An exact algorithm for the mixed-model level scheduling problem

Abstract

The Monden Problem, also known as the Output Rate Variation Problem (ORV), is one of the original formulations for mixed-model assembly line level scheduling problems in a just-in-time (JIT) system. In this paper, we develop a new branch-and-bound procedure as an efficient solution of this problem that uses several new and previously proposed lower and upper bounds. The algorithm also includes several dominance rules that leverage the symmetry in the problem as well as a new labelling procedure that avoids repeated exploration of previously examined partial solutions. The branching strategy exploits the capabilities of current multi-processor computers by exploring the search tree in a parallel fashion. The proposed algorithm has been tested on several sets of instances from the literature and is able to optimally solve problems that are double the size of those addressed by other procedures previously reported in the literature for the ORV problem.

Keywords. Mixed-model level scheduling, production, manufacturing, JIT.

1 Introduction

Highly homogeneous mass-marketed products are commonly manufactured on assembly lines. Despite this uniformity, certain products exist (e.g., cars) for which the customer is able to choose among a great variety of options (also referred to as parts, e.g., air conditioning, three or five doors, etc.), and each different configuration of the possible options constitutes a model. If the differences among these models are sufficiently small that the preparation time required between models does not exist or can be neglected, then the different models are likely to be produced on the same assembly line in a mixed sequence. These lines are known as mixed-model lines and present a sequencing problem because the sequence in which the models are processed directly affects the station workload, the production rate of each model, the usage rate for each option, and the intermediate and safety stocks required, among others.

According to the survey and classification by Boysen et al. (2007), mixed-model assembly lines sequencing problems may be classified into three groups depending on their objective:

- Mixed-model sequencing: The main aim of these problems is to avoid sequence-dependent work overloads. This process includes a wide variety of goals, such as minimising the size of the line, minimising utility work (Boysen et al., 2011), minimising space constraint violations (Yano and Bolat, 1989) or minimising the total amount of incomplete work (Bautista and Cano, 2008; Yano and Rachamadugu, 1991).
- Car sequencing: Car sequencing problems address feasibility issues. In this problem, a sequence must fulfil given restrictions on the number of times an option may be used in consecutive positions in the sequence (Cordeau et al., 2008; Fliedner and Boysen, 2010; Gravel et al., 2005; Prandtstetter and Raidl, 2008; Solnon, 2008; Solnon et al., 2008).
- Level scheduling: The problems in this group include just-in-time goals such as minimising setup times, reducing storage costs, minimising intermediate buffers, and evenly distributing the option requirements. An example of a problem in this category would be an attempt to level the production rates of each model along the sequence. This problem is known as the PRV (Product Rate Variation) and can include other secondary goals, such as minimising the difference between the production rates and the demand rates (Drexl and Kimms, 2001; Kubiak and Sethi, 1991; Miltenburg, 1989; Tavakkoli-Moghaddam and Rahimi-Vahed, 2006; Zhu and

Ding, 2000; Yavuz, 2013). According to Boysen et al. (2009), the PRV model does not accurately represent the objectives of level scheduling because it does not consider smoothness in the usage of parts or options.

This work studies a level scheduling problem in which the objective is to level the usage rates of the options present in each model, while indirectly minimising the work overloads. The problem was first proposed by Monden in his study of the Toyota Production system (Monden, 1983), and it is known as the Monden problem (Bautista et al, 1996), or the ORV (Output Rate Variation) problem (Kubiak, 1993). More specifically, the paper describes an exact algorithm based on the branch-and-bound paradigm for solving the ORV problem. To this end, a new set of lower bounds based on the separability of the problem has been developed in addition to several dominance rules that are intended to reduce the size of the exploration tree. In the proposed algorithm, these elements are combined with previously published heuristics (upper bounds) and lower bounds.

The remainder of the paper is structured as follows. In subsection 1.1, we review the literature pertaining to the ORV problem, and subsection 1.2 presents an integer programming and a dynamic programming (DP) formulation of the problem. In Section 2, we describe the applied upper bounds, which include both a greedy goal-chasing method (Monden, 1983) and a Bounded Dynamic Programming technique (Bautista et al., 1983). Section 3 focuses on the lower bounds used throughout the algorithm. Section 4 is devoted to the branch-and-bound procedure and its description. The ramification strategy is laid out in subsection 4.1, and subsection 4.2 focuses on the bounding techniques. Several dominance rules are introduced in subsection 4.3 together with their use in the procedure. The general structure of the branch-and-bound algorithm is schematised in subsection 4.4. Section 5 presents the instances used to test the quality of the procedure and the results obtained in a computational experiment. Finally, Section 6 gives a summary and conclusions on the work.

1.1 Literature review

The literature pertaining to the ORV has been intensely focused on heuristic approaches. Monden (1983) developed two greedy goal-chasing-methods, the GC-1 and GC-2. Miltenburg (1989) proposed three sequencing procedures that considered each stage separately to construct the new solution (M-A1, M-A3H1, M-A3H2). Sumichrast and Russell (1990) developed a mixed-integer programming approach. In their article, their results were compared with the five cited heuristics. Despite finding the optimal solution for the smallest instances, the procedure was too slow for practical applications.

Bautista et al. (1996) developed a DP solution method that reduced the computing time by limiting the number of states that should be taken into account in each stage. This approach is known as the BDP (Bounded Dynamic Programming), see Subsection 2.2 for a description of the method. This paper also presented a set of lower bounds that can be used to reduce the number of states that must be developed.

Other methods for solving the ORV problem have been proposed, including the scheduling heuristic of Miltenburg and Sinnamon (1989), two competitive Beam Search procedures (Erel et al 2007; Leu et al., 1997) that consider not only the levelling of options usage but also the levelling of workloads, and an improved goal-chasing method, Jin and Wu (2002), which uses a cost function for sequencing a model in each position that takes into account not only the variation in options usage for that position but also for the remaining positions and models. Fliedner et al. (2010) studied the symmetry of the ORV problem along with methods which exploit the structural properties of the ORV instances to strengthen the algorithms applied to them.

To the best of our knowledge, three exact methods exist for solving the problem under consideration in the literature, and all of them are based on DP methods. The first procedure, proposed in Bautista et al. (1996), is able to optimally solve instances with up to 20 units and can be used to solve larger instances heuristically. The second procedure, presented in Miltenburg (2007) uses a similar dynamic programming method and was able to solve instances of varied size depending on the objective under consideration (for the objective studied in this paper, only one instance with 40 units is solved). The third procedure, presented in Fliedner et al. (2010), uses the symmetry of the problem to strengthen the previous exact algorithm and is able to optimally solve instances with up to 30 units.

The literature also covers other level scheduling problems that use similar data instances but have different objectives. In Boysen et al. (2008) the problem presented considers a line supplied by consignment stock and aims to schedule the models so as to minimise the inventory costs of the options. The exact algorithm presented is capable of solving instances of the aforementioned problem with up to 35 units. In Drexl and Kimms (2001), a column generation algorithm is proposed to solve a problem with a PRV objective and car sequencing constraints and is able to optimally solve instances with up to 30 units. The same problem is studied in Drexl, Kimms and Matthiessen

(2006) and Yavuz (2013) in which instances with up to 50 units were solved. In comparison, the procedure presented in this article is able to solve instances with much larger sizes. In certain cases, instances previously used in the literature with up to 200 units have been optimally solved with our method.

In the following subsection, we present the mathematical model for this problem, both as an integer program (IP), using the notation shown in Jin and Wu (2002), and as a DP, using the notation shown in Bautista et al. (1996).

1.2 Problem description and mathematical model

An instance of the ORV problem can be stated as follows: let there be *M* different models with demands $d=[d_1,d_2,...,d_M]$ such that $T=\sum_{i=1}^{M} d_i$ is the total number of units to be manufactured. These models are produced in a single assembly line in which products enter at a constant rate. Consequently, the solution to this problem can be represented by a sequence of *T* positions that univocally associate each time instant with the corresponding model.

Each model may contain up to *P* different options used in the production of these models. The usage of each option by each model is represented using a matrix *C*, in which the element c_{ij} indicates the usage of option *j*, *j*=1,…,*P*, by model *i*, *i* =1,…,*M*. The ideal rate for each option *j*, *j*=1,…,*P*, can be defined as: $r_j=\sum_{i=1}^{M} d_i \cdot c_{ij} \big/ T$. In other words, for any given position *t* in the sequence, the amount of option *j* that would be used if the sequence were perfectly smooth would be given by $r_j \cdot t$. The solution to the problem attempts to find the sequence that minimises the sum for each sequence position of the differences between the defined ideal rates and the actual usage rates for each option.

Let *X* be a set of binary variables, in which each element x_{it} , *i* =1,…,*M* and *t*=1,…,*T*, has a value of 1 if a unit of model *i* has been sequenced in the position *t* of the sequence and a value of 0 otherwise. Next, the real usage for an option *j* and a position *t* can be obtained as follows: $\sum_{i=1}^{M}\sum_{\tau=1}^{t} x_{i\tau} \cdot c_{ij}$, and the global objective function is as indicated in (1).

$$[MIN]\sum_{t=1}^{T}\sum_{j=1}^{P}\left(\sum_{i=1}^{M}\sum_{\tau=1}^{t} x_{i\tau} \cdot c_{ij} - r_j \cdot t\right)^2 \qquad (1)$$

We note that there are several valid methods for expressing the difference so as to minimise it. In this article, we consider the objective function as minimisation of the sum of squared differences, known as SSD (Miltenburg, 2007), as shown in in the seminal work of Monden (1983), as well as in Bautista et al (1996), Erel et al. (2007), Fliedner et al. (2010) and Jin and Wu (2002), among others.

The motivation behind the use of a squared difference function is threefold: (1) the need to penalise both the positive and the negative differences; (2) the desire to penalise large differences over a quantity of small differences; and (3) the need to consider all of the deviations from the ideal rates, rather than the largest deviation, which was the objective considered in Miltenburg (2007).

Using the variables defined above, objective (1) combined with restrictions (2)-(4) defines a valid IP model for the problem.

$$\sum_{t=1}^{T} x_{it} = 1; \forall i = 1,...,M \tag{2}$$

$$\sum_{i=1}^{M} x_{it} = 1; \forall t = 1,...,T \tag{3}$$

$$x_{it} \in \{0,1\}; \forall i = 1,...,M \ \forall t = 1,...,T \tag{4}$$

Constraint set (2) ensures that the demand of every model is met. Constraint set (3) ensures that only one model is scheduled at a given instant. Finally, constraint set (4) makes certain that each variable in set *X* is binary.

An alternative formulation for this problem is based on the DP paradigm (Bellman, 1957). As in every deterministic DP with a finite horizon, the model can be represented as a multistage decision problem. Each state is defined by a vector$[u_1, u_2, ..., u_M]$, where u_i is the number of sequenced units for model *i* up to stage *t* and $0 \leq u_i \leq d_i$, and $\sum_{i=1}^{M} u_i = t$ must hold. There exists an initial state *α* in which no models have been sequenced and a final state *ω* in which every model has been sequenced.

The cost associated with the transitions is given by the recurrence function, equations (5) or (6). Two possible recurrence functions exist, depending on whether we consider a forward or a backward formulation.

Considering a forward formulation, the initial state α is *S*=[0, 0,…, 0] with cost *C*(*α*)=0. The cost of every other state is determined using the recurrence equation (5). We refer to the cost of the last transition used to reach the state as the contribution of a state, and to the total cost to reach the state from α as the cumulative contribution of the state. Stage T has a single state S=[d_1, d_2, …, d_M], and its cost C(ω) gives the cost, SSD, of the optimal sequence.

$$C^t(S) = MIN\left\{C^{t-1}(S \setminus \{i\}) + \sum_{j=1}^{P}\left(\sum_{i=1}^{M} u_i \cdot c_{ij} - r_j \cdot t\right)^2\right\}_{1 \le i \le M \,/\, 0 \le u_i < d_i} \tag{5}$$

For the backward formulation, the initial state α is S=[d_1, d_2, …, d_M] with cost C(α)=0. The cost of every other state is determined using the recurrence equation (6). The last state ω is S=[0, 0,…, 0] and its cost C(ω) gives the cost of the optimal sequence.

$$C^t(S) = MIN\left\{C^{t-1}(S \cup \{i\}) + \sum_{j=1}^{P}\left(\sum_{i=1}^{M} u_i \cdot c_{ij} - r_j \cdot t\right)^2\right\}_{1 \le i \le M \,/\, 0 \le u_i < d_i} \tag{6}$$

The forward formulation will be used to obtain the upper bounds (see subsection 2.2), and the backward formulation will be used for the calculation of lower bounds (see subsections 3.1 and 3.2).

a summary of the notation used throughout the paper is presented in Table 1.

Table 1: Notation used in the paper

Symbol	Description
i	Model identifier
j	Option identifier
t	Instant (time) identifier
C_{it}	Cost of sequencing model *i* at instant *t* in the transportation problem
T	Number of periods and of units to sequence
M	Number of models, $1 \le i \le M$
P	Number of options, $1 \le j \le P$
d_i	Demand for model *i*
c_{ij}	Usage of option *j* by model *i*
r_j	Ideal usage rate of option *j*

2 Upper Bounds

Before beginning the branch-and-bound procedure, three initial solutions are obtained. The calculation of the upper bounds has the objective of increasing the quantity of nodes fathomed during the exact algorithm as well as reducing the memory requirements of the lower bounding techniques. The first solution is given by a goal-chasing method, and the second and the third are given by BDP approaches.

2.1 Goal-chasing

The first upper bound is obtained by means of a greedy heuristic (Monden, 1983), known as goal-chasing. For each given position, the heuristic selects the model with unmet demand that minimises the value for (1) in that instant. This method is myopic because it uses the ‘best’ combinations (those with smaller SSD contributions) early in the sequence but is quite fast in calculations.

2.2. Bounded Dynamic Programming

Solving the ORV problem by means of a DP formulation is usually deemed infeasible for medium and large-sized instances due to the high number of states that must be developed and stored. Alternatively, a Bounded Dynamic Programming approach is proposed (Bautista et al., 1996). This procedure reduces the number of states that must be stored by the following: (1) eliminating the inefficient states and (2) developing a limited number *w* of states of the DP for each stage. Note that the goal-chasing method is equivalent to a BDP in which for each stage only one state is developed (*w*=1).

For this purpose, it is necessary to calculate a lower bound on the SSD for any given state for the following reasons: (1) to fathom the states that cannot lead to a better solution than the best known solution and (2) to rank the states of each stage in best to worst lower bound order to select the *w* most promising states. This lower bound takes into account the SSD of the current state plus a lower bound on the states that are yet to be developed.

Our BDP implementation uses a two-list method (Bautista and Pereira, 2009). In this method, the stages of the DP are consecutively explored, generating every state of the current stage from the states of the previous stage. The exploration is started at stage 0 and is repeated until the final stage has been generated.

Once a stage has been completely explored, the BDP behaviour differs from its DP equivalent in that only a subset of the states is maintained. The number of maintained states is controlled by a parameter *w*, known as the window size. The window size limits the number of states used to develop the states of the following stage such that only the *w* best states (the *w* first states in the ranking) for each stage are maintained. This approach reduces the number of explored states, the computation time and the memory usage. As a consequence, the solution obtained can only be proven as optimal if no state is discarded due to window size limitations, leading to a heuristic method for large-size instances.

The proposed algorithm uses two different BDP versions. The first BDP does not make use of a lower bound to fathom and to rank the states. The second BDP uses the separability lower bounds (see subsection 3.1).

3. Lower Bounds

In the present work, several lower bounds are used in both the exact procedure (branch-and-bound) and in the DP methods. Subsection 3.1 is devoted to the symmetry property of the problem, and subsections 3.2 and 3.3 study the lower bounding methods used by the DP and the branch-and-bound procedures.

3.1 Symmetry-based lower bound

The ORV problem has symmetry properties that have been previously studied in Bautista et al. (1996) and Fliedner et al. (2010). The following can be proved:

- For any state *A* in stage *t*, its complementary state $\overline{A}$ in the stage *T*-*t* exists and can be defined as follows: let *A* be defined by the cumulative production plan $[u_1, u_2, ..., u_M]$; $\overline{A}$ is subsequently defined by the cumulative production plan $[d_1 - u_1, d_2 - u_2, ..., d_M - u_M]$.
- The increase in the SSD between any given state *A* and one of its successors *A'* is identical to that between state $\overline{A}'$ and $\overline{A}$.
- The SSD from the initial state α to any state *A* is identical to the SSD increase from state $\overline{A}$ to the final state ω.

Considering these properties, for any state *A* with a cumulative production plan *u*, the cost of its complementary state $\overline{A}$, C($\overline{A}$), is equal to the cost from *A* to ω. Consequently, C($\overline{A}$)+ C(*A*) corresponds to the SSD of the optimal sequence from α to ω through state *A*.

This fact is exploited by DP procedures (see subsection 3.2) to avoid development of nodes in the graph that cannot lead to better solutions. The symmetry of the problem is also used in the branch-and-bound procedure to find the lower and upper bounds by combining the partial solution under exploration with the previously explored partial solutions (see subsection 4.3.4) and to avoid symmetries in the solution (see subsection 4.4.3).

3.2 Option separability-based lower bound

The ORV problem may be divided into sub-problems containing a subset of options. The optimal solution for any of these sub-problems is always a lower bound to the optimal solution of the original problem. This lower bound is a novel contribution of the present work.

The separability can be deducted from the objective function, and equation (1) can be rewritten as equation (7). Each option or subset of options can be independently solved**,** and the combination of sub-problems with disjoint subsets of options provides a lower bound on the value of the objective function.

$$\sum_{j=1}^{P}\left(\sum_{t=1}^{T}\left(\sum_{i=1}^{M}\sum_{\tau=1}^{t}x_{i\tau}\cdot c_{ij}-r_j\cdot t\right)^2\right) \tag{7}$$

Each sub-problem is subsequently defined by the same set of constraints (2), (3) and (4) from the original problem (see Subsection 1.2).

Erasing certain options from the problem can cause several models to produce identical usage of the options. These models can subsequently be merged into a single model with the same usage as any of the identical models and a demand equal to the sum of demands for all the identical models. It should be mentioned that any state in the DP associated with the original problem corresponds to a single state in the DP for any of the sub-problems. Additionally, the cost of the state in the sub-problem represents a lower bound on the cost of the corresponding state in the original problem.

The merging of models is important because the number of states in the DP ~~Dynamic Program~~ grows exponentially with the number of models in the instance, and these sub-problems will be solved using a DP based procedure. Even if the solution of the ORV problems by means of DP is theoretically difficult, in most cases it is empirically possible to solve and store the solution of sub-problems with up to three options and 200 units within a few seconds using modern commodity computers (see the computational experiment in Section 5 for further details).

Consequently, for problems with up to seven options, all sub-problems with one, two and three options are solved and stored. When determining the lower bound ~~value~~ for a node during the branch-and-bound or the BDP, any combination of sub-problems that combine each option at most once provides a valid lower bound. Hence, we test all of the combinations and select the best among them.

For instances with a high number of options and units, the number of possible combinations can cause memory issues (e.g., insufficient memory to store all of the required states), and thus, for instances with more than 7 options, only the sub-problems with one option are solved, and the lower bound will be obtained by combining each of the sub-problem optimal solutions (summing their lower bound).

The DP formulation used to solve these sub-problems is the backward formulation. The cost in the backward formulation for any state *A* is associated with the cost from the current state *A* to the final state. As such, this value can be used as a lower bound on the cost of the remaining states left to explore (from state *A* to the final state) in a forward formulation by subtracting the contribution of the current state from the total contribution.

To avoid the generation and storage of inefficient states, the method uses an upper bound *UB* equal to the solution obtained by the BDP heuristic. Any state *S* with $C(S)\geq UB$ is not stored because their corresponding states in the original DP will not lead to improved solutions. The UB can be further restricted by subtracting the best known lower bound for the options not considered in the current sub-problem.

Two lower bounds on the states yet to be developed are calculated to further prune the total states: (1) for any state of any stage over $T/2$, the symmetry lower bound (see subsection 3.1) is calculated and used to avoid storage of inefficient states, and (2) for any sub-problem with two or more options, the lower bound is calculated using the optimal solution of the sub-problems with a single option.

3.3 Transportation problem-based lower bound

Considering the mathematical formulation presented in subsection 1.2, the problem can be studied as a transportation problem in which the suppliers are considered as the models, and the customers correspond to the instants of the sequence, see Bautista et al. (1996).

Let there be *M* models with demand $[d_1,d_2,\ldots,d_M]$ and *T* positions in the schedule, each with capacity equal to 1. Each element C_{it} of matrix *C* is a lower bound of sequencing model *i* in the instant *t*. The solution is given by a set of binary variables x_{it} with a value of 1 if the model *i*, $i=1,\ldots,M$, is sequenced in position *t*, $t=1,\ldots,T$, and a value of 0 otherwise.

The problem is subsequently represented by objective (8) with constraint sets (2), (3) and (4).

$$[MIN]\sum_{t=1}^{T}\sum_{m=1}^{M} x_{it}\cdot C_{it} \qquad (8)$$

The method calculates C_{it} as a bound on the combined contributions for two consecutive positions in the sequence. For every instant and model, each coefficient is obtained by calculating the contribution of sequencing a unit of the model during *t* plus the contribution of the previous situation in *t*-1. This contribution is calculated separately for each option, and among all possible situations, those with a minimum cost for each option are selected. The C_{it} is set to half of the summed contribution value because two positions in the sequence are being considered instead of one.

To evaluate the coefficients C_{it}, a new variable Y_j is defined that indicates any possible cumulative usage of any option *j* up to stage *t*. The resulting equation is shown in (9).

$$C_{it} = \sum_{j=1}^{P} MIN\left\{\left(Y_j - c_{ij} - r_j \cdot (t-1)\right)^2 + \left(Y_j - r_j \cdot t\right)^2\right\}_{\forall Y_j / c_{ij} \le Y \le \sum_{\forall i} c_{ij} \cdot d_i} \Big/ 2 \qquad (9)$$

The thresholds for the minimum and maximum usage of options in equation (9) are too wide, considering that the only value needed is the minimum. Tighter minimum and maximum values of Y_j can be obtained using (10) and (11), respectively (see the demonstration in Appendix A).

$$\lfloor r_j \cdot t + (c_{ij} - r_j)/2 \rfloor \qquad (10)$$

$$\lceil r_j \cdot t + (c_{ij} - r_j)/2 \rceil \qquad (11)$$

Once the values for the components C_{it} have been obtained, solving the transportation problem yields a lower bound for the ORV problem. While the use of a transportation problem to derive bounds was documented in Bautista et al. (1996), this work is the first one that documents its use in any enumeration procedure. Furthermore the method proposed to obtain the costs is also a novel contribution of this work.

4 Branch-and-bound algorithm

The branch-and-bound procedure used for solving the ORV problem is presented in this section. The proposed branch-and-bound algorithm explores a tree of partial solutions, each represented by a node in the search tree. The method begins with a root node representing a partial solution in which no decision has been made. Each step of the algorithm selects a node for exploration, known as the parent node. The exploration creates descendant nodes and generates several new nodes or a final solution by fixing a model to be scheduled at the first available position of the partial solution. The process is repeated until no node is left unexplored. Every node has a parent, except for the root node, and every node has one or more descendants, except for the terminal nodes. Terminal nodes are those corresponding to the final stage *T* and thus represent a solution, or those nodes for which all the descendants have been fathomed.

The branch-and-bound approach is an alternative to the DP formulation, but a relationship exists between the nodes of the branch-and-bound tree and the states of the DP because there exists a state $[u_1,u_2,\ldots,u_M]$ associated to each ~~node~~ partial solution in the branch-and-bound algorithm, but there may exist more than one partial solution for each state. Despite the need to explore the same state several times because a single state corresponds to different partial solutions, the main advantage of a branch-and-bound procedure over a DP approach is that it does not require exponential memory. However, the relationship between the DP and the enumeration procedure in the branch-and-bound can be used to eliminate certain partial solutions of the latter.

The remainder of the section is structured as follows. First, the method of exploration for the search tree is discussed in subsection 4.1, including the use of parallel computation. Subsection 4.2 is devoted to the use of the lower bounds proposed in Section 3 within the branch-and-bound algorithm. Subsection 4.3 addresses the different dominance rules used to detect dominated or identical nodes in the search tree, and finally, subsection 4.4 discusses the general structure of the implemented procedure.

4.1 Ramification strategy

The ramification strategy for this procedure uses multiple and limited depth-first searches, starting from nodes selected by a best-first criterion. The objective of this method is to use all available processors in modern commodity computers to concurrently explore different areas of the search space. To that end, the number of nodes explored in each depth-first search is limited by a parameter of the algorithm. When the limit is reached, the depth-first search stops, and all generated but unexplored descendants are stored in a memory pool. Whenever a process is idle, the method chooses a node from the memory pool to begin a depth-first search using a best-first policy. If the parameter is set to a large value or set to infinity, the resulting algorithm is similar to independent depth-first searches whereas if the parameter is set to 1, the resulting algorithm will behave in a manner similar to a best-first search. Note that this branching strategy ressembles the “Scattered Branch & Bound” method proposed in Fliedner and Boysen (2008) for the car sequencing problem.

Additionally, and to avoid running out of memory due to the storage of an excessive number of nodes in the pool of generated but unexplored descendants, whenever their number reaches a certain quantity (a parameter based on the available computer memory), the starting nodes are chosen using a depth-first strategy instead of a best-first one.

4.2 Bounding techniques

The proposed method uses the two lower bounds described in Subsection 3.2 and Subsection 3.3.

When creating a new partial solution, i.e., a node in which the number of developed stages is less than the total number of stages, a lower bound for the stages yet to be sequenced is calculated. This lower bound can be obtained as the best combination of the lower bounds belonging to the sub-problems with separate options or by means of the transportation-based lower bound.

The transportation problem is solved using a negative cycle cancelling algorithm (Goldberg and Tarjan, 1989; Ahuja, Magnanti and Orlin, 1993) for the min-cost max flow problem. The procedure starts from an initial solution (derived from the optimal solution to the transportation problem of the parent of the current node) and repeatedly improves its objective value. We opted for the use of a cycle cancelling algorithm after observing that few iterations were required to reach optimality if a near optimal initial solution was provided

Because the search for a lower bound value among the states stored for several sub-problems with separated options requires less computational effort than solving the transportation problem, the procedure first calculates the lower bound by option separability, and the transportation problem is only solved if the first lower bound did not fathom the node.

The dominance rules (see the following subsection) are only examined for the partial solutions that are not fathomed in this step.

4.3 Dominance rules

To detect dominated partial solutions that cannot lead to a better solution, several dominance rules are applied. These rules are based on two principles. The first is that several nodes in the branch-and-bound tree correspond to the same state in the DP, and these equivalent solutions do not need to be explored more than once. The second principle is related to the symmetry of the problem.

Two approaches are considered to avoid the development of several equivalent solutions: (1) active creation of new sequences that correspond to the same state in a DP by means of a greedy algorithm and (2) storage of the explored states together with their associated SSD value.

The dominance rules related to the symmetry of the problem are based on (1) not exploring a sequence that is symmetric to a previously explored one and (2) not developing a complete sequence if the complementary sequence has already been explored. Each of the four rules is examined in a separate subsection. Note that the four dominance rules are novel contributions of this work.

4.3.1 Greedy dominance rule

This dominance rule is based on an active search for a sequence that corresponds to the same state as that of the current partial solution being explored. To this effect, a new instance is created in which the demand for each model is equal to $[u_1,u_2,\dots,u_M]$. The instance is subsequently solved using the goal-chasing algorithm, as covered in subsection 2.1. If the sequence obtained by the heuristic displays a better SSD value than that of the current partial solution, the node is fathomed because it cannot lead to a better solution than that obtained by the goal-chasing method. If this is not the case, all of the positions in the goal-chasing sequence up to the first position that differs from the partial solution are fixed. The remainder of the sequence is again solved by the goal-chasing process, and its SSD value is compared again with the current partial solution. The process is repeated iteratively until (1) the sequence is found to be dominated and the node is thus fathomed, or (2) the sequence obtained by goal-chasing is exactly the same as the solution from the branch-and-bound procedure and the node is thus not dominated.

If the sequence found by the goal-chasing procedure is different from the original sequence but the SSD value is the same, both sequences are equivalent, and only one must be explored. The decision is made using a tie-breaking procedure. This procedure compares the first differing position between both sequences. The one with the lowest numbered model in that position is considered as dominating the other sequence.

4.3.2 Storing explored states

This dominance rule consists of deciding whether the current partial solution is equivalent to a previously explored partial solution. For this purpose, the nodes are transformed to their equivalent states and are stored in a memory structure that associates each state with its best-known cumulative contribution of the state.

As each partial solution is enumerated, its corresponding state is assessed for equivalence with any of the states stored in the memory structure. If the corresponding state has not been stored, it is now stored together with its value SSD. If the state is present but the stored cumulative contribution is worse than the current value, then the stored value is substituted with the new one. In contrast, if the stored value is better than that of the current one, then the current partial solution is fathomed.

To avoid memory issues, the implementation limits the number of states that can be stored. When the memory limit is reached, the dominance rule continues to assess whether the new partial solutions are dominated, but if the corresponding state is designated as non-dominated, it is not stored.

4.3.3 Symmetrical sequences

To further reduce the number of partial solutions that must be explored, the method attempts to detect symmetries. Two solutions are symmetrical if for each position in the sequence, the model sequenced in position t of the first sequence is identical to the model sequenced in position $T+1-t$ of the second sequence.

The property of symmetry studied in subsection 3.1 states the following: (1) any state has the same contribution as that of its complementary state and (2) any sequence explores a series of states that correspond directly to the states complementary to those explored by its symmetrical sequence; thus, any two symmetrical sequences have the same SSD value.

Based on the aforementioned property, only one of the two symmetrical sequences must be explored. This determination is achieved by ensuring that up until stage $\lfloor T/2 \rfloor$, only the sequences with a lower lexicographical value for their associated states compared to that of their complementary state are explored.

A state $[u_1,u_2,\ldots,u_M]$ is said to have a lower lexicographical value than state $[u'_1,u'_2,\ldots,u'_M]$ if, for the first position i of the vector with different values, $u_i<u'_i$ holds. The rule is applied to each partial solution with up to $\lfloor T/2 \rfloor$ models sequenced.

It is important to note that the dominance rule may present problems with the greedy dominance rule (see subsection 4.3.1). For certain sequences, both rules could label the original sequence and its symmetrical as dominated when at least one of them must be explored. To solve this issue, for any partial solution with over $T/2$ models sequenced, the greedy dominance rule detects whether the sequences produced by the goal-chasing procedure would have been dominated using the symmetrical sequences rule. If this is the case, the greedy dominance rule does not label the partial solution as dominated.

4.3.4 Symmetry in stored nodes

The second dominance rule using the symmetry of the problem is based on the stored states shown in subsection 4.3.2. Each time a state *A* is stored, the procedure assesses whether its complementary state $\overline{A}$ (see Subsection 3.1) has been explored and stored. If complementary $\overline{A}$ exists and state *A* belongs to a stage $t \geq T/2$, it is not necessary to continue developing the current node because the best sequence from *A* to the final stage is the best sequence from the initial stage to $\overline{A}$, and if the best sequence from the initial stage to $\overline{A}$ does not correspond to the best known thus far, the algorithm will eventually find it (and this dominance rule will reconstruct the complete solution). We note that (1) until stage $t \geq T/2$, this property cannot be certified, and (2) in a highly favourable case, only half of the tree, up to stage $T/2$, would need to be explored (Fliedner et al., 2010).

Any state in which the exploration has been halted because of the aforementioned rule is known as a frozen state. Once the branch-and-bound procedure ends, if the method has not yet verified the optimality of the current best solution, let τ be the smallest stage with an unexplored partial solution. The lower bound of any frozen state belonging to a stage *t*, such as $t + \tau \leq T$, must be taken into account to calculate the final lower bound reported by the algorithm because the frozen state has not been fathomed.

Finally, $A \cup \overline{A}$ always constitutes a solution that may improve the best-known solution, and thus, the test for symmetry is also used as a heuristic method to obtain improved upper bounds without reaching terminal nodes of the branch-and-bound tree.

4.4 General structure of the proposed branch-and-bound procedure

The proposed branch-and-bound procedure begins by initialising all of the timers and the previously discussed memory limits. Following this step, the data are read, and the ideal usage rates are calculated.

After storing the data, the models are ordered according to an efficient criterion that prioritises models with a lower demand and breaks possible ties by prioritising the models with a lower cumulative options usage. The re-ordering rule was chosen based on a computational test that compares the proposed reordering, a direct application of the method without re-ordering, and the reordering by prioritising the models with a higher demand.

The following step is applied to obtain a first upper bound by means of a goal-chasing method. Next, a first lower bound is sought by assimilation to a transportation problem. If the solution displays the same SSD value as the lower bound, the solution is proven to be optimal, and the procedure ends.

A second upper bound is calculated using the BDP. Next, the sub-problems with separate options are created and solved by means of a DP. The states are stored, and the combinations that generate lower bounds are determined (these combinations must include all of the options without repetition). Once the lower bounds are known, the problem is solved again by the BDP using the new information.

If the new solution is not proven as the optimum, the branch-and-bound algorithm is initialised using a single processor to run a depth-first search until the node limit is reached (see subsection 4.1). Afterwards, each available thread selects a node from the memory pool using the criterion proposed in subsection 4.1 and begins a depth-first search from the chosen node. For each non-terminal node, the procedure tries to fathom the node using the lower bounds and dominance rules described in Subsections 4.2 and 4.3.

After applying the bounds and the dominance rules, the non-fathomed partial solutions are ordered from lowest to highest lower bound, and the node associated to first partial solution in the resulting order is chosen for exploration. This partial solution is explored until a terminal node is reached. Then, the procedure backtracks until a non-developed node is found with a bound lower than the best upper bound and the ramification process is continued. If the node limit marking the end of the depth-first exploration is reached, all of the non-explored partial solutions are stored in the memory pool, and the thread selects a new node to develop according to the criterion exposed in subsection 4.1.

When the time limit for the global procedure is reached or the optimum is verified, the procedure ends. Afterwards, if the optimality has not been verified, a lower bound is calculated using the non-developed nodes and the frozen nodes.

5 Computational experience

To determine the quality of the procedure presented in the previous sections, the algorithm has been implemented in C++ and executed in a four 3.2 GHz Intel Core i5 650 processor computer with 4 GB RAM using a 64-bit 3.4.4 Linux operating system. The implementation has been tested using two instance sets from the literature described below:

- Original and modified instances from Bautista et al. (1996), expanded as shown in Jin and Wu (2002): The original set contains 225 instances with integer valued usage of options. The number of options are P={4,5,8}, the number of models is fixed at M=4, and the number of total units sequenced is 20. The expanded set (Jin and Wu, 2002) multiplies the demand of each model by 10 to generate instances with 200 units. In order to evaluate the effect of increasing the number of units, we also multiply the instance by 2, 3,…,10 generating a total of 2250 instances. Miltenburg (2007) also proposes five instances using a similar approach with increased number of models, options and units, which we will discuss separately.

- Instances generated as described in Drexl and Kimms (2001). These instances were also used in Drexl, Kimms and Matthiessen (2006) and Yavuz (2013): 60 instances were generated following the method described in the aforementioned paper. Instances differ on the total number of units (T={10,20,30,40,50,60,70,80,90, 100}), number of options (P={3,5,7}), and ratio of options usage (divided into two sets: easy and hard). Instances are then created saturating the given ratios of the options usage. Given a number of units (T), a number of options (P) and a set of constraints (one per option j: N_j/D_j), the instance is created as follows: taking each option j separately, the N_j first units have a usage of 1 for the option and the following D_j-N_j have a usage of 0. This process is repeated for the $n \cdot D_j$-th units until T is reached. When this process has been performed for every option, the units with the same usage for all the options are merged into one model.

The reported computational experience corresponds to the following combination of parameters: the window size (w) for the BDP procedure is set to 1000 states, the limit of nodes to explore with a depth-first search in the branch-and-bound procedure is set to 10000, the number of nodes to be stored in the memory pool before starting a depth-first criterion instead of a best-first is set to 50000, and the memory reserved for storing nodes is 2 GB. These parameters for the BDP and number of depth-first searches were selected by comparing the results of different sets of parameters on a subset of the proposed instances. Note that the results would not significantly change if similar values were used. The limit of the depth-first criterion and the memory reserved for storing nodes were chosen in order to avoid surpassing the 4GB RAM of memory available in the computer used for the experiments (larger values of memory reserved for storing nodes would improve the quality of the results).

Additionally, the branch-and-bound time was limited to 1 hour of CPU time. Consequently, the branch-and-bound can be compared with a single thread algorithm with the same allotted time. The parallelism provides an improvement in terms of the real time required to compute a solution and, possibly, some performance improvements due to earlier detection of improved solutions.

The optimality gaps for any lower bound are calculated using equation (12), where *LB* is the lower bound being tested and UB_{best} is the best known upper bound.

$$Opt.Gap\,LB(\%) = \left(UB_{best} - LB\right)/UB_{best} \cdot 100 \qquad (12)$$

The optimality gap for any upper bound is calculated as shown in (13), where *UB* is the upper bound being tested and LB_{best} is the best known lower bound when the upper bound procedure is applied.

$$Opt.Gap\,UB(\%) = \left(UB - LB_{best}\right)/UB \cdot 100 \qquad (13)$$

Table 2 provides results for the instance set used in Bautista, Companys and Corominas (1996) and Jin and Wu (2002). The table reports the results from the BDP and the branch-and-bound procedure. Please note that that the BDP method provides the optimal solution of all of the 2250 instances, and the branch-and-bound algorithm verifies its optimality within the imposed time limit. Hence, table 2 only reports measures of time requirement (average number of explored nodes and average running time). Only average values are reported, but the maximum number of explored nodes and the maximum computing time required by any of the 2250 instances is 8720 and 3.54 seconds respectively.

These results show that the proposed algorithm, dominance rules and lower bounds are very effective to solve these instances, and we can conclude that the aforementioned instances are easily solved using the algorithms proposed in this paper.

Table 2: Average number of nodes before verifying optimality and average running time (in seconds) of the branch-and-bound (columns B&B) and BDP method (columns BDP) for the extended instance set based on Bautista, Compans and Corominas (1996). Instances are grouped by size (number of units to schedule).

size	**BDP t(s)**	**B&B #nodes**	**B&B t(s)**	**size**	**BDP t(s)**	**B&B #nodes**	**B&B t(s)**
20	0.003	97	0.003	**120**	0.399	581	0.411
40	0,034	194	0.035	**140**	0.496	678	0.515
60	0,12	291	0.122	**160**	0.602	775	0.632
80	0,207	388	0.211	**180**	0.714	872	0.757

100	0,302	484	0.309	**200**	0.814	969	0.877

In Miltenburg (2007) five instances derived from the work by Bautista, Companys and Corominas (1996) were used to test the performance of the proposed DP approach. A basic instance with 5 models, 4 options and 48 units was considered. This instance was solved using different objective functions, including the objective considered in this paper (SSD minimization). For the SSD objective, their DP required 45.22 seconds and explored 12625 nodes (Miltenburg, 2007, p. 3569). Compared to these results our branch-and-bound algorithm is capable of obtaining the optimal solution in the root node in less than a second, clearly indicating the improvement provided by the proposed lower bounds and dominance rules (even considering that the computer used in the present experiment is much faster than the computer used by Miltenburg).

For the remaining four instances considered in Miltenburg (2007), the paper does not use the SSD objective, preferring to minimize the maximum absolute deviation (MADT). The decision to use MADT is based on the observation that it is easier to solve than the other tested objectives while obtaining high quality solutions even if the solution is evaluated using any of the other objectives. Therefore, a direct comparison of our method with the DP presented in Miltenburg (2007) for the remaining instances is not possible.

Nevertheless, we tried to solve the remaining four instances using our algorithm and a SSD minimization objective. These instances are constructed by doubling the number of units, the number of options, the number of models or all of the three characteristics. The proposed branch-and-bound algorithm was only able to solve the problems in which the demand of the models, or the number of components is doubled (the optimal solutions were found in less than a second in both cases).

For the remaining two cases, the method runs out of memory during the initialization phase of the separability bounds. The reason is that these instances feature multiple different levels of usage of options among the models, leading to subproblems with the same number of models than the original problem during the separability lower bounds calculation.

This limitation leads us to consider that the proposed algorithm is limited to problems in which the models share similarities among levels of usage of the options (as the subproblems will contain fewer models, and the DP will require fewer states).

An example of real life problem with the aforementioned conditions corresponds to the Car Sequencing problem, as the levels of consumption are binary (presence or absence of a specific part or job).

The remainder of the section is devoted to instances with the Car Sequencing characteristics, as those found in Drexl and Kimms (2001).

Table 3 compares the quality of the transportation and separability lower bounds when instances are combined according to size. The results show that the transportation bound displays a much more steady value, and that it degrades as the instance size increases. In contrast, the separability lower bound shows a tendency to tighten as the instance size increases, and it outperforms the transportation bound for instances with 30 or more units. The results also show that both methods provide much better lower bounds on these instances than on the instances derived from Bautista, Companys and Corominas (1996).

Table 3: Optimality gaps (12) for the separability (column separability) and transportation (column transportation) lower bounds for the instance set from Drexl and Kimms (2001), when instances are grouped by size.

Size	**separability**	**transportation**	**Size**	**separability**	**transportation**
10	4.71	3.09	**60**	1.26	3.89
20	7.82	7.36	**70**	1.84	3.58
30	1.50	3.99	**80**	1.74	4.52
40	2.11	4.95	**90**	0.92	4.25
50	3.05	4.33	**100**	1.35	4.07

Table 4 compares the results provided by the branch-and-bound and the BDP procedure. We report individual results for instances with $T \geq 60$ and $P \geq 5$. For instances with up to 50 units, the BDP is always able to find the best known solution, and the branch-and-bound procedure is capable of verifying its optimality within seconds. For instances with three options, the problem is optimally solved in a reduced computing time using the DP proposed in Section 3.2 to calculate the separability bound, hence, they are considered easy instances. Note that we attempted to use the DP method to solve problems with a greater number of options, much in the line of previous proposals such as Miltenburg (2007) or Fliedner and Boysen (2008), but this approach was deemed unfeasible due to memory and time constraints. For each instance, the table reports the optimality gap (13) before branching, column root, the optimality gap (13) of the branch-and-bound method, column B&B, the running time required by the branch-and-bound measured in seconds, column Time B& B (s), and the number of nodes of the branch-and-bound tree, column #nodes B&B.

Table 4: Results of the BDP and branch-and-bound for the instance set found in Drexl and Kimms (2001). For each instance (column size, options and constraints), the optimality gap (13) for the BDP (column BDP), the optimality gap (13) for the branch-and-bound algorithm (column B&B), the running time required by the branch-and-bound method and the number of nodes required by the branch-and-bound are reported. The boldface type indicates the instances in which the branch-and-bound improves the upper bound provided by the BDP.

Size	Options	Constraints	root (13)	B&B (13)	Time B&B s	# nodes B&B
60	5	Easy	1.836	0	0.4	1
		Hard	1.770	0	2.1	24301
	7	Easy	0.949	0	43.4	257738
		Hard	3.045	0	368.2	9995393
70	5	Easy	2.156	0	0.9	1
		Hard	0.331	0	1.5	5587
	7	Easy	3.513	0	604.2	4075382
		Hard	**3.574**	**1.878**	**1124.5**	**21394887**
80	5	Easy	3.129	0	1.7	1
		Hard	0.747	0	3.4	10528
	7	Easy	1.137	0	850.7	5907647
		Hard	2.341	1.994	2030	16984304
90	5	Easy	0.247	0	2.9	1
		Hard	0.945	0	9.6	74005
	7	Easy	**0.480**	**0**	**3240.5**	**15608831**
		Hard	**4.104**	**3.104**	**3600**	**13229696**
100	5	Easy	0.760	0	4.1	1
		Hard	0.566	0	8.4	29438
	7	Easy	3.141	2.519	3600	15274535
		Hard	**3.362**	**3.334**	**3600**	**11513364**

The results show that both the BDP and the branch-and-bound procedures are able to effectively solve most of the instances in the set. As with the instances proposed in Bautista et al. (1996), Boysen et al. (2008) and Jin and Wu (2002), the BDP implementation is capable of obtaining the best known solution in most cases (16 out of 20 instances) whereas the branch-and-bound algorithm is able to verify the optimality of the solutions found for most of the instances (15 out of 20 instances). Furthermore, the difficulty of each instance depends on both the number of options and the size of the instance, but it is noteworthy that the original distinction between easy and hard instances, which originated from a Car Sequencing problem, still applies to the lower bounding performance during the branch-and-bound.

If we compare the results of the instance set from Bautista, Companys and Corominas (1996) with the instance set from Drexl and Kimms (2001), we can see that the latter set contains much harder instances, as the algorithm is unable to close five of the instances within the imposed time limit even if the number of models to schedule and the options to consider is smaller. We conjecture that even if the lower bounds provide smaller gaps, as the number of models in these instances is larger, a larger branching factor is responsible of the increase in running times.

6 Summary and conclusions

Optimally solving sequencing problems in mixed-model assembly line systems aids in balancing workloads for workers and managing the line in a Just-In-Time fashion. In this paper, a branch-and-bound exact method using several lower bounds and dominance rules is proposed to solve the ORV problem, one of the variants found in the literature that addresses the balancing workload objective.

The most characteristic features of the presented algorithm are the usage of a new lower bound based on separating the problem by options, and use of dominance rules to exploit the symmetry of the problem.

These new techniques, together with the previously known BDP method to obtain initial upper bounds, result in an efficient branch-and-bound algorithm. The results obtained in the computational experiments show that the procedure verifies optimality for instances that are double the size of those solved by any exact algorithm for this problem or other similarly complex problems known to date (Bautista et al 1996; Boysen et al. 2008, Drexl and Kimms; Fliedner et al. 2010; Yavuz, 2013). Let us note that instances solved to optimality in the present paper are of equal or larger size than instances used for heuristic comparison in previous papers (Erel et al., 2007; Jin and Wu, 2002; Yavuz, 2013). The exact algorithm is a viable alternative for scheduling problems with up to 200 units and 8 options, which have been optimally solved within reduced running times.

The main limitation of the algorithm corresponds to the limits of the separability lower bound, which requires that the models in the instance share similar levels of option requirements, as in the Car Sequencing instances and some of the level scheduling instances in the literature. This limitation is highlighted when instances with differing characteristics, like the ones studied in Miltenburg (2007) are considered, but even in such a case, larger instances than those reported in the literature for the same objective function are optimally solved.

The results also demonstrate the quality of the BDP working as a heuristic for the problem at hand, as the procedure obtains most of the best-known solutions even for large instances within very reduced running times (in all of the tested instances, the required time to reach a solution using the BDP algorithm was below 30 seconds).

Further research should focus on problems with larger number of options or multiple usage levels for each option, as the separability lower bound, which appears to be the most effective lower bounding procedure, may not be used on these instances due to memory requirements. Another line of possible research is the application of the proposed method to other objectives. While the SSD objective corresponds to the original formulation provided in Monden (1983), the literature has identified and used several other objectives. Obviously, each objective would require the development of different lower bounding techniques, and possibly modify some of the dominance rules described in the present work.

Appendix A

Considering the variable Y_j defined in subsection 3.3, the contribution of any option j to the objective function in this instant can be expressed as shown in (A.1).

$$(Y_j - r_j \cdot t)^2 \tag{A.1}$$

This expression is a continuous and differentiable function that has a single minimum. Let us consider the value of a coefficient C_{it}, as shown in (A.2), which is also a continuous and differentiable function with a single minimum.

$$(Y_j - c_{ij} - r_j \cdot (t-1))^2 + (Y_j - r_j \cdot t)^2 \tag{A.2}$$

If we derive (A.2) with respect to Y_j we obtain (A.3).

$$Y_j = r_j \cdot t + (c_{ij} - r_j)/2 \tag{A.3}$$

Let Y_j* be the minimum value yielded by (A.3). Given that the cumulative value for the usage of an option must be an integer number and a single minimum exists, the optimum must take on the value $\lfloor Y_j^* \rfloor$ or $\lceil Y_j^* \rceil$. Consequently, only situations with a usage between (9) and (10) must be considered.

References

Bautista, J.; Cano, J., 2008. Minimizing work overload in mixed-model assembly lines. Int J Prod Econ, 112, 177-191.

Bautista, J.; Companys, R.; Corominas, A., 1996. Heuristics and exact algorithms for solving the Monden problem. Eur J Oper Res, 88, 101-113.

Bautista, J.; Pereira, J.; 2009. A dynamic programming based heuristic for the assembly line balancing problem. Eur J Oper Res, 194, 787-794.

Bellman, R.E., 1957. Dynamic Programming. Princeton University Press, Princeton, NJ.

Boysen, N.; Fliedner, M.; Scholl, A., 2007. Sequencing mixed-model assembly lines: Survey, classification and model critique. Eur J Oper Res, 192, 349-373

Boysen, N.; Fliedner, M.; Scholl, A., 2008. Sequencing mixed-model assembly lines to minimize part inventory cost. OR Spectrum, 30, 611-633.

Boysen, N.; Fliedner, M.; Scholl, A., 2009. The product rate variation problem and its relevance in real world mixed-model assembly lines. Eur J Oper Res, 197, 818-824.

Boysen, N.; Kiel, M.; Scholl, A., 2011. Sequencing mixed-model assembly lines to minimize the number of work overloads situations. Int J Prod Res, 49, 4735-4760.

Cordeau, J-F.; Laporte, G.; Pasin, F., 2008. Iterated tabu search for the car sequencing problem. Eur J Oper Res, 191, 945-956.

Drexl, A.; Kimms, A., 2001. Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints. Manage Sci, 12, 480–491.

Drexl, A.; Kimms, A.; Matthiessen L, 2006. Algorithms for the Car Sequencing and the Level Scheduling Problem. J Sched, 9, 153-176.

Erel, E.; Gocgun, Y.; Sabuncouğlu, I., 2007. Mixed-model assembly line sequencing using beam search. Int J Prod Res, 45, 5265-5284.

Fliedner, M.; Boysen, N., 2008. Solving the car sequencing problem via Branch & Bound. Eur J Oper Res, 191, 1023-1042.

Fliedner, M.; Boysen, N.; Scholl, A., 2010. Solving symmetric mixed-model multi-level just-in-time scheduling problems. Discrete Appl Math, 158, 222-231.

Goldberg, A. ; Tarjan, R., 1989. Finding minimum-cost circulations by cancelling negative cycles. J. ACM, 36, 873-886.

Gravel, M.; Gagné, C.; Price, W.L., 2005. Review and comparison of the three methods for the solution of the car sequencing problem. J Oper Res Soc, 56, 1287-1295.

Jin, M.; Wu, D.S., 2002. A new heuristic method for mixed model assembly line balancing problem. Comput Ind Eng, 44, 159-169.

Kubiak, W.; Sethi, S., 1991. A note on schedules for mixed-model assembly lines in just-in-time production systems. Manage Sci, 37, 121-122.

Kubiak, W., 1993. Minimizing variation of production rates in just-in-time systems: A survey. Eur J Oper Res, 66, 259-271.

Leu, Y. –Y.; Huang, P. Y.; Russell, R. S.; 1997. Using beam search techniques for sequencing mixed-model assembly lines. Ann Oper Res, 70, 379-397.

Miltenburg, J., 1989. Level schedules for mixed-model assembly lines in just-in-time production systems. Manage Sci, 35, 192–207.

Miltenburg, J.; Sinnamon, G., 1989. Scheduling mixed-model multi-level just-in-time production systems. Int J Prod Res, 27,1487–1509.

Monden, Y., 1983. Toyota production system (2nd ed) Institute of Industrial Engineering, Norcross, GA

Prandtstetter, M.; Raidl, G. R., 2008. An integer linear programming approach and an hybrid variable neighborhood search for the car sequencing problem. Eur J Oper Res, 191, 1004-1022.

Solnon, C., 2008. Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. Eur J Oper Res, 191, 1043-1055.

Solnon, C.; Cung, V-D.; Nguyen, A.; Artigues, C., 2008. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. Eur J Oper Res, 191, 912-927.

Sumichrast R. T ; Russell, R. S., 1990. Evaluating mixed-model assembly line sequencing heuristics for just-in-time production systems. J Oper Manag, 9, 371-390.

Tavakkoli-Moghaddam, R.; Rahimi-Vahed, A. R., 2006. Multi-criteria sequencing problem for a mixed-model assembly line in a JIT production system. Appl Math Comput, 181, 1471-1481.

Yano, C. A; Bolat, A., 1989. Survey, development, and applications of algorithms for sequencing paced assembly lines. Technical Report 88-13.

Yano, C. A.; Rachamadugu, R., 1991. Sequencing to minimize work overload in assembly lines with product options. Manage Sci, 37, 572-586.

Yavuz, M., 2013. Iterated beam search for the combined car sequencing and level scheduling problem. Int J Prod Res, 51, 3698-3718.

Zhu, J.; Ding, F-Y., 2000. A transformed two-stage method for reducing the part-usage variation and comparison of the product-level and part-level solutions in sequencing mixed-model assembly lines. Eur J Oper Res, 127, 203-216.