS is also a feasible solution for 3-Partition: First, any reset cycle containing more than three type 1 models must result in work overload at the station due to the restriction on the processing time values $a_m > B/4 = c/4$. Furthermore, if there exists a reset cycle R_A with less than three type 1 models there must exists another reset cycle R_B having more than three of these models, which would inevitably cause work overload in reset cycle R_B. Thus, any feasible solution for MM-OS must contain exactly three type 1 models per reset cycle.

Analogously, it can be shown that a reset cycle (with three type 1 models), which does not exactly reach its right station border just before the reset model of type 2 approaches, must cause a work overload in another reset cycle. Due to the fact that stations are closed and, thus, idle time at the left hand border cannot be utilized to start working at the next workpiece earlier in combination with the condition $\sum_{m=1}^{3q} a_m = q \cdot B$, idle time in one (reset) cycle enforces work overload in others.

Consequently, in a YES-instance of MM-OS processing any three type 1 models of a reset cycle must exactly reach the right-hand border at station 1. It directly follows that a YES-instance of MM-OS without work overload ($Z = 0$) does exist if and only if the answer to the corresponding instance of 3-Partition is YES. NP-hardness in the strong sense for MM-OS immediately follows. □

References

Becker, C., Scholl, A., 2009. Balancing assembly lines with variable parallel workplaces: Problem definition and effective solution procedure. European Journal of Operational Research 199, 359–374.

Bolat, A., 1997. Stochastic procedures for scheduling minimum job sets on mixed model assembly lines. Journal of the Operational Research Society 48, 490–501.

Bolat, A., Yano, C., 1992a. Scheduling algorithms to minimize utility work at a single station on a paced assembly line. Production Planning and Control 3, 393–405.

Bolat, A., Yano, C., 1992b. A surrogate objective for utility work in paced assembly lines. Production Planning and Control 3 (4), 406–412.

Boysen, N., Fliedner, M., Scholl, A., 2009a. Assembly line balancing: Joint precedence graphs under high product variety. IIE Transactions 41, 183–193.

Boysen, N., Fliedner, M., Scholl, A., 2009b. Sequencing mixed-model assembly lines: Survey, classification and model critique. European Journal of Operational Research 192 (2), 349–373.

Celano, G., Costa, A., Fichera, S., Perrone, G., 2004. Human factor policy testing in the sequencing of manual mixed model assembly lines. Computers & Operations Research 31 (1), 39–59.

Garey, M., Johnson, D., 1979. Computers and intractability: A guide to the theory of np-completeness. Freeman, San Francisco, CA.

Glover, F., Laguna, M., 1997. Tabu search. Kluwer Academic Publishers, Norwell, MA.

Krüger, D., Scholl, A., 2009. A heuristic solution framework for the resource constrained (multi-)project scheduling problem with sequence-dependent transfer times. European Journal of Operational Research 197 (2), 492–508.

Scholl, A., 1999. Balancing and sequencing of assembly lines. 2nd ed., Physica, Heidelberg.

Scholl, A., Klein, R., Domschke, W., 1998. Pattern based vocabulary building for effectively sequencing mixed model assembly lines. Journal of Heuristics 4, 359–381.

Sumichrast, R., Oxenrider, K., Clayton, E., 2000. An evolutionary algorithm for sequencing production on a paced assembly line. Decision Sciences 31 (1), 149–172.

Xiaobo, Z., Ohno, K., 1994. A sequencing problem for a mixed-model assembly line in a jit production system. Computers & Industrial Engineering 27 (1-4), 71–74.

Xiaobo, Z., Ohno, K., 1997. Algorithms for sequencing mixed models on an assembly line in a jit production system. Computers & Industrial Engineering 32, 47–56.

Xiaobo, Z., Ohno, K., 2000. Properties of a sequencing problem for a mixed model assembly line with conveyor stoppages. European Journal of Operational Research 124 (3), 560–570.

Yano, C., Bolat, A., 1989. Survey, development, and application of algorithms for sequencing paced assembly lines. Journal of Manufacturing and Operations Management 2, 172–198.

Yano, C., Rachamadugu, R., 1991. Sequencing to minimize work overload in assembly lines with product options. Management Science 37, 572–586.

Yoo, J., Shimizu, Y., Hino, R., 2005. A sequencing problem for mixed-model assembly line with the aid of relief-man. JSME International Journal. Series C, Mechanical Systems, Machine Elements and Manufacturing 48 (1), 15–20.