Sequencing and Scheduling in Coil Coating with Shuttles

Wiebke Höhn[1], Felix G. König[1], Marco E. Lübbecke[1], and Rolf H. Möhring[1]

Technische Universität Berlin, Institut für Mathematik, MA 5-1
Straße des 17. Juni 136, 10623 Berlin, Germany
{hoehn, fkoenig, m.luebbecke, rolf.moehring}@math.tu-berlin.de

Abstract. We consider a complex production planning problem in integrated steel production. Coils of sheet metal need to be coated in several stages with few of a broad variety of coating materials. Different coil geometries as well as changes in the required coating materials may necessitate time-consuming setup work. Most coating stages are equipped with so-called shuttles, i.e., two parallel color tanks are available at a coater to reduce setup time. Devising a production plan for coil coating comprises the sequencing of coils, the assignment of coils to color tanks, and the scheduling of scarce work resources for setup work, partly performed concurrently with production. The aim is to minimize the makespan for a given set of coils.
We present a precise mathematical formulation of the problem in its entirety and develop a graph theoretical model for the integrated tank assignment and setup scheduling problem. The latter can be solved quickly to optimality in the case where no constraints on work resources are present. For the resource constrained case, the model gives rise to a fast heuristic which is integrated into a genetic algorithm to solve the sequencing problem in practice. We evaluate the quality of our algorithm via a combinatorial relaxation of the problem. This is formulated as an integer program and solved by branch-and-price. It shows that our solutions are within 5–15% of the optimum.
Based on our model and the fast heuristic, we developed a software package in cooperation with PSI Business Technology, which is about to enter production at Salzgitter Flachstahl GmbH, a major German steel producer. Our optimized plans yield a reduction in non-productive time by over 20% compared to the system in place and have greatly exceeded expectations. The integrity of the solutions obtained has been verified by the responsible planners in Salzgitter.

Keywords. sequencing, scheduling, coil coating, mutli-interval graphs, heuristics, branch-and-price

1 Introduction

Almost every stage in integrated steel production requires solving a sequencing problem as subproblem, that is, deciding about a *good* order of steel slabs or coils

Dagstuhl Seminar Proceedings 09261
Models and Algorithms for Optimization in Logistics
http://drops.dagstuhl.de/opus/volltexte/2009/2165

in which they should be processed. This starts with the melt in the blast furnace, adding product specific alloys, and casting the strand of hot melt to solid slabs; continuing with surface treatment of the slabs, rolling them in the hot and cold rolling mills to coils, before applying the surface finishing. Even with today's state-of-the-art operations research methodology, this entire process is far too complex to be planned in an integrated manner. In practice, the different stages are considered hierarchically, and much planning is essentially done manually. In this paper we deal with the final processing step in sheet metal production, the coating of steel coils, which turns out to be much more than just a complicated sequencing problem.

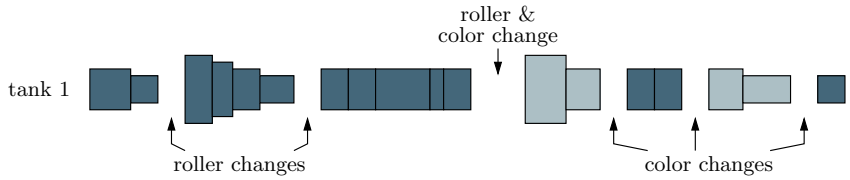
The coil coating process plays an essential role in shaping steel producers' extremely diverse product portfolio: The coils used for home appliances, for instance, already have their typical white coating when bought from the steel supplier; the sheet metal used for car bodies already has an anti-corrosion coating before it arrives at the automotive plant for pressing; coils destined for building construction receive their coatings, which are very specific for technical as well as aesthetic reasons, while still at the steel plant.

Steel producers and manufacturers of coating materials on the one hand and distributors of pre-coated sheet metal on the other hand have formed associations to promote the evolution of coil coating on national[1] and international[2] levels already in the 1960s. Progress in the development of new and improved coating materials and techniques fosters an ongoing diversification in pre-coated metal products, and in recent years there have been quite a few scientific publications on coil coating, e.g., [1], [2]. Yet, to the best of our knowledge, the present work is the first dealing with optimization in the planning process for coil coating.

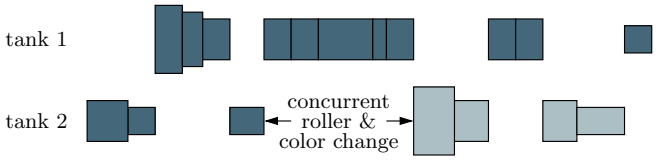
As is typical for paint jobs, the coil coating process may be subject to long setup times, mainly for the cleaning of equipment, and thus very high setup cost. In order to reduce this cost, so-called *shuttle coaters* have been introduced. In our application, four coaters apply one of two required coating layers to either the top or the bottom side of a coil. Three of these coaters are shuttle coaters, i.e., they possess two separate tanks which allow to hold two different coatings at the same time. The advantage is twofold: The shuttle can be used to switch between two different coatings on the same coater at (essentially) no setup cost; or alternatively, the unused tank can be set up already while coating is performed from the other tank in the meantime. This possibility of setting up a coater *during production* is called *concurrent setup*; see Fig. 1 for an example. It is of key importance for the whole paper.

The introduction of shuttles significantly changes the flavor and the complexity of the sequencing problem. As we will see, it necessitates the solution of an integrated resource allocation and scheduling problem for the evaluation of a given sequence: Which tank do we use for which coil, and how do we schedule concurrent setup work without exceeding available work resources?

[1] http://www.coilcoating.org/
[2] http://www.eccacoil.com/



(a) Solution for a classical coater with only one tank.



(b) Solution for a shuttle coater.

Fig. 1. Coils of different widths as they are processed in the coating line (from left to right). Setup work (such as color or roller changes, to be defined later) has to be performed whenever a coil has to coated with another color or is wider than its predecessor on that tank.

We develop a fast, practical heuristic which solves the integrated sequencing, resource allocation, and scheduling problem and computes an exact and detailed production plan for the coil coating line. An elegant connection to a graph theoretic model paves the way for the design of our algorithm. The quality of our plans is assessed with the help of an integer program which we solve by branch-and-price.

Our algorithm is being integrated into PSI Business Technology's planning software suite[3], and is about to be installed at Salzgitter Flachstahl GmbH[4] (SZFG for short). There, it yields a 20% reduction in setup times as compared to the previous manual planning process. In addition, our lower bounds prove that the makespan of the solutions computed by our algorithm is within 5–15% of the optimal makespan for typical instances.

Furthermore, we justify our heuristic approach by proving strong *NP*-hardness, even for the resource allocation and scheduling subproblem, for which we still present a fixed-parameter tractable algorithm.

2 Problem Formulation

We consider the coil coating process, the very last production step of the integrated steel production line. On-site logistics before and after the surface finishing steps are amply dimensioned, thus, the actual coating process poses a main

[3] www.psi-bt.de
[4] www.salzgitter-flachstahl.de/en/

bottleneck in filling customer orders on time. Since orders have strict deadlines, the set of coils which need to be coated in roughly a 24-hour planning period is fixed by a higher-level planning system.

Coils are coated continuously, and during non-productive time, scrap coils are run in between actual coils, so essentially a never-ending strip of sheet metal is running through the coating line. When entering, a coil is unrolled and stapled to the end of its predecessor. After undergoing some chemical conditioning of their surface, the coils run through a top and bottom primer coater, an oven, a top and bottom finish coater, and through a second oven. A coater essentially consists of one or two tanks holding the required coating material, which is applied from the tank to the surface of the coil by a rubber roller. Coaters with two parallel tanks are called *shuttle coaters*. In the ovens, the respective coating layers are fixed. After the coating process, the coils are rolled up again, now ready for shipping.The coil coating line at SZFG is equipped with three shuttle coaters and one classical coater; seeFig. 2.We will intuitively refer to the possible types of coating material as *colors*.

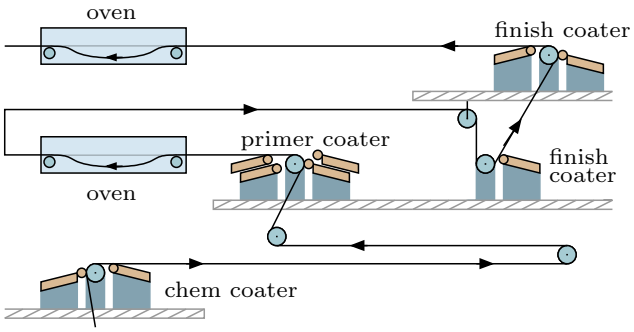


Fig. 2. Schematic view of the coil coating line.

Non-productive time may ensue during production for various reasons: Depending on the order of the coils and the usage of the shuttle coaters, setup work must be performed. To satisfy certain technical restrictions, intermediate (scrap) coils of different lengths are required in between certain coils. Finally, sample runs need to be conducted at certain points in the sequence. We refer to all of the above as *setup*, which may (or may not) entail an increase of the time needed to coat the given set of coils, as we will describe in Sect. 2.3.

Naturally, in order to ensure smooth operation, a production plan needs to include all information relevant to the above in sufficient detail. Let us assume $[n] := \{1, \ldots, n\}$ is the set of coils to be coated. A plan for the coil coating process consists of the following three parts:

- a *sequence* $\pi \in \Pi_n$, i.e., a permutation stating the order in which the coils are to be run through the plant,

- a *tank assignment* $T \in \{1,2\}^{n\times 3}$ stating for each coil from which tank it is coated for each of the three shuttle coaters,
- a *schedule* $S \in [n] \times \mathbb{R}^{|W|}$ assigning to each setup task $w \in W$, the set of necessary setup, a pair (i_w, s_w), where i_w is the last coil which started no later than w, and $s_w \geq 0$ is the time from i_w's start time until w starts.

The optimization goal is to determine a coil coating plan (π, T, S) minimizing the *makespan*, i.e., the completion time of the last coil in the sequence π. The schedule S needs to be feasible w.r.t. criteria described in Sect. 2.3. Throughout this paper, we refer to the included non-productive time in between coils due to setup work, scrap coils and coils for sample runs, as *cost*, which is always associated with time durations, e.g., time needed for setup work or the runtime of scrap coils.

As we will describe below in more detail, the set W of necessary setups is completely determined by the coil order π and the tank assignment T. It is essential to note, that setup does not necessarily have an impact on the makespan of a plan. In Sect. 2.3, we discuss when setup tasks may be performed while coils are being coated, thus entailing no cost. We refer to such setup tasks, which are performed concurrently with production, as *concurrent setup*.

Since all necessary setup depends on the characteristics of the coils, we will state the most important ones for a better understanding. Coils usually have a length of 1–5 km, running for 20–100 minutes, and some of their central attributes are of course the colors for the four coating stages, chosen from a palette of several hundred. But also their width and height—usually 1–1.8 m, and 1–3 mm, respectively—their processing speeds, temperatures in the primer and finish oven, or the information if they require a protective surface lamination impact setup times. We refrain from listing further coil attributes, since their list is very long; yet all relevant information is included in our setup calculations.

We classify every setup either as *local* or *global setup*. By a local setup, we refer to a setup depending only on a pair of consecutive coils in the sequence π, whereas a global setup depends on consecutive coils *run on the same tank*, and thus may depend on the whole subsequence up to its occurrence.

2.1 Local Setup

Local setups are always associated with the insertion of scrap coils in the production sequence. They mainly occur due to restrictions on the difference in certain attributes of the coils. An example is a difference in the weight per meter of the coil. The reason is the long distance between the supporting points in the oven: If the weights of consecutive coils differ too much, the stapled joint between the coils could break. There are many more restrictions that limit certain differences. We omit a detailed description, since for the later considerations only the structure of these restrictions is important. Let $\delta_r(i,j)$ denote the necessary length of a scrap coil between consecutive coils i, j to bridge the difference in criterion r, then the total time for scrap coils needed in between i and j is given by

$$s_{\text{loc}}(i,j) := \max_r \{\delta_r(i,j)\}.$$

Another class of local setup is caused by *sample runs*, i.e., a coating of scrap coils to check the quality of the coating. Local sample runs are e.g., required when protective surface lamination is applied to a coil for the first time, i.e., its predecessor did not receive lamination. Let $t_{\text{loc}}(i, j)$ denote the duration of the local sample run necessary if coils i and j are run consecutively. The time for the sample is added to the remaining setup time, since a sample run requires the same conditions as the actual coil coated afterwards, hence, we cannot use a scrap coil inserted previously for a different reason.

Also note that local setup is completely oblivious of tank assignment and solely depends on the order of the coils given by π.

2.2 Global Setup

At each stage of the coating process, the coating is extracted from a tank in the coater and applied to the coil surface by a rubber roller. When the tank assignment T assigns coils i and j to the same tank t in some coating stage, i is the last coil run before j *on that tank*, and i and j require different colors in this coating stage, tank t needs to be cleaned and refilled with j's color, resulting in a setup task added to the set W of tasks scheduled by S that has to be performed after i ends, but before j starts.

Quite similarly, the rollers incur wear at the edges of the coil in the coating process. Consequently, a roller that has been used to coat a coil i of width w_1 cannot be used to coat a coil j of width $w_2 > w_1$ later in the plan, as $j's$ coating would bear imperfections stemming from the wear the roller incurred at the edges of i. So whenever a coating plan calls for the coating of a coil j which is wider than the coil i coated last *on the same tank*, the roller needs to be changed after i ends and before j starts—a corresponding setup task is added to W.

Color and roller changes require the same amount of time. We denote the setup time for a color or roller change by t_c. Furthermore, let $s_{cc}^{(m)}(i, j)$ and $s_{rc}(i, j)$ denote the time for setup in between coils i and j on coater m due to a color and roller change, respectively. Then

$$s_{cc}^{(m)}(i,j) = \begin{cases} t_c & \text{if } i \text{ and } j \text{ have different colors on } m \\ 0 & \text{otherwise} \end{cases},$$

$$s_{rc}(i,j) = \begin{cases} t_c & \text{if } i \text{ is wider than } j \\ 0 & \text{otherwise} \end{cases}.$$

We define $s_{\text{glob}}^{(m)}(i, j) = s_{cc}^{(m)}(i, j) + s_{rc}(i, j)$.

A third source of global setup can be global sample runs, which are required for some predetermined complicated coatings when they are first applied for the first time *from a certain tank*, i.e., after a color change. Let $t_{\text{glob}}(i, j)$ denote the duration of the global sample run necessary if coils i and j are run consecutively on the same tank. One example for global sample runs are coils for home appliances, where the correct shade of white is very important. When there is a

global as well as a local reason for a sample run (cf. Sect. 2.1), only one sample run needs to be performed.

In the case that no setup for coil i could be performed concurrently with production, the total setup $s(i)$ for i is given by

$$\begin{aligned} s\big(\mathrm{pred}_{\mathrm{tank}}(i),\, \mathrm{pred}_{\mathrm{seq}}(i),\, i\big) = \;& \max\big\{s_{\mathrm{loc}}(\mathrm{pred}_{\mathrm{seq}}(i), i),\; \sum_m s^{(m)}_{\mathrm{glob}}(\mathrm{pred}_{\mathrm{tank}}(i), i)\big\} \\ & + \max\big\{t_{\mathrm{loc}}(\mathrm{pred}_{\mathrm{seq}}(i), i),\; t_{\mathrm{glob}}(\mathrm{pred}_{\mathrm{tank}}(i), i)\big\}, \end{aligned}$$

where $\mathrm{pred}_{\mathrm{seq}}(i)$ and $\mathrm{pred}_{\mathrm{tank}}(i)$ denote the coil directly preceding i in the sequence, and on the same tank, respectively.

In order to determine if a global setup occurs for a coil i in the sequence, it is clearly not sufficient to consider i's predecessor in π as it is for local setup—rather the last coil run before i *on the same tank* as i needs to be taken into account. Consequently, in order to evaluate the setup of a given (sub-)sequence, a tank assignment T for it must already be given.

As mentioned before, global setup does not necessarily result in cost, as illustrated in Fig.3: The first, orange part of global setup w on tank 2, takes place after coil i while coils are being coated on tank 1, and tank 2 is idle. When coil j finishes coating, a period of non-productive time begins. First, a scrap coil is run to prepare for coil k—during this time, the global setup still does not cause any additional cost. Only once this scrap coil has finished, an additional scrap coil needs to be inserted for the remaining duration of w, now leading to an actual increase in cost due to w. Once w finishes, the required sample for coil k may begin and production continues.

In the following section, we will describe how concurrent setups may be scheduled and it will become even clearer, why the problem of computing a coating plan (π, T, S) does not decompose well into the underlying subproblems, but that the problem needs to be tackled in its entirety.

2.3 Tank Assignment and Concurrent Setup

As described in the previous section, a good tank assignment can save global setup by "keeping" a color and/or a roller for a later coil and using the other tank for coating coils in the meantime. Essentially, this reduces the number of jobs in W, such that we may hope to be able to find a schedule S for W which results in a shorter makespan.

In this section, we describe a second, in a sense opposite way of utilizing the second tank of a shuttle coater to obtain better solutions. As mentioned before, many of the setup times incurred in coil coating are due to tank cleaning and roller changes. However, whenever a tank is not used for an extended period of time, setup work on this tank may be performed simultaneously with regular production, thus avoiding some cost (cf. Fig. 3). Since regular production requires a share of the work resources available, concurrent setup work cannot be performed as quickly as setup work during non-productive time, i.e., in between processing two coils. We denote the time for a color and roller change during

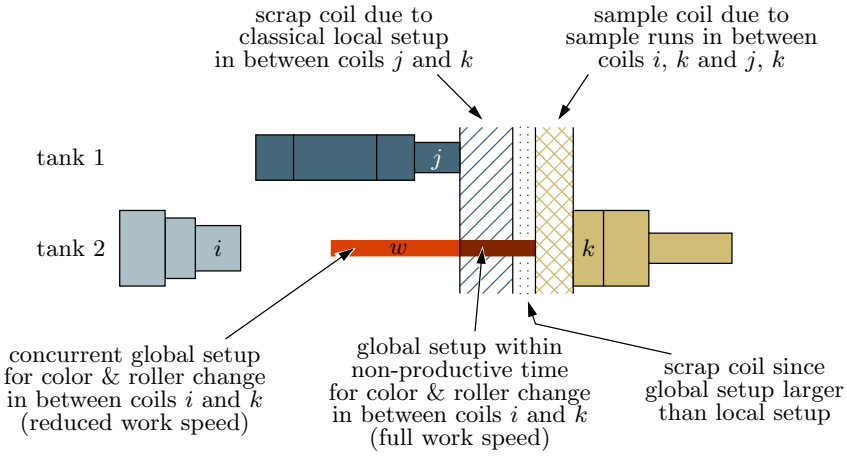
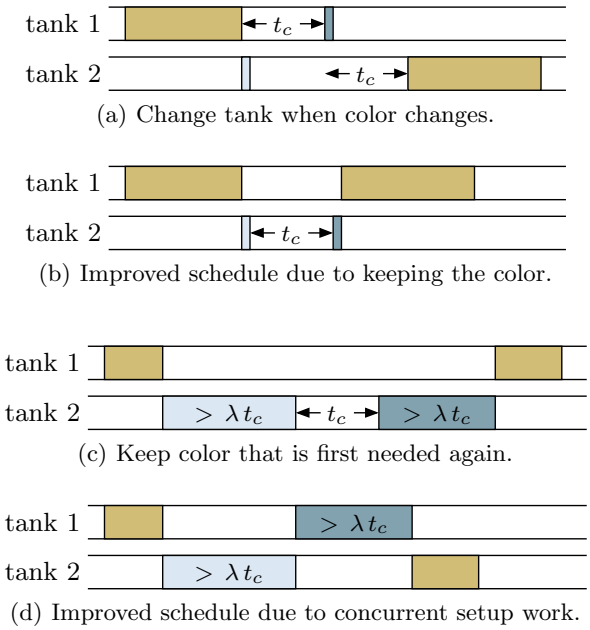


Fig. 3. Components of necessary non-productive time, i.e., of cost.

non-productive time by t_c. The slowdown factor incurred by concurrent setup during production is denoted by $\lambda \geq 1$. In our application, there is only one team of workers who can perform setup work during regular production, so at any point in time, only one concurrent setup job may be performed. Moreover, concurrent setup work does not allow preemption by another concurrent setup job, i.e., a concurrent setup job on one coater must be finished before concurrent setup work is started on another.

In particular, concurrent setup is possible on all idle tanks (and their rollers), but only on one tank at a time. On the one hand, concurrent setup can save cost, but on the other hand, it conflicts with keeping a certain color and/or roller in a shuttle tank for a later coil. This illustrates another inherent difficulty in optimal planning for shuttle coaters: a plan needs to include an assignment of tanks to coils for every shuttle coater, such that maximum savings in cost are realized from a combination of concurrent setup and keeping colors and/or rollers in unused tanks for later coils (thereby avoiding some setup completely).Already very small examples show that simple rules like the one previously in use at SZFG, which switched tanks on a shuttle coater whenever a new color is required, fail to be optimal;see Fig. 4.

Note that it suffices to consider global setup tasks only, when scheduling the tasks in W. Scrap coils and local sample runs can easily be added to the schedule S after it has been computed for the remaining tasks: Scrap coils do not require any work resources and can thus be added to S in parallel, delaying the start time of the following coil j only if the length of the scrap coils exceed the length of global setup in between coils, i.e., setup work performed after the previous coil i and before j starts. Since the start time of j is not specified by S, but rather follows from it, this has no impact on S. Sample runs, on the other hand, bind all work resources available and are always performed directly before the actual



(a) Change tank when color changes.

(b) Improved schedule due to keeping the color.

(c) Keep color that is first needed again.

(d) Improved schedule due to concurrent setup work.

Fig. 4. Suboptimal canonical tank assignment rules.

coil j they are for. Thus, unless a global sample run is specified by S right before j already, local sample runs definitely delay the start of the subsequent coil by their length, which as above is not specified by S. Thus, we will call tasks in W stemming from scrap coils and local sample runs *trivial*.

In the next section, we introduce a model to simultaneously compute an optimal tank assignment and a schedule for all non-trivial setup tasks, assuming a given order of the coils.

3 A Graph Model for Optimal Tank Assignments

In this section we lay the theoretical ground for the design of our heuristic. Given a fixed sequence π of coils, we now aim to find a tank assignment T and concurrent setup schedule S leading to a coil coating (π, T, S) plan with minimal cost. We develop a representation of π and all possible tank assignments with feasible concurrent setup schedules (T, S) as a 2-union graph [3]:

Definition 1. *A graph $G = (V, E)$ is called a 2-union graph, if its nodes $v \in V$ can be associated with axis-parallel rectangles R_v positioned in the plane, such that $(v, w) \in E$ if and only if the projections of R_v and R_w on one of the coordinate axes intersect.*

In our model, an optimal pair (T, S) will correspond exactly to a maximum weight independent set (MWIS) in the corresponding graph. The MWIS prob-

lem is APX-hard for 2-union graphs [3], and we will show later, that it is still strongly NP-hard on 2-union graphs of the special structure resulting from the construction described in this section. Still, in Sect. 4.2 we will describe a dynamic programming approach which is fixed-parameter tractable in k, the number of coaters.

We model every possible assignment of a maximal subsequence of coils to the same tank and every resulting possibility to perform concurrent setup work as weighted intervals called *coater intervals* and *work intervals*, respectively. Then, we will combine coater and work intervals to form rectangles, i.e., the nodes of G.

The weights of intervals represent the reduction in cost (which may be negative) achieved by the corresponding tank assignment and the concurrent setup work performed, compared to processing all coils on the same tank without performing concurrent setup work, which would incur cost $c_{\text{init}} = \sum_{i=1}^{n-1} s(i,\, i,\, i{+}1)$. Work intervals of all coaters compete for the one available resource for concurrent setup work, i.e., a chosen work interval for one coater impacts the choice of work intervals on other coaters. In contrast, we can choose coater intervals of different coaters independently. We will position the rectangles in the plane accordingly, such that there is a one-to-one correspondence between tank assignments with concurrent setup work schedules and maximal independent sets of rectangles.

We denote by p_i the processing time of coil i, and throughout this section, we assume that the coils are numbered as they appear in the fixed sequence, i.e., $\pi = \mathrm{id} \in \Pi_n$.

3.1 Coater Intervals

Coater intervals associated with a particular coater represent maximal sequences of coils that are processed on the same tank. So before and after it, we switch tanks. Hence, a set of non-intersecting coater intervals covering every coil defines a unique tank assignment.

We denote an interval which contains coils $i, j \in [n]$, $i \le j$, and all intermediate coils by $I = [i, j]$. The length of I is $|I| := \sum_{k=i}^{j} p_k$. The last coil before and the first coil after the interval I is referred to as *predecessor* $\mathsf{pred}(I)$ and *successor* $\mathsf{succ}(I)$, respectively. The last coil in interval I is denoted by $\mathsf{last}(I)$.

We define the weight $w_{\text{coat}}(I)$ of a coater interval I as the cost savings before coil $\mathsf{succ}(I)$ resulting from keeping color and roller in the idle tank between $\mathsf{pred}(I)$ and $\mathsf{succ}(I)$ as compared to running $\mathsf{succ}(I)$ on the same tank as $\mathsf{last}(I)$. Savings resulting from concurrent setup work during production will be assigned to work intervals as described in the next section. Thus, w_{coat} computes as

$$w_{\text{coat}}(I) \;=\; c\big(\mathsf{last}(I),\, \mathsf{last}(I),\, \mathsf{succ}(I)\big) - c\big(\mathsf{last}(I),\, \mathsf{pred}(I),\, \mathsf{succ}(I)\big).$$

Note that some coater intervals with negative savings may have to be selected in the definition of a tank assignment due to the requirement that all of π be covered.

3.2 Work Intervals

Every work interval is associated with a coater interval $I = [i, j]$. It represents concurrent setup work performed for $\mathsf{succ}(I)$ on this coater's idle tank during coils $k = i, \ldots, j$. Since concurrent setup work must not be preempted by other concurrent setup, partial concurrent setup work is only allowed directly before $\mathsf{succ}(I)$, i.e., at the end of the corresponding coater interval—it is completed during non-productive time in this case. Hence, there are at most two work intervals for a coater interval, one for color and one for roller change. The sum of their lengths must not exceed $|I|$.

If $s_{cc}^{(m)}(\mathsf{pred}(I), \mathsf{succ}(I)) > 0$ for a coater m, cost may be saved by adding a work interval of length $\lambda\, t_c$ to I. This interval may start at an arbitrary point in I and may only have shorter length if starting later than $\lambda\, t_c$ before the end of I. We proceed analogously for roller changes.

Due to varying start times, there may be many work intervals corresponding to a particular coater interval I. Fortunately, we do not need to consider all of them in the model. It is sufficient to consider start times of the form $\sum_{k=0}^{\ell} p_k + q\,\lambda\, t_c$ for some $\ell \leq i$ and $q \in \mathbb{N}$, ignoring non-productive time. In Sect. 4.2 we will show that the number of work intervals needed in the model to guarantee an optimal solution is polynomially bounded in the n.

We now define *saving rectangles* $R = (I,\, I_\mathrm{c},\, I_\mathrm{r})$ as a combination of a coater interval I and two (possibly empty) work intervals I_c and I_r for color and roller changes. By construction, the cost savings w_work due to these work intervals is

$$w_\mathrm{work}(I_\mathrm{c},\, I_\mathrm{r}) \;=\; 1/\lambda\,(|I_\mathrm{c}| + |I_\mathrm{r}|).$$

Thus, the total weight of a saving rectangle R is

$$w(R) \;=\; w_\mathrm{coat}(I) + w_\mathrm{work}(I_\mathrm{c},\, I_\mathrm{r}).$$

This weight may be negative. Also note that we use the term rectangle in a slightly generalized way, since up to two work intervals may be associated with each coater interval. As will become clear in Sect. 4.2 however, this poses no problem for our algorithmic ideas.

3.3 Positioning Rectangles

We say that two saving rectangles *conflict* if their work intervals intersect, or if their coater intervals belong to the same coater and intersect. Coater intervals whose intersection contains a coil i lead to a conflicting tank usage, since both intervals assign i to a different tank. Intersecting work intervals violate the constraint that only one work resource is available.

Consequently, we position rectangles in the x-y-plane as follows: Each coater's rectangles have their own section of the x-axis, namely $(m-1)\,n$ is added to the x-coordinate of the mth coater's rectangles, while all y-coordinates remain unchanged. Now, the savings corresponding to two rectangles R_v, R_w conflict if

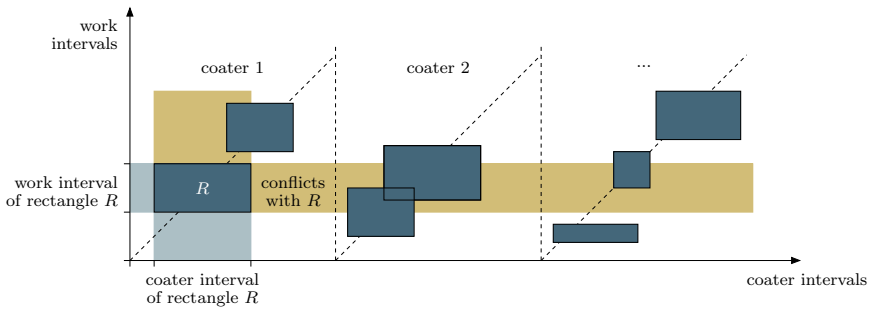


Fig. 5. Correspondence of conflicting saving rectangles and adjacent nodes in the 2-union graph.

and only if R_v's and R_w's projections on the x-axis or on the y-axis, corresponding to their coater intervals or their work intervals, respectively, intersect. This is exactly the case if nodes v and w in G share an edge, see Fig. 5.

By the construction above, the rectangles have a specific structure.

Definition 2. *A 2-union graph is called* diagonally k-partitionable*, if it has a rectangle representation in which one of the axes, say the x-axis, can be subdivided into k sections such that*

- *the projections of all rectangles onto the x-axis are contained completely in one of the sections*
- *the affine line through origin for each of the sections goes through both sides parallel to the x-axis of each rectangle in that section.*

In our model, the coater intervals for different coaters define k sections, and since work intervals are always contained in coater intervals, the second property follows as well.

Diagonally k-partitionable 2-union graphs have the following nice property, which follows immediately from their definition:

Proposition 1. *When the projections on the y-axis of two rectangles belonging to the same section intersect, their projections on the x-axis intersect as well.*

We will exploit this fact in Sect. 4.2 to devise a dynamic programming algorithm for the MWIS problem on these graphs, which runs in polynomial time if we assume k to be fixed in advance.

For the case that k is part of the input, we can prove the following result, suggesting that our dynamic programming approach is the best possible.

Theorem 1. *The maximum independent set problem on diagonally k-partitionable 2-union graphs is strongly* NP*-hard, even in the unweighted case.*

Proof. We give a reduction from 3-SAT [4]. Let I denote an arbitrary instance of it with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. We may assume w.l.o.g.,

that no clause contains two literals of the same variable. With $k = n$, we now construct a k-composite 2-union graph G, such that G admits an independent set of cardinality $n+m$ if and only if there is a truth assignment for the variables of I such that all clauses are satisfied.

We will construct G in terms as the edgewise union of two simple interval graphs G_1 and G_2, or equivalently, as a non-homogeneous 2-interval representation where the rectangles' latitudes in the two dimensions are given by the corresponding intervals in the two interval graphs.

We introduce a node for each of I's $2n$ literals. The intervals for the nodes in G_1 and G_2 are given in Tab. 1.

Literal	Interval in G_1	Interval in G_2
x_i	$[(i-1)\cdot 2m; (i-\frac{1}{2})\cdot 2m)$	$[i-1; i)$
$\overline{x}_i$	$[(i-\frac{1}{2})\cdot 2m; i\cdot 2m)$	$[i-1; i)$

Table 1. Intervals associated with literals in the two simple interval graphs G_1 and G_2.

Thereby, the intervals belonging to literals of different variables are always disjoint in both G_1 and G_2, while the two literals of one variable share an edge in G_2 (see Fig. 6).

Furthermore, we introduce a node for each occurrence of a literal in a clause. Let y denote a literal in c_j and s the start point of the interval associated with y's negation in G_1 as in Tab. 1. These nodes' intervals in G_1 and G_2 are defined in Tab. 2.

Occurrence in clause	Interval in G_1	Interval in G_2
c_j	$[s+(j-1)/m; s+j/m)$	$[n+j-1; n+j)$

Table 2. Intervals associated with occurrences of literals in clauses in the two simple interval graphs G_1 and G_2 where s denotes the start point of the interval in G_1 associated with the negation of the occurring literal according to Tab. 1.

Now all occurrences of literals in the same clause are adjacent in G_2, while occurrences of literals in different clauses are always independent in both G_1 and G_2 (again see Fig. 6).

Now suppose the graph G, i.e., the edgewise union of G_1 and G_2, admits an independent set S of cardinality at least $n + m$. Since the two literals of each variable are adjacent, S may contain at most n nodes corresponding to literals. On the other hand, the three occurrences of literals in each clause form

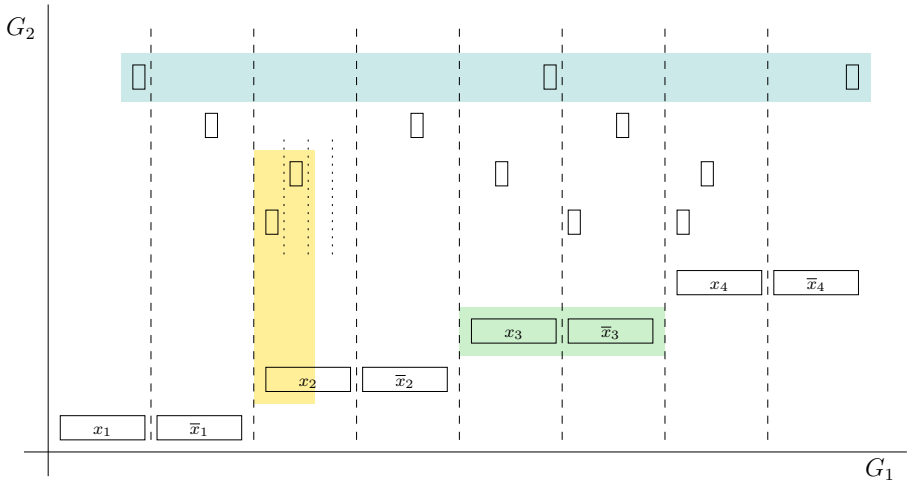


Fig. 6. Non-homogeneous 2-interval representation of the graph constructed from the 3-SAT instance with four variables and clauses $(\overline{x}_2 \vee x_3 \vee \overline{x}_4)$, $(\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)$, $(x_1 \vee x_2 \vee x_3)$, $(\overline{x}_1 \vee \overline{x}_3 \vee x_4)$.
The 2-intervals associated with the two literals of variable x_3 are highlighted green, those belonging to the fourth clause $(\overline{x}_1 \vee \overline{x}_3 \vee x_4)$ blue. The adjacency of the occurrences of literal $\overline{x}_2$ in the first two clauses with x_2 is highlighted golden.

a triangle, so similarly, S may contain at most m of these nodes. Consequently, S contains exactly one literal of each variable and one occurrence of a literal for each clause. Let us interpret the choice of literals in S as a truth assignment X for the variables of I. Each occurrence of a literal in a clause is adjacent to its negation in G, so such occurrence being in S implies it is true in X, for S is an independent set. Hence, X fulfills at least one literal in each clause of I, which is consequently a yes-instance.

Conversely, suppose I is a yes instance and X is a truth assignment fulfilling all clauses. Then an independent set in G can be constructed in the same way, picking n nodes corresponding to the literals in X and an occurrence of a literal true in X for each clause.

Furthermore, for $k = 2n$, G is clearly k-composite: No interval in G_1 intersects the end points of the intervals corresponding to literals, so these intervals define k connected components of G_1. Finally, also by construction, the intervals in G_2 of no two nodes in such component intersect, since every clause of I was assumed to contain at most one literal of each variable.

3.4 Building Schedules from Maximal Independent Sets

First note that by adding a sufficiently large constant to the weight of each rectangle weighted by its length, we can force each MWIS to be maximal, i.e.,

the selected rectangles to have coater intervals covering all coils, even if some rectangles have negative weight.

Given an independent set of saving rectangles such that every coil is covered by a coater interval on every coater, a feasible tank assignment and concurrent setup schedule can be constructed as follows. At the end of each coater interval, we switch tanks on the corresponding coater. The schedule for concurrent setup work is given directly by the work intervals. For performing setup work during non-productive time, and for adding scrap coils and sample runs, we insert sufficiently long breaks in between coils. By construction, there is a one-to-one correspondence between the cost of this schedule and the cost of the chosen saving intervals $\mathcal{R}$, i.e.,

$$c(\mathcal{R}) \;=\; c_{\text{init}} \;-\; \sum_{R \in \mathcal{R}} w(R)\,.$$

Note that in the case where there is a separate work resource available for each coater, i.e., concurrent setup can be performed simultaneously on all coaters, two saving rectangles conflict if and only if they belong to the same coater and their coater intervals intersect. Thus, an optimal tank assignment and concurrent setup schedule can be computed independently for each coater and each graph constructed is an interval graph. Maximum weight independent sets in interval graphs can be computed very efficiently [5], hence our model yields a fast algorithm for the tank assignment and concurrent setup scheduling problem when sufficient work resources are available.

Also note that for the sake of clarity, we have omitted the case where coils require no color in one or more coating stages in the preceding discussion. This case can be incorporated into the model seamlessly by adapting some definitions of interval weights and allowing coater intervals to intersect on colorless coils. Similarly, for an exact definition of the cost savings due to coater and work intervals, local setup directly after these intervals needs to be subtracted from their savings, since setup work may be performed during scrap coils.

4 Algorithm

We will now describe how we produce high quality, fully detailed production plans for the entire coil coating problem. Because of the enormous complexity of the task, an optimal solution is currently out of reach. On top of that, quick computation times are indispensable for the practical usability of our algorithms: A plan covering 24 hours must be computed within 180 seconds. This is why "intelligent" heuristics are our design of choice. We would like to make the point that these heuristics could never have developed without the insights gained in the previous sections. Moreover, as we detail in Sect. 5, we dedicate considerable effort to evaluating the quality of our approach with the help of lower bounds on the optimum makespan. This will demonstrate that our heuristics actually perform very well.

Mainly because of global setup cost we cannot base an algorithm on local decisions only. Instead, meta heuristics which generate many *complete* solutions quickly are a natural approach. We utilize a genetic algorithm, which, by its nature, combines solutions in such a way that beneficial characteristics of solutions found persist, while costly characteristics are eliminated. Since we need to evaluate the cost of many sequences, tank assignment and scheduling need to be performed very often as well. Hence, fast algorithms for these subproblems are required. We describe these ingredients in the following subsections.

4.1 Sequencing

We use a classical genetic algorithm (see [6] for a general overview) for generating sequences $\pi \in \Pi_n$, exploiting our special cost structure for choosing the initial population. We only state its basic ideas and describe the construction of the initial population. See also [7] for a recent successful application to a different production sequencing problem.

Maintaining a constant population size, we perform mutation and crossover operations that are common for sequencing problems. We use the mutation operation INVERT, which inverts the order of a randomly chosen subsequence, and crossover operations proposed by Mühlenbein [8]. Further details on the parameter settings are specified in Sect. 6.

Concerning the initial population, practical experience has shown that not so much the absolute quality of the solutions used, but much more their diversity w.r.t. different beneficial aspects lead to rapidly evolving individuals. Also, choosing the initial population "too well" can lead a genetic algorithm to approach local minima quickly, but to end up with suboptimal solutions after all.

Motivated by the diversity of coil properties which may have an impact on local and global cost (like width, height, or oven temperatures), we generate sequences for the initial population, each trying to be good at only one (or a few) of these properties at once. The considered properties P can all be described as functions from $[n]$ to $\mathbb{R}$. For each property $p \in P$, there is a restriction r_p which maps the property p of two coils to the setup time between them which is caused by this restriction. If a property value for subsequent coils is monotone, or respects a certain maximal gap, then the corresponding restriction does not lead to cost. We call pairs of coils with this property *feasible*. When sorting the coils with respect to a certain property, the corresponding restriction never (or rarely) leads to cost.

Using this idea, we pick as parameters to construct an initial individual a list σ of u *sorting properties* σ_i, and a small set ϕ of *feasibility properties*. Now, we sort the coils lexicographically by their properties in $(\sigma_1, \ldots, \sigma_u)$ to obtain an order π_σ. Then, we develop a sequence $\pi_{\sigma\phi}$ by adding the first coil in π_σ and then greedily appending coils in the order of π_σ which are feasible with respect to all restrictions regarding properties in ϕ. When we reach the last coil, we add the first coil in π_σ which has not yet been added to $\pi_{\sigma\phi}$ and repeat.

Repeating this process for different σ and ϕ, we obtain a broad variety of completely different individuals, each attractive regarding its own part of the

objective, which provide a particularly diverse basis for constructing sequences with low overall cost by simple crossover and mutation operations. Compared to choosing only random sequences or heuristics attempting to cover all aspects of our cost function, this initial population leads to better solutions and to a shorter runtime of the algorithm.

Furthermore, we add one sequence which is (close to) optimal w.r.t. local cost $s_{\text{loc}} + t_{\text{loc}}$, but which ignores all global setups. Determining such a sequence can be formulated as an asymmetric traveling salesman problem which we (typically optimally) solve using the LKH heuristic [9].

4.2 Tank Assignment

We know that a good tank assignment aims to satisfy two, in a sense opposite, requirements: On the one hand, try to keep a roller and/or color for a subsequent coil in order to avoid setup whenever possible. On the other hand, try to use idle times on the tanks to perform setup work during regular production which otherwise might increase the makespan. Hence, try to maximize concurrent setup work that is accomplished by the scarce work resource. Clearly, the quality of a tank assignment w.r.t. the second goal can hardly be assessed without scheduling the scarce work resource to perform concurrent setup whenever made possible by the tank assignment.

We will first describe, how an optimal tank assignment and concurrent setup schedule can be computed by a dynamic programming approach, which is fixed-parameter tractable in the number of coaters considered.To satisfy the strict runtime requirements of the application, we will then devise a fast tank assignment heuristic based on ideas from the former exact algorithm, and state a simple and fast rule to find a schedule for such tank assignment. Finally, we will describe an intuitive tank assignment rule which is in use at SZFG today, and to which we will compare our results inSect. 6.2.

Optimal Tank Assignment Based on the Interval Model Assume the number of coaters k is fixed before hand. The runtime of our algorithm will be polynomial in the number of coils n, but exponential in the fixed parameter k. The core of the algorithm is a dynamic programming scheme. We define a state of a solution as a $(k+1)$-tuple $R := (i, R_1, \ldots, R_k)$, where $i \in [n]$ signifies the i-th coil in the sequence and $R_m = (I_m, I_c^{(m)}, I_r^{(m)})$ a saving rectangle as in Sect. 3.2, where I_m either starts at or properly contains i, i.e., R_m is a certain combination of a coater interval with two (possibly empty) work intervals. Since at each point in the sequence, exactly one saving rectangle is selected for each coater, there are no more than $n \cdot N^k$ states, where N denotes the total number of potential saving rectangles.

We call a state R *admissible*, if in the set $W_R := \bigcup_{m\in[k]}\{I_c^{(m)}, I_r^{(m)}\}$ of all work intervals of all coaters in R, no two intervals intersect. We call two admissible states $R = (i, R_1, \ldots, R_k)$, $Q = (i+1, Q_1, \ldots, Q_k)$ *compatible*, if for each $m \in [k]$ either

- the saving rectangles R_m and Q_m are equal or
- Q_m starts with the coil right after R_m, and in the set $W_R \cup W_Q$, no two intervals intersect.

We call a sequence of admissible states, where subsequent states are compatible, an *admissible sequence*. When such sequence is viewed as a selection $\mathcal{R}$ of saving rectangles, it corresponds to a tank assignment with a feasible schedule of cost $c(\mathcal{R})$ as described in Sect. 3.4.

Furthermore, we have the following optimal substructure: When $\mathcal{R}$ is an optimal admissible (sub)sequence, then the admissible subsequence obtained by removing the last state Q from $\mathcal{R}$ is optimal among all admissible subsequences which end in a state R such that Q is compatible with R.

This defines a straight forward algorithm with runtime $\mathcal{O}(n^2 \cdot N^{2k})$, computing an optimal tank assignment and concurrent setup schedule: we generate the graph of all states and two additional nodes s and t. We connect states R and Q if and only if they are compatible, s to all states associated with coil 1, and all states for coil n to t. The edges have weights corresponding to the cost savings of the states they lead to, and zero if they lead to t. Due to the optimal substructure of the states, it now suffices to compute a longest s-t- path in this acyclic graph. For the tractability of the algorithm, it remains to show, that N is polynomial in n, which is the case as long as the ratio between runtimes of coils and λt_c is polynomial:

Lemma 1. *There is a set D of at most*

$$|D| = \frac{\max_{i\in[n]} p_i}{\lambda\, t_c} \cdot \frac{(n-1)(n-2)}{2}$$

points in time such that there exists an optimal selection of saving rectangles $\mathcal{R}$, whose work intervals all have end points in D.

Proof. Work intervals may start with every but the last coil in the sequence, accounting for the first $n-1$ points necessary. In addition, given a potential start point for a work interval, also the point exactly $\lambda\, t_c$ after it is a potential start point. Clearly, any optimal selection of saving rectangles can always be transformed to a solution where all work intervals start at a point specified above by shifting every selected work intervals to the left until its start point coincides with a point of the above form.

Assuming w.l.o.g. that the fixed sequence of coils is $\pi = \mathrm{id} \in \Pi_n$, observe that moving through the sequence in order, the number of new potential starting points during the processing of coil i is bounded by $i(p_i/\lambda t_c)$. Thus, the total number of potential start points for work intervals necessary in an optimal solution is

$$\sum_{i\in[n-1]} i\,\frac{p_i}{\lambda t_c} \le \frac{\max_{i\in[n]} p_i}{\lambda\, t_c} \cdot \frac{(n-1)(n-2)}{2}.$$

Thereby, the number of savings rectangles which need to be considered for an optimal plan in bounded by

$$N \leq n^2 + n^2|D| + n^2|D|^2 \leq n^2(|D| + 1)^2,$$

the sum of all possibilities to pair each coater interval with no, one, or two work intervals. Note that N is polynomial in n as long as the ratio $(\max_{i\in[n]} p_i/\lambda\, t_c)$ is bounded polynomially in n. In realistic instances, the maximum length of coils is roughly equal to the time for a single setup task performed concurrently, so this ratio is roughly one. This bound is also very coarse, since it takes into account all possibilities to pair *any* work interval with each coater interval, while only work intervals starting within the coater interval are really relevant. Theoretically, however, extreme cases with many coils with extremely short processing time followed by one very long coil cannot be ruled out.

Heuristic Based on Interval Model The dynamic programming approach is obviously not reasonable for practical use due to its runtime. Nevertheless, we can use a similar idea to define a heuristic to compute a good tank assignment. The scheduling can then be performed by a simple rule described in Sect. 4.3, which yields a concurrent setup schedule for a given tank assignment.

The complexity of the exact algorithm stems from the need to consider interval selections for all coaters at once in order to ensure, that savings from work intervals can actually be realized by the scarce work resource within the associated coater interval. So intuitively, the possibility that potential cost savings from work intervals can actually be realized by a feasible schedule for concurrent setup across all coaters, increases with the length of the associated coater interval. This intuition is the core idea for the following tank assignment heuristic.

Instead of considering all coaters at once as the exact algorithm, we consider all coaters separately, and hence, ignore that parallel work on different coaters is not possible. Now the cost savings from selected coater intervals can still always be realized, but the realization of cost savings from work intervals could conflict with work intervals corresponding of other coaters. Thus, we usually overestimate the actual cost saving. To readjust this behavior, we scale the interval length, i.e., the potential time for concurrent setup work, by a factor $\alpha \in [0, 1]$. When choosing $\alpha = 0$, concurrent setup work is assumed impossible. For the contrary extreme $\alpha = 1$, the entire intervals are assumed to be available for work. In the heuristic, we assume that we perform as much concurrent setup work in the scaled interval as possible, and add the corresponding cost savings to the savings of the coater interval.

Now, cost savings of intervals are independent of actual work intervals, so it suffices to consider coater intervals only. As a consequence, similar to the case of sufficient work resources mentioned in Sect. 3.4, our problem reduces to finding a maximum weight independent set in a regular interval graph, which can be dealt with very efficiently [5], yielding a fast heuristic. Finally, this results in a tank assignment as in Section 3.4, for which we compute a concurrent setup schedule as described in Section 4.3.

Online Rule: First-in First-out (FIFO) Planners at SZFG have previously used the following simple rule for tank assignment, which only regards the colors of subsequent coils: Whenever subsequent coils have different colors, switch the tank. If the new tank does not contain the required color, a color change on that tank becomes necessary. So whenever a third color besides the two in the shuttle tanks is required, the color which was in use earlier is discarded, i.e., we follow a first-in-first-out (FIFO) rule.

The advantage of this FIFO rule is its online character: the choice of tank depends only on the current and the previous coil. We exploit this fact in Sect. 5, where we devise an integer programming model to compute lower bounds on the optimal makespan of coil coating plans, assuming the FIFO rule is used for tank assignment.In fact, planners at SZFG initially insisted this rule was also used in our algorithm, assuming it yielded the best possible tank assignment in any case. After observing the savings potential of our more sophisticated tank assignment mechanisms in some of our solutions, they abandoned this initial requirement.

Improvements through Local Search Heuristics Due to the fact that the tank assignment heuristics described above relax the constraints on work resources for concurrent setup, the tank assignments computed may be suboptimal in the sense, that the need for setup work on different shuttle coaters occurs within the same short part of the sequence. Consequently, a large part of this setup work cannot be scheduled concurrently and causes cost.

In many cases, this excessive stress on the work resource can be evened out by switching the tank assignment on some shuttle coaters for a certain subsequence of coils. Such possibilities for improvement can be found by a local search heuristic on the tank assignment. Local search, however, requires too much runtime to be performed during the sequencing process. In fact, local search in any neighborhood of size quadratic in the number of coils turned out to be too time-consuming for our purposes, even if only run on the final solution computed by the sequencing algorithm.

We use the following linear-sized Twist-neighborhood for a local search on the tank assignment of the final solution: For any coil i in the sequence, flip the tank assignment for all coils occurring after i in π. We report on the results obtained in Sect. 6.2.

4.3 Concurrent Setup Scheduling

Once a sequence and a tank assignment have been determined, the coil coating plan may still be lacking a feasible schedule for concurrent setup work. A solution to the dynamic program described in Sect. 4.2 already contains a non-intersecting selection of work intervals, so we do not need to perform this step in that case—an optimal schedule is given by performing concurrent setup according exactly to the selected work intervals as described in Sect. 3.4.

If one of the heuristics for tank assignment described in Sect. 4.2 and 4.2 was used, we now have a sequence and a tank assignment at hand, but still need to perform scheduling in order to even assess the cost of the coating plan.

Obviously, the dynamic program could be used to compute an optimal schedule for the given tank assignment, by only specifying exactly the coater intervals corresponding to it, together with all necessary work intervals (cf. Lem. 1). For practical use, however, this approach is not reasonable, again due to its runtime. Thus, in our implementation, we use an earliest-deadline-first strategy as a simple scheduling rule.

As before, each global setup task $w \in W$ is associated with the coil i it is for. We define a release time and a deadline for w as follows: Job w is released at r_w when the tank on which i is run becomes idle for the last time before i; its deadline d_w is the start time of coil i. Since only one concurrent setup task may be performed at a time, we are trying to schedule all tasks on a single work resource. Whenever the work resource becomes available, say at time t, we schedule the setup task w with the earliest deadline for which $t \in [r_w, d_w]$.

This strategy satisfies tight practical runtime requirements and appears to perform well for our application, even though it is in general not guaranteed to lead to optimal concurrent setup schedules.

5 Lower Bounds from a Combinatorial Relaxation

Assessing the quality of heuristic solutions is not only a theoretical contribution, but an important question in practice: How much optimization potential is left? In this section we describe a general way of computing an instance-dependent lower bound on the optimal makespan, when an online tank assignment rule is used (cf. Sect. 4.2). Obtaining good lower bounds is strongly related to devising good relaxations of the problem at hand. Ignoring the need for setups altogether we obtain the trivial lower bound as the sum of processing times of all coils, $LB_{\text{triv}} := \sum_{j=1}^{n} p_j$. A more elaborate idea is to relax the complicating global setup costs only. In fact, this reduces the coil coating problem to determining an optimal sequence with respect to local setup costs—which can be formulated as a (small) asymmetric traveling salesman problem, thus we denote the obtained bound by LB_{TSP}.

As we stressed several times, global setup cost for a coil j depends—in the extreme case—on the entire solution, in particular on the entire tank assignment, prior to running coil j. Our relaxation now limits this dependency in limiting the number of coils which are considered when computing global setup cost, i.e., we "don't look back too far." More precisely, we concatenate subsequences containing a constant number of coils, say k, for which we exactly compute the global cost. In between subsequences we only consider local setup cost. By means of an integer linear program we find a cheapest such concatenation among all possible combinations. A solution is a sequence of all coils, and the tank assignment will be given implicitly.

5.1 An Integer Program: Concatenating Short Subsequences

The logic of the model is to assign subsequences of k coils each to $\lceil n/k \rceil$ *time slots*, each consisting of k consecutive positions, except the last which may possi-

bly be shorter. For a subsequence s, we denote by $t(s) \in \{1, \dots, \lceil n/k \rceil\}$ its time slot, and by $p(s,j) \in \{1, \dots, n\}$ the absolute position of coil j in subsequence s. We subsume all possible subsequences of k coils in the set $\mathcal{S}$. We slightly abuse the set notation $j \in s$ to express that coil j is contained in subsequence s. Naturally, the cardinality of $\mathcal{S}$ is exponential in n, and listing $\mathcal{S}$ explicitly in an integer program is out of the question. The general idea is to solve the linear relaxation of our model by dynamically adding sequences, a.k.a. column generation, and embedding this into a branch-and-price framework [10,11].

We have binary variables $x_{p,j}$ for deciding whether coil j is assigned to position p or not. Variable $z_s \in \{0,1\}$ indicates whether subsequence $s \in \mathcal{S}$ is selected or not. Each $s \in \mathcal{S}$ has its local plus global setup cost c_s which is computed as if all coaters were entirely clean and empty at the beginning of s. The local setup cost $s_{\text{loc}}(i,j) + t_{\text{loc}}(i,j)$ between coils i and j is abbreviated by $c_{i,j}$.

Note that in order to compute global setup cost, we need to know a tank assignment which we do not explicitly compute. For the optimal value of this model to deliver a guaranteed lower bound, it is essential that the tank assignment rule assumed within subsequences can be concatenated to a tank assignment on the whole sequence following the same rule. It is exactly the set of *online* assignment rules (cf. Sect. 4.2), which possess this property. Hence, we use the FIFO rule in the integer program and thereby obtain lower bounds for the case when the FIFO rule is used.

$$\min \sum_{s} c_s z_s + \sum_{i,j \in [n]} c_{i,j} y_{i,j} \tag{1}$$

$$\sum_{p \in [n]} x_{p,j} = 1 \qquad j \in [n] \tag{2}$$

$$\sum_{j \in [n]} x_{p,j} = 1 \qquad p \in [n] \tag{3}$$

$$\sum_{\substack{s \ni j: \\ p(s,j)=p}} z_s = x_{p,j} \qquad p,j \in [n] \tag{4}$$

$$x_{\text{last}(t),i} + x_{\text{first}(t+1),j} \leq 1 + y_{i,j} \qquad i,j \in [n],\ t = 1, \dots, \lceil n/k \rceil - 1 \tag{5}$$

$$x_{p,j} \in \{0,1\} \qquad p,j \in [n] \tag{6}$$

$$y_{i,j} \in \{0,1\} \qquad i,j \in [n] \tag{7}$$

$$z_s \in \{0,1\} \qquad s \in \mathcal{S} \tag{8}$$

The constraints are interpreted as follows. Each coil gets coated exactly once (2), each position is filled exactly once (3), and a subsequence s needs to have coil j in position p if and only if the corresponding $x_{p,j}$ indicates so (4). The precedence constraints (5) enforce that in between two consecutive subsequences in time slots t and $t+1$ we incur the local setup cost between coil i in the last position $\text{last}(t)$ in t and coil j in the first position $\text{first}(t+1)$ in $t+1$. The binary variable $y_{i,j}$ precisely takes care of that, as its use is penalized in

the objective function (1) accordingly. The optimal objective value of the integer program is denoted LB_{IP}.

We may alternatively use variables $x_{p,j,ta} \in \{0,1\}$ which additionally decide about which tank to use on every coater in each position of the sequence. The index ta reflects the different combinations on all coaters (in our case eight essentially different tank assignments since we have three shuttle coaters). This way, an optimal tank assignment can simultaneously be computed for each subsequence, obtaining a lower bound on the makespan in any case. We did not further follow this idea, for it would unduly increase the computational effort necessary to obtain optimal solutions to the model.

5.2 Pricing

Initially, the integer program (1)–(8) contains all $x_{p,j}$ and $y_{i,j}$ variables, but only some z_s variables, namely those which correspond to subsequences derived from a solution produced by our sequencing algorithm (it is restricted to use the FIFO tank assignment rule in that case). Its linear relaxation is called the restricted master problem. All other z_s variables are generated only as needed. The coupling constraints (4) are the only ones which contain these variables; as a consequence the corresponding dual variables $\mu_{p,j} \in \mathbf{R}$ are the only relevant ones for calculating their reduced cost (which must be negative in order to add the variable to the problem). The reduced cost of z_s is

$$\bar{c_s} = c_s - \sum_{j \in s} \mu_{p(s,j),j} \quad , \tag{9}$$

and the pricing problem is to find a subsequence of minimum reduced cost. To the best of our understanding, there is no principal alternative to a brute force method since we are able to evaluate the total setup cost of a subsequence only when we see it in its entirety, thus we have to construct it. This also rules out most of the traditionally used dominance criteria in dynamic programming. Thus, a straight forward depth first search backtracking algorithm is used to solve this problem. Yet, we use a pruning criterion based on the dual variable values of coils not yet added to a subsequence, which helps in mildly reducing the search space without compromising optimality.

5.3 Branching

When we determine an integer solution for the $x_{p,j}$ variables, all other variables automatically assume integer values as well. That is, a natural candidate for taking branching decisions in the branch-and-bound tree is to branch on

$$\sum_{s \ni j : p(s,j)=p} z_s \notin \{0,1\}$$

for a given coil j at position p. In fact, speaking in terms of branch-and-price methodology, this is branching on the so-called original variables $x_{p,j}$ of the

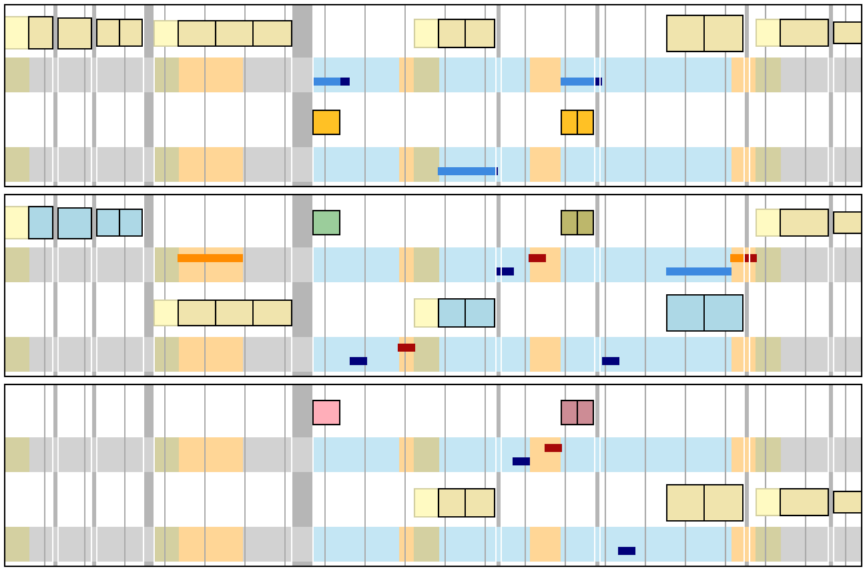


Fig. 7. Detail of the visualization of an integer optimal solution to our integer program with subsequences with $k = 4$ coils each. Each block represents one shuttle coater with its two tanks; the dark rectangles reflect the coils in the coating sequence. It is clearly visible that the integer program ignores global setup cost in between subsequences. The expert planner can also see where and when which type of setup has to be performed.

problem which are present in this extensive column generation as well. The branching itself can be realized as a modification to the pricing problem instead of adding an explicit branching constraint to the master problem: One simply eliminates the forbidden coil j from position p (on the down branch), or enforces it by eliminating all other $[n] \setminus \{j\}$ coils from position p (on the up branch). Note that the master problem has to be updated according to branching decisions: Variables corresponding to subsequences which do not respect the branching constraint have to be eliminated (technically, via an upper bound of zero). In order to stay feasible after each branching decision, we further introduce artificial variables (with large costs) in constraints (2), one for each coil.

6 Computational Study

Since the project was initiated with the explicit goal to develop a tool to be integrated in the existing production planning software at SZFG, we were provided with realistic data right from the beginning. In fact, we tested our algorithms on instances provided by planners at SZFG, and in the following process of their repeatedly validating our solutions, various issues regarding the accuracy

of cost calculations and the practicability of work schedules were addressed, and eventually solved.

A typical batch for a 24 hour planning horizon consists of 20–40 coils, but for testing purposes we also considered batches which were three times larger. We cannot report on the precise numerical characteristics of our data and results; instead a qualitative assessment of our approach is presented.

6.1 Some Implementation Details

Parameters for the genetic sequencing algorithm were determined by rigorous testing with the goal to have one fixed parameter set yielding good performance across all data sets. We use a constant population size of 200 and in each generation create 100 new individuals each from mutation of random individuals and crossover of two randomly chosen individuals, resulting in a total population of 400. We keep the 200 with the best makespan for the next generation. Especially on small instances, the population quickly evolves to a set of individuals with almost identical cost. When the makespan of all individuals is within 5% of the best individual, we discard 90% of the population and replace it by individuals generated as the initial population. This usually leads to further improvement of the best individuals in subsequent generations.

In the construction heuristic for the initial population described in Sect. 4.1, we use as sorting and feasibility sets the coil properties width, height, processing speed, and temperature in primer and finish oven, respectively. We compute sequences resulting from each combination of two sorting and one feasibility property.

We also performed extensive testing using the tank assignment heuristic described in Sect. 4.2 for various values of the parameter $\alpha \in [0, 1]$ when computing a tank assignment for the evaluation of the makespan of a sequence. The best results were obtained for $\alpha \approx 0.5$ and were equal to the makespan obtained using the FIFO rule in most cases, insignificantly better on some instances. Similarly, local search on tank assignments proved by far too time consuming to be performed within the sequencing process and did not yield significantly better final results—minor improvements in makespan were observed on some instances when it was performed on some individuals of the final population in the sequencing step.

While the possibility to use local search and different tank assignment heuristics is preserved in the final optimization module, we refrain from reporting detailed computational results for them here, since they do not seem to yield conclusive evidence of their superiority over FIFO on the data sets considered—this may be different in other data scenarios.

We implemented a branch-and-price algorithm to solve the integer program (1)–(8) within the publicly available SCIP framework [12]. The implementation is not tuned to performance as we used it as a proof-of-concept only.

instance	LB_{triv}	gap%	LB_{TSP}	gap%	LB_{IP}	gap%	genetic	reference
2008-09-18_1	0.73	36.87	0.85	17.77	0.92	8.16	1.00	–
2008-09-19_1	0.86	16.87	0.87	15.11	0.95	5.18	1.00	1.19
2008-09-19_2	0.68	47.01	0.76	31.29	0.87	15.08	1.00	1.27
2008-09-22_1	0.81	23.08	0.84	18.96	0.90	11.51	1.00	1.35
2008-10-15_2	0.89	12.96	0.91	9.68	0.92	8.63	1.00	1.10
2008-10-15_3	0.88	13.10	0.92	8.37	0.93	7.44	1.00	1.09
2008-10-17_2	0.89	12.54	0.91	9.37	0.93	7.36	1.00	1.06
2008-10-28_1	0.92	8.25	0.96	4.65	0.96	4.28	1.00	–
2008-10-28_2	0.86	16.72	0.87	15.03	0.90	11.02	1.00	–

Table 3. Comparison of bounds on the makespan for representative problem instances, normalized to the makespan obtained by our genetic algorithm. We give the three lower bounds LB_{triv}, LB_{TSP}, and LB_{IP} (in that order), and the makespan of a reference solution provided by SZFG (not available for every instance). The columns headed "gap%" represent the remaining theoretical optimization potential when using the solution provided by our genetic algorithm, computed as $100 \cdot (\text{genetic} - LB)/LB$.

6.2 Interpretation of Results

It was surprising for SZFG's planners that our heuristic produces plans with makespan reductions of up to 30% (and typically 10%) as compared to reference solutions actually gone to production, cf.Tab. 3 and Fig. 8. Since this is also true for manual plans which "looked optimal," we take this as another proof that a rigorous mathematical analysis of the savings potential of shuttle coaters fully paid off.

A similar statement holds for our lower bounds based on combinatorial optimization problems which are clearly superior to a naïve approach, see again Fig. 8. If we concentrate on setup cost—in contrast to makespan—it can be seen from Fig. 9 that the integer programming lower bound is able to close (much) more of the gap to the upper bound than the TSP bound which ignores global setup cost. Yet, we did not solve all instances to integer optimality and also used short subsequences only ($k = 4$), so the lower bound is certainly improvable. We are thus convinced that the actual quality of our heuristic solutions is even better than the currently proven 5–15% within makespan optimality.

It should be noted again that indeed every detail of production is controlled by our plan, and that these plans were verified on-site at SZFG's coil coating line.

7 Summary and Conclusions

We have developed an exact mathematical model for the complex production planning task in coil coating and implemented optimization software solving it

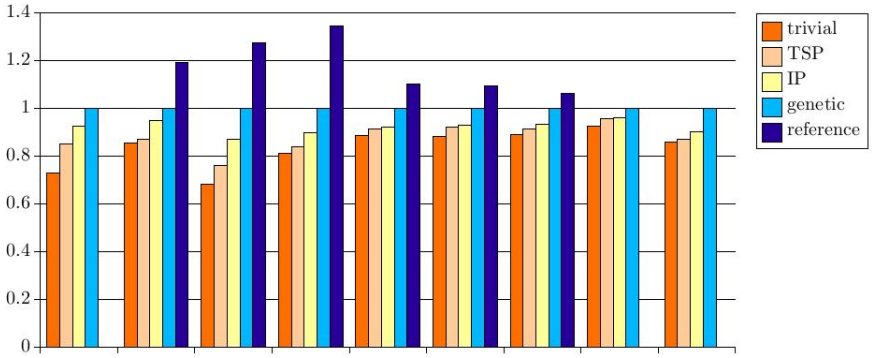


Fig. 8. Comparison of bounds on the makespan for representative instances. The ordering of bounds per instance is the same as in Tab. 3.

in practice. It fulfills all requirements regarding speed and robustness necessary for the application in the production environment. Our model takes into account all relevant aspects of production reality, including the subtasks of sequencing the coils, assigning tanks in shuttle coaters, and scheduling scarce work resources to perform setup work concurrently with production.The integrity of the plans computed and the correctness of cost calculations have been verified by the planners in charge of the coil coating line at Salzgitter Flachstahl GmbH.

Compared to previous plans, the optimized solutions yield double digit percentage reductions in makespan, greatly exceeding what was deemed possible: After all, the careful analysis of the problem necessary for devising the mathematical model has lead to a deep understanding of structure and interdependencies in the planning task, which enabled the realization of hidden optimization potential. It is not exaggerating to state that practitioners got fully acquainted with operating their machines only because of our theoretical investigations.

For the tank assignment and concurrent setup scheduling subproblem, we have developed a model allowing for the fast computation of optimal solutions in an environment where sufficient work resources are available. For the present resource-constrained case, we derived heuristic ideas from this model, which however at best slightly improve over the obvious FIFO rule for tank assignment on our test instances. Our ideas may still prove useful when applied to other, similar applications.

Finally, we have developed and implemented an integer programming model of a combinatorial relaxation of the problem, which we are able to solve to optimality via branch-and-price.The solutions yield lower bounds on the optimal makespan of our test instances, proving that our heuristic solutions have an optimality gap of no more than 5–15%, suggesting that not much optimization potential is left in the application after introducing our optimization module to production, which is planned for April 2009.

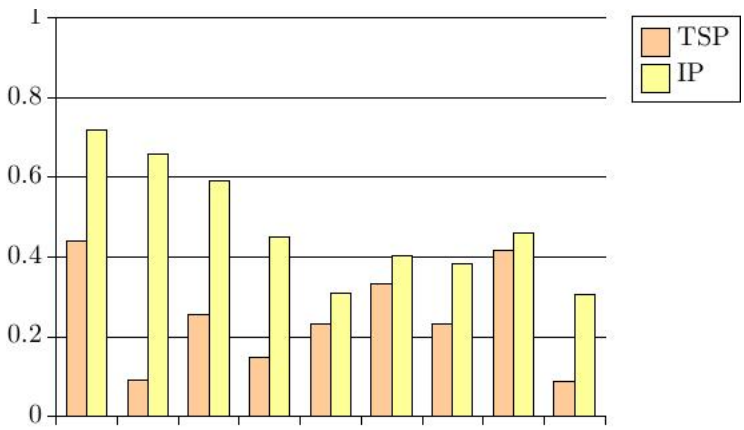


Fig. 9. Comparison of the quality of lower bounds in terms of cost, i.e., non-productive time. The cost of the solution obtained by our genetic algorithm defines the reference cost of 100%. Depicted is the percentage of the gap which could be closed by the two non-trivial bounds. The ordering of instances is identical to that in Tab. 3.

Acknowledgments We thank Michael Bastubbe, Torsten Gellert, and Olaf Maurer for their help in implementing the branch-and-price and genetic algorithms. Furthermore, we appreciate the patience of Frank Barcikowski, Andreas Holdinghausen, Sigurd Schwarz, and Meister Krake at SZFG for answering our countless questions concerning the even more countless details of setup cost, and for carefully verifying our solutions.

References

1. Delucchi, M., Barbucci, A., Cerisola, G.: Optimization of coil coating systems by means of electrochemical impedance spectroscopy. Electrochimica Acta **44** (1999) 4297 – 4305
2. Meuthen, B., Jandel, A.S.: Coil Coating. Vieweg+Teubner (2005)
3. Bar-Yehuda, R., Halldórsson, M., Naor, J., Shachnai, H., Shapira, I.: Scheduling split intervals. In: SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (2002) 732–741
4. Cook, S.: The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on Theory of computing. (1971) 151–158
5. Gupta, U., Lee, D., Leung, J.Y.T.: Efficient algorithms for interval graphs and circular-arc graphs. Networks **12** (1982) 459–467
6. Aarts, E., Lenstra, J., eds.: Local Search in Combinatorial Optimization. John Wiley & Sons (1997)
7. Meloni, C., Naso, D., Turchiano, B.: Multi-objective evolutionary algorithms for a class of sequencing problems in manufacturing environments. IEEE International Conference on Systems, Man and Cybernetics **1** (2003) 8–13

8. Mühlenbein, H., Gorges-Schleuter, M., Krämer, O.: Evolution algorithms in combinatorial optimization. Parallel Computing **7** (1988) 65–85
9. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. European J. Oper. Res. **126** (2000) 106–130
10. Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., Vance, P.: Branch-and-price: Column generation for solving huge integer programs. Operations Research **46** (1998) 316–329
11. Desrosiers, J., Lübbecke, M.: Selected topics in column generation. Operations Research **53** (2005) 1007–1023
12. Achterberg, T.: Constraint Integer Programming. PhD thesis, Technische Universität Berlin (2007) http://opus.kobv.de/tuberlin/volltexte/2007/1611/.