Sequencing and Scheduling in Coil Coating with Shuttles

Wiebke Höhn* Felix G. König Marco E. Lübbecke Rolf H. Möhring

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

Preprint 2009/1

January 31, 2009
Revised: March 12, 2010

Abstract

We consider a complex planning problem in integrated steel production. A sequence of coils of sheet metal needs to be color coated in consecutive stages. Different coil geometries and changes of coatings may necessitate time-consuming setup work. In most coating stages one can choose between two parallel color tanks in order to reduce setup times. As a complicating consequence, setup times for a coil may depend on the whole sequence of predecessors. A production plan comprises the sequencing of coils, and the scheduling of color tanks and setup work. The aim is to minimize the makespan for a given set of coils.

We present an optimization model for this integrated sequencing and scheduling problem. A core component is a graph theoretical model for scheduling. It is instrumental for building a fast heuristic which is embedded into a genetic algorithm to solve the sequencing problem. The quality of our solutions is evaluated via an integer program based on a combinatorial relaxation, showing that our solutions are within 10% of the optimum.

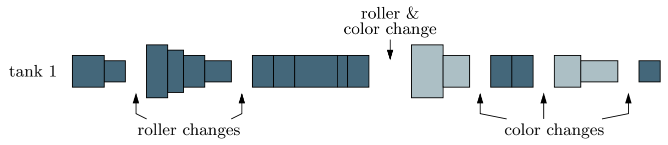
Our algorithm is implemented at Salzgitter Flachstahl GmbH, a major German steel producer. This has led to an average reduction in makespan by over 13% and has greatly exceeded expectations.

1 Introduction

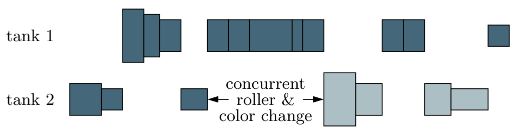
Almost every stage in integrated steel production requires solving a sequencing problem as subproblem, that is, deciding about a good order of steel slabs or coils in which they should be processed. This starts with the melt in the blast furnace, adding product specific alloys, and casting the strand 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.

In all of the above, however, *sequencing* is only one part of devising a complete production plan: For a given processing sequence, also a *schedule* for performing certain tasks on workpieces

*Supported by the German Research Foundation (DFG) as part of the Priority Program "Algorithm Engineering" (1307).



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



(b) Solution for a shuttle coater.

Figure 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 be coated with another color or is wider than its predecessor on that tank.

or machinery needs to be computed. The nature of this scheduling task naturally depends largely on the processing sequence, hence the quest for good sequences can only be answered accurately by taking into account the cost of the ensuing scheduling problem as well. Besides satisfactorily solving the practical problem, we consider our paper also as a significant mathematical contribution to tackling integrated sequencing and scheduling.

In this paper, we deal with the final processing step in sheet metal production, the coating of steel coils, which may be seen as a prototype of such an integrated problem, where computing a good sequence is inseparably linked to solving a subordinate scheduling 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 esthetic 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 already in the 1960s[1,2]. 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., Delucchi et al. (1999), Meuthen and Jandel (2005). 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 (in the scheduling literature uncommon) 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

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

problem. As we will see, it necessitates the solution of an integrated 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? We aim to find a sequence (the order of the coils) and a schedule for that sequence (the tank assignment and scheduling of setup work) that minimizes a joint objective (the makespan). Herein, even the second step alone (scheduling for a given sequence) comprises solving a separate NP-hard optimization problem. In contrast to sequence dependent setup cost known from literature, where non-productive time depends on consecutive work pieces, our costs may depend on *the whole sequence* in the worst case.

An elegant connection to a graph theoretic model paves the way for the design of our algorithm. It enables us to adequately model the scheduling step as an independent set problem in specially structured graphs. The theoretical results are of independent interest in their own right. We show that the scheduling step is indeed NP-hard when the number of shuttle coaters is part of the input. For a fixed number of shuttle coaters, we obtain a polynomially solvable dynamic program, inspiring a fast and good heuristic which we embed into a genetic algorithm for sequencing.

Altogether, we develop a fast, practical heuristic which solves the integrated sequencing and scheduling problem and computes a detailed production plan for the coil coating line. The quality of our plans is assessed with the help of an integer program that we solve by branch-and-price.

Our algorithm has been added to PSI Business Technology's planning software suite[3], and is currently in use at Salzgitter Flachstahl GmbH[4], Germany (SZFG for short). There, it yields an average reduction in makespan by over 13% as compared to the previous manual planning process. In addition, our lower bounds suggest that the makespan of the solutions computed by our algorithm is within 10% of the optimal makespan for typical instances. This success has made this contribution a finalist of the 2009 EURO Excellence in Practice Award.

2 Problem Formulation

We now give a detailed description of the optimization task at hand. For a better understanding, we first describe the production process at the coil coating line in some detail in Sect. 2.1 before giving a formal problem definition in Sect. 2.2.

2.1 Application

Steel coils are a highly individualized product, and all non-productive time in coil coating depends on certain characteristics of coils, so we first state the most important ones for a better understanding. Coils usually have a length of 1–5 km, take 20–100 minutes to run through the coating line, and some of their central attributes are naturally the colors they receive in the four coating stages, chosen from a palette of several hundred. But also their width, usually 1–1.8 m, and their weight per meter of the coil impact setup. We refrain from listing further coil attributes, since their list is very long; yet all relevant information is included in our setup calculations.

Before entering production, each coil is unrolled and stapled to the end of its predecessor. During all non-productive time, scrap coils are inserted in between actual coils, so essentially a never-ending strip of sheet metal is continuously running through the coil coating line. Sling

[3]`www.psi-bt.de`

[4]`www.salzgitter-flachstahl.de/en/`

buffers at the beginning and the end of the line compensate for time needed to staple and unstaple joints between coils. 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. In the ovens, the respective coating layers are fixed. After the coating process, the coils are rolled up again, now ready for shipping.

The following terms are essential to understand the coil coating process and our optimization task:

- We intuitively refer to the possible types of coating material as *colors*.
- A *coater* comprises all machinery necessary for applying one of the four layers of color to the coils, primer and finish on top and bottom.
- A *tank* holds the color currently in use at a coater, and each tank has its own rubber *roller* for applying the color to the coil.
- Naturally, a coater has at least one tank. A *shuttle coater* has two tanks, each with its own roller, which can be used alternatingly to apply color.
- A *color change* refers to cleaning a tank and filling it with a new color. When a tank's roller has incurred wear, it needs to be replaced, and this is referred to as a *roller change*.

A schematic view of a typical coil coating line is depicted in Fig. 2.

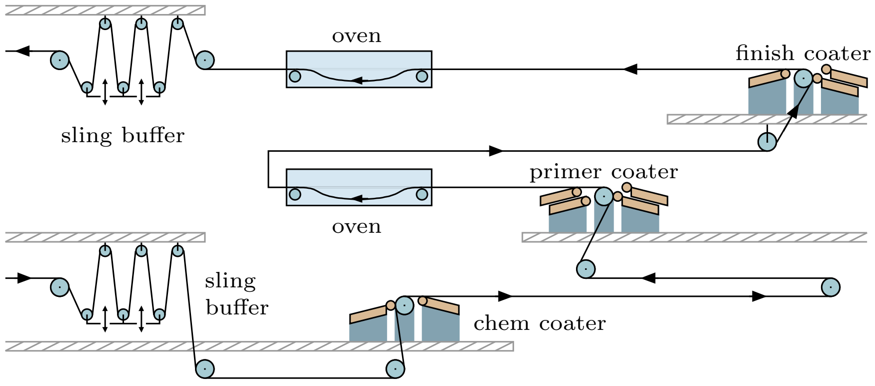


Figure 2: Schematic view of a coil coating line with chem, primer, and finish coater. The chem and the bottom finish coaters are standard coaters, the remaining have shuttles.

Given a set of coils, a plan for their coating comprises the following two parts:

- a *sequence* of the coils, i.e., an order in which the coils are to be run through the line,
- a *schedule* consisting of a *tank assignment* stating for each coil from which tank it is coated on each of the three shuttle coaters, and a *setup schedule* assigning to each necessary color or roller change a time interval for performing it.

The optimization goal is to find a coil coating plan minimizing the *makespan*, i.e., the completion time of the last coil in the sequence. This is essentially equivalent to minimizing non-productive time in the plan, which ensues for three reasons:

- To satisfy certain technical restrictions, intermediate scrap coils of different lengths, widths etc. may be required in between coils.
- Depending on the order of the coils and the usage of the shuttle coaters, non-productive time may occur when the color in a tank needs to be changed or a roller must be exchanged due to wear.
- Sample runs need to be conducted at certain points in the sequence.

We refer to all of the above uniformly as *setup*. The need for scrap coils due to the first point depends only on two consecutive coils in the sequence, consequently we call these setups *local*. In contrast, color and roller changes on shuttle coaters depend on two coils run in consecution *on the same tank*. Such pairs of coils may span a large part of the sequence, hence we call these setups *global*.

Sample runs may fall in either of the two categories. In any case, they blend into our optimization scheme seamlessly, i.e., they can be thought of as (easy) special cases of the other local or global setups. Hence, for the sake of clarity of our presentation, we do not elaborate on them further from here on.

Local setups always increase the makespan of a plan, as they correspond to the insertion of scrap coils into the plan in any case. Contrarily, the impact on the makespan of global setup, i.e., color and roller changes, can be avoided or diminished when smartly utilizing shuttle coaters. For brevity, we refer to all non-productive time in between coils, i.e., the length of the inserted scrap coils, as *cost*.

2.1.1 Local Setup.

Some scrap coils need to be inserted into the production sequence to bridge differences in certain attributes of subsequent coils. An example is the difference in weight per meter of the coil: Due to the long distance between the supporting points in the oven, the stapled joint between the coils could break if the weights of consecutive coils differ by too much. Hence, one or more scrap coils with intermediate weight per meter need to be inserted to bridge the difference.

There are many more coil attributes which may require such transitions, like oven temperatures, thickness of coils etc. It is essential to note, however, that *one* appropriate scrap coil may bridge differences in *several* attributes simultaneously: The length of scrap coils needed in between two coils, i.e., the resulting cost, is determined by the maximum scrap coil length required by any one of the attributes.

Also note that local setup is completely oblivious of the scheduling. It solely depends on the order of the coils given by the sequence.

2.1.2 Global Setup.

In order to determine whether a color or roller change needs to be performed on a shuttle coater, it does not suffice to consider coils consecutive *in the sequence*. Rather, coils run in consecution *on the same tank* need to be considered. Hence, setup of this type may depend on large parts of the sequence—the entire sequence in the worst case.

If consecutive coils run on the same tank of a coater require different colors in this coating stage, the tank needs to be cleaned and refilled with the new color. Quite similarly, the rollers have to be changed when a coil with smaller width is preceding a coil with larger width on the same tank: Due to the wear the roller incurred at the edges of the first coil, the coating of the second coil would bear imperfections. Obviously, color and roller changes have to be performed after the first coil has been coated, and before coating of the second commences. In Sect. 2.1.3, we discuss all constraints to be respected when scheduling these setup tasks.

2.1.3 Concurrent Setup.

An obvious way for a smart tank assignment to avoid cost is to preserve a color and/or a roller for a later coil on an idle tank, i.e., to use the other tank for coating coils temporarily before switching back to it. This reduces the number of setup tasks to be performed. A second, in a sense complementary way to utilize a shuttle coater's second tank is to reduce or eliminate the impact of setups on the makespan: Color and roller changes can be performed on an idle tank while coating continues on the other. We refer to such tasks performed simultaneously with production as *concurrent setup*.

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. Also, in our application there is only one team of workers who can perform setup work during regular production. Consequently, concurrent setup is possible on all idle tanks and their rollers, but only on one tank at a time.

Quite clearly, concurrent setup on an idle tank on the one hand, and preserving a certain color and/or roller for a later coil on the other hand cannot be realized at the same time, illustrating an inherent difficulty in optimal planning for shuttle coaters. Already very small examples show that simple tank assignment rules like the one previously in use at SZFG, which switches tanks on a shuttle coater whenever a new color is required, fail to be optimal. We devote our attention to the scheduling, i.e., to tank assignment and concurrent setup scheduling, in Sect. 3.

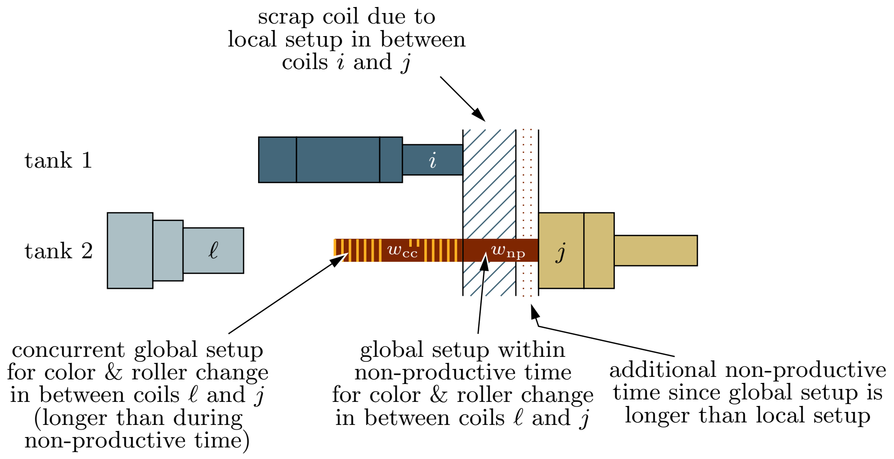

scrap coil due to local setup in between coils i and j
tank 1
tank 2
i
ℓ
j
w_{cc}
w_{np}
concurrent global setup for color & roller change in between coils ℓ and j (longer than during non-productive time)
global setup within non-productive time for color & roller change in between coils ℓ and j
additional non-productive time since global setup is longer than local setup

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

2.1.4 Cost Calculation.

We shall briefly illustrate how cost, i.e., non-productive time included in the makespan, computes for a given coil coating plan. The cost incurred before a certain coil results from the different kinds of setup described above. In turn, the set of setup to be performed depends on both the sequence of coils and the tank assignment.

Consider Fig. 3. The first part of global setup w_{cc} on tank 2 is performed concurrently: It takes place after coil ℓ, while coils are being coated on tank 1. When coil i is finished, a period of non-productive time begins. First, a scrap coil is run as local setup for coil j—during this time, the global setup w_{np} still does not result in additional cost. Only once local setup ends, additional non-productive time ensues while w_{np} is finished.

2.2 Problem Formulation

We proceed to give a formal problem definition capturing all relevant aspects of the application described above. An instance comprises a set $[n] := \{1, \dots, n\}$ of coils to be coated. For each coil j,

- its colors $c^{(j)} \in \mathbb{N}^m$ on the m coaters,
- its width $w_j \in \mathbb{R}^+$,
- and its processing time $p_j \in \mathbb{R}^+$

are specified. Finally, for each pair of coils $(i, j) \in [n] \times [n]$, the duration of local setup if they are run consecutively is specified as $s_{\text{loc}}(i, j)$.

A plan for the coil coating process consists of the following two parts:

- a *sequence* $\pi \in \Pi_n$, i.e., a permutation stating the processing order of the coils
- a *schedule* S comprising a *tank assignment* $T \in \{1, 2\}^{n \times m}$ stating for each coil from which tank it is coated for each of the m shuttle coaters, and a *feasible setup schedule* assigning a start time to every setup task in the set of global setup $W = W(\pi, T)$ defined by π and T. We elaborate on the structure of feasible schedules below.

There is a strong interdependence among the different parts of a solution: A tank assignment T is only meaningful in conjunction with a sequence π, and the set W of global setup to be scheduled depends on T and π.

The set W comprises color and roller changes. Both require roughly the same amount of time, which we denote by t_c. Furthermore, for a coil j, let $\text{pred}_{\text{seq}}(j)$ denote the coil directly preceding it in π, and $\text{pred}^{(k)}_{\text{tank}}(j)$ its predecessor on the same tank of coater k.

Now, with $i := \mathrm{pred}_{\mathrm{tank}}^{(k)}(j)$, the respective durations of color and roller changes on coater k for coil j are

$$s_{cc}^{(k)}(i,j) = \begin{cases} t_c & \text{if } c_k^{(i)} \neq c_k^{(j)}, \\ 0 & \text{otherwise,} \end{cases}$$

$$s_{rc}^{(k)}(i,j) = \begin{cases} t_c & \text{if } w_i < w_j, \\ 0 & \text{otherwise,} \end{cases}$$

yielding a total global setup $s_{\mathrm{glob}}^{(k)}(i,j) := s_{cc}^{(k)}(i,j) + s_{rc}^{(k)}(i,j)$ on coater k.

Adding the duration of local setup, and assuming for a moment that all global setup for coil j is performed during non-productive time right before it, the total cost incurred by a solution right before j is

$$s\left(\mathrm{pred}_{\mathrm{tank}}(j),\, \mathrm{pred}_{\mathrm{seq}}(j),\, j\right) := \max\left\{s_{\mathrm{loc}}(\mathrm{pred}_{\mathrm{seq}}(j), j),\, \sum_k s_{\mathrm{glob}}^{(k)}(\mathrm{pred}_{\mathrm{tank}}^{(k)}(j), j)\right\}, \tag{1}$$

where $\mathrm{pred}_{\mathrm{tank}}(j)$ denotes the vector of the $\mathrm{pred}_{\mathrm{tank}}^{(k)}(j)$; see Fig. 4 a).

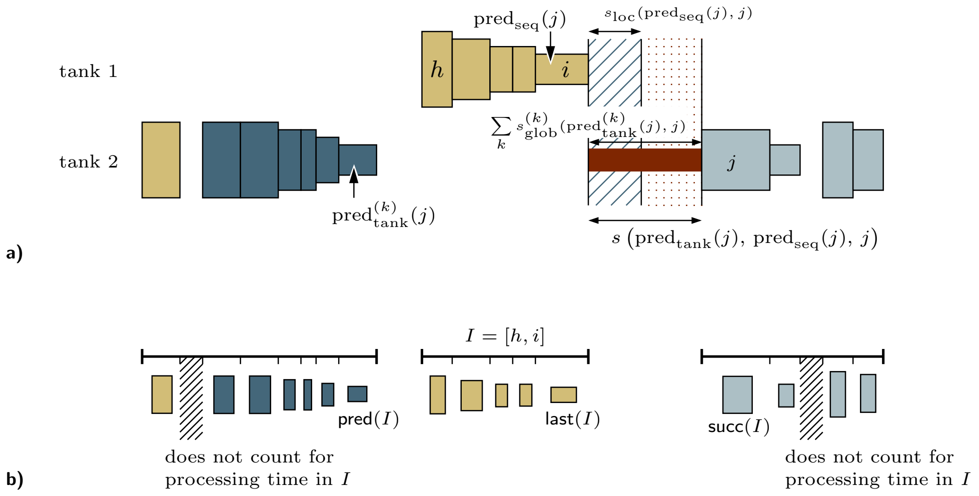

$\mathrm{pred}_{\mathrm{seq}}(j)$
$s_{\mathrm{loc}}(\mathrm{pred}_{\mathrm{seq}}(j), j)$
tank 1
h
i
$\sum_k s_{\mathrm{glob}}^{(k)}(\mathrm{pred}_{\mathrm{tank}}^{(k)}(j), j)$
tank 2
j
$\mathrm{pred}_{\mathrm{tank}}^{(k)}(j)$
a)
$s\left(\mathrm{pred}_{\mathrm{tank}}(j),\, \mathrm{pred}_{\mathrm{seq}}(j),\, j\right)$
$I = [h, i]$
pred(I)
last(I)
succ(I)
does not count for processing time in I
does not count for processing time in I
b)

Figure 4: Components of cost s (i.e., we assume no concurrent setup), and definition of coater intervals.

This cost can be greatly reduced, however, by a good setup schedule, i.e., by performing as much global setup as possible concurrently to regular production: On a shuttle coater, color and roller changes can be performed on one tank, while coils are being coated on the other. Due to limited work resources, only one concurrent setup task may be performed at any point in time in a feasible such schedule. Furthermore, concurrent setup work does not allow preemption by another concurrent setup job, i.e., a job on one coater must be finished before work is started on another. Finally, concurrent setup incurs a slowdown factor $\lambda \geq 1$, increasing the duration of a color or roller change to $\lambda\, t_c$ when performed while coils are coated on the other tank. If a global setup is started as a concurrent task but does not finish in time, it is completed during non-productive time. In this case, only part of the global setup incurs the slowdown factor.

The start time of coils is implicitly given by the setup schedule, since all setup for a certain coil needs to be finished before it can be processed. The objective is to determine a coil coating plan (π, S) minimizing the *makespan*, i.e., the completion time of the last coil in π.

We conclude this section with a remark forming the basis for concurrent setup scheduling in Sect. 3. Assume we are given a fixed sequence of coils π. Once a subschedule S_{cst} for all concurrent setup tasks has been computed, all other setup, i.e., local setup and remaining color and roller changes in non-productive time, can easily be added to it to obtain a complete feasible schedule for all tasks in $W(\pi, T)$: Local setup can be added to S_{cst} in parallel to remaining global setup tasks, delaying the start time of the following coil by the maximum of both (see Fig. 3). Color and roller changes in non-productive time are always performed directly before the actual coil they are for, yielding an unambiguous rule for adding the remaining setup to S_{cst}.

3 Interval Model for Scheduling

In our solution approach, which we describe in Sect. 5, we repeatedly compute schedules for different sequences of coils. This leads to the following subproblem: Given a fixed sequence π of coils, we now aim to find a schedule S leading to a coil coating plan (π, S) with minimal cost. We develop a representation of schedules for a given π as a family of weighted 2-dimensional intervals, where the first dimension is related to tank assignment, the second to performing concurrent setup. We call two such intervals, or axis-parallel rectangles, *independent*, if their projections onto neither of the axes intersect. An optimal solution S for given π will correspond exactly to a maximum weight subset of pairwise independent 2-dimensional intervals, cf.Fig. 5. This section essentially lays the groundwork for the design of both the fast heuristic algorithms and a dynamic programming approach yielding optimal solutions for fixed sequences of coils in polynomial time for any constant number of coaters.

We model every possible assignment of a maximal consecutive 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 combine coater and work intervals to form 2-dimensional intervals, i.e., axis-parallel rectangles. The weights of intervals represent the reduction in cost (which may be negative) achieved by the corresponding partial tank assignment and the concurrent setup work performed, compared to processing all coils on the same tank without performing concurrent setup work, which by (1) would incur cost

$$c_{init} = \sum_{i=1}^{n-1} s(i,\, i,\, i+1).$$

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 position the rectangles in the plane accordingly, such that there is a one-to-one correspondence between tank assignments with concurrent setup work schedules on the one hand, and maximal independent sets of rectangles on the other. We denote by p_j the processing time of coil j, and throughout this section, we assume that the coils are numbered as they appear in the fixed sequence, i.e., $\pi = id \in \Pi_n$.

3.1 Coater Intervals

A coater interval, always associated with a particular coater, represents a maximal consecutive subsequence of coils that are processed on the same tank. Thus, for each coater, any two coils $i, j \in [n]$, $i \leq j$, define a coater interval $I = [i, j]$ containing i, j, and all scrap coils. Selecting such an interval will correspond to switching the tank directly before i, and directly after j; see Fig. 4 b). Hence, a set of non-intersecting coater intervals, covering every coil, defines a unique tank assignment, and the processing time in I is

$$p(I) := \sum_{x=i}^{j} p_x. \tag{2}$$

We call the last coil before and the first coil after I its predecessor $\mathsf{pred}(I)$ and successor $\mathsf{succ}(I)$, respectively. Also, let $\mathsf{last}(I)$ denote the last coil in I. The weight $w_{\text{coat}}(I)$ of a coater interval I comprises the cost savings before $\mathsf{succ}(I)$ 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 are assigned to work intervals as described in the next section. Thus, with (1), w_{coat} computes as

$$w_{\text{coat}}(I) \;=\; s\left(\mathsf{last}(I),\, \mathsf{last}(I),\, \mathsf{succ}(I)\right) - s\left(\mathsf{last}(I),\, \mathsf{pred}(I),\, \mathsf{succ}(I)\right).$$

Note that certain coater intervals may have negative weight, i.e., selecting them actually increases cost. Due to the requirement that all of π be covered by coater intervals, such intervals cannot be ignored. For computational purposes however, negative weights can easily be eliminated, as we demonstrate in Sect. 3.4.

3.2 Work Intervals

A work interval is always associated with a coater interval $I = [i, j]$, hence also with a particular coater. It represents a certain concurrent setup task performed for $\mathsf{succ}(I)$ on this coater's idle tank during coils $i, \ldots, j$. A work interval's length describes the amount of work completed, which is at most $\lambda\, t_c$. Since concurrent setup work must not be preempted by other concurrent setup, work intervals of length less than $\lambda\, t_c$ are only allowed directly before $\mathsf{succ}(I)$, i.e., at the end of the corresponding coater interval—work is completed during non-productive time directly after $\mathsf{last}(I)$ in this case. There are at most two work intervals for a coater interval, one each for a color and a roller change, and the sum of their lengths must not exceed $p(I)$.

If a color change is necessary in between $\mathsf{pred}(I)$ and $\mathsf{succ}(I)$, i.e.,

$$s_{cc}^{(k)}(\mathsf{pred}(I), \mathsf{succ}(I)) > 0$$

for a coater k, cost may be saved by adding a work interval I_c for a color change to I. Analogously, if

$$s_{rc}^{(k)}(\mathsf{pred}(I), \mathsf{succ}(I)) > 0,$$

a work interval I_r for a roller change may be added to I in the same way.

In principle, a work interval may start at any point of a particular coater interval I, making it hard to specify a set of work intervals from which to select. However, since preemption is not allowed, it suffices to consider start times of the form $\sum_{h=1}^{\ell} p_h + q\lambda\, t_c$ for some $\ell \leq i$ and $q \in \mathbb{N}$, ignoring non-productive time. We formally prove this fact in Sect. 5.2.1, where we also show that the number of work intervals needed in the model to guarantee an optimal solution is polynomially bounded in the number of coils n.

For an exact definition of the cost savings due to work intervals, local setup directly after their coater interval needs to be subtracted from their savings, since concurrent setup work may be performed during the resulting scrap coils at no cost as well. Consequently, the cost saving associated with work intervals I_c and I_r is

$$w_{\text{work}}(I_c, I_r) \;=\; \max\{0,\; 1/\lambda(p(I_c) + p(I_r)) - s_{\text{loc}}(\mathsf{last}(I), \mathsf{succ}(I))\}.$$

We now define *saving rectangles* $R = (I, I_c, I_r)$ as a combination of a coater interval I and two (possibly empty) work intervals I_c and I_r for color and roller changes. The total weight of a saving rectangle R is

$$w(R) \;=\; w_{\text{coat}}(I) + w_{\text{work}}(I_c, I_r).$$

As for coater intervals, 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. We show in Sect. 5.2.1 that this poses no problem for our algorithmic ideas, while thinking of R as a rectangle facilitates the description of the model.

3.3 Positioning Rectangles

Recall that a schedule shall correspond to the selection of a maximum weight independent subset of all possible saving rectangles. We now position axis-parallel saving rectangles in the plane such that their projections onto one of the axes intersect, if and only if they *conflict*, i.e., if their work intervals intersect, or if they belong to the same coater and their coater intervals intersect. Quite naturally, the x and y-dimensions of the plane, for coater and work intervals respectively, are both associated with a sense of time. However, each coater's rectangles have their own section of the x-axis. Namely, by (2), $(k-1)p\,([1,n])$ is added to the x-coordinate of the k-th coater's rectangles, while all y-coordinates remain unchanged.

The savings of conflicting rectangles can clearly not be realized at the same time: Coater intervals of one coater, whose intersection contains a coil j, lead to conflicting tank usage, since both intervals assign j to a different tank, while intersecting work intervals violate the constraint that only one work resource is available. Hence, a maximum weight subset of pairwise independent saving rectangles, whose coater intervals cover all coils on all coaters, corresponds to a tank assignment with feasible concurrent setup schedule which is optimal for the given sequence of coils. It realizes the greatest savings possible over running all coils on the same tank.

By the construction above, saving rectangles have a very specific structure, see Fig. 5. We build upon these properties when analyzing the independent set problem ensuing from concurrent setup scheduling in Sect. 4:

- The projection of each rectangle onto the x-axis is contained completely in one of the distinct sections corresponding to the coaters.
- If the projections onto the y-axis of two rectangles intersect, their projections onto the x-axis contain common coils.

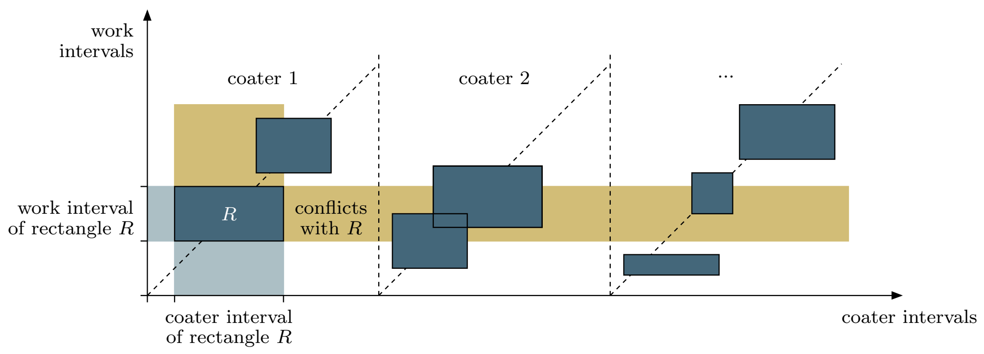


Figure 5: Some saving rectangles, appropriately positioned in the plane. Selecting rectangle R, i.e., running all coils belonging to its coater interval on the same tank of coater 1 and performing concurrent setup during its work interval, prohibits the selection of any rectangle whose projection onto one of the axes intersects that of R. Non-disjoint coater intervals deny the derivation of a proper tank assignment, while intersecting work intervals violate resource constraints.

The latter is due to the fact that time-wise, a rectangle's work interval is always contained in its coater interval. Note that projections of rectangles onto the x-axis containing common coils need not necessarily intersect when positioned as in Fig. 5—they may well belong to different coaters and thus lie in different sections of the x-axis.

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 maximum weight independent set to be maximal, i.e., we can force 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 local setup, 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 rectangles $\mathcal{R}$, i.e.,

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

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.

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 finding a maximum subset of savings amounts to selecting a maximum weight subset of pairwise disjoint intervals, which can be done very efficiently (Gupta et al. 1982). Hence our model yields a fast algorithm for the tank assignment and concurrent setup scheduling problem when sufficient work resources are available.

4 Independent Sets in Special 2-Union-Graphs

We now make a connection from tank assignments and concurrent setup scheduling to the independent set problem for a special class of multi-interval intersection graphs. While various notions of multi-intervals and their intersection graphs have been studied in literature (e.g., Gyárfás and Lehel 1969, Kaiser 1997, Butman et al. 2007), we are mainly concerned with the following class of graphs.

Definition 1 *An undirected graph $G = (V, E)$ is called a t-union graph, if it is the edgewise union of t interval graphs with common node set V.*

Recall that an interval graph $G = (V, E)$ is a graph whose node set can be represented as a family of $|V|$ intervals where two intervals contain a common point, if and only if the corresponding nodes share an edge in G. As is common in literature, we assume that a suitable interval representation for G is given and use the notion of a node in G and its interval in the given representation interchangeably. In a t-union graph, we denote by v^i the interval associated with a node v in its i-th interval graph.

Coater and work intervals as defined in Sect. 3 define two interval graphs, and their edgewise union is the graph on the set of savings rectangles where two rectangles share an edge exactly if their savings cannot be realized jointly. Hence a maximum pairwise non-conflicting subset of savings rectangles corresponds to a maximum weight independent set in a 2-union graph. This problem is studied by Bar-Yehuda et al. (2002), and the authors prove it to be APX-complete, even for 2-union graphs which have a representation where all intervals are half open and have length two, and all of their end points are integral. On the positive side, they give a $2t$-approximation algorithm based on an LP-relaxation of the problem for the more general class of 2-interval graphs.

In this section, we demonstrate how the special structure of the intervals in our application can be exploited to partially circumvent the hardness result above. In order to precisely define this special class of 2-union graphs, we first introduce a certain notion of affinity among the disjoint sections of the coater dimension corresponding to the m different shuttle coaters.

Definition 2 *Let V' be a family of intervals contained in a section $L = [s_L, e_L]$ of the real line. For an interval $v = [s_v, e_v] \in V'$, we denote by*

$$v_L := [s_v - s_L, e_v - s_L]$$

*the L-*relative interval *of v. When referring to some canonical section of the real line, we call these intervals* section-relative.

Definition 3 *Let G be a 2-union graph, i.e., the edgewise union of two interval graphs G_1, G_2. We call G m-*composite, *if G_1 and G_2 are the intersection graphs of two families of intervals such that*

1. G_1 has at least m connected components $C_1, \ldots, C_m$.

2. Let $L_1, \ldots, L_m$ denote the minimal sections of the line containing all intervals in $C_1, \ldots, C_m$ respectively, and assume w.l.o.g. that, by scaling G_1's intervals, all L_i have equal length. Then for any $u, v \in V(G)$ with $u^1 \in C_i$, $v^1 \in C_j$,

$$u^2 \cap v^2 \neq \emptyset \quad \Rightarrow \quad u^1_{L_i} \cap v^1_{L_j} \neq \emptyset. \tag{3}$$

Loosely speaking, when two nodes share an edge in G_2, their intervals in G_1 would intersect, if they belonged to the same section. Note that 1-composite and n-composite 2-union graphs are regular interval graphs: In the former case, G_2 is a subgraph of G_1, so $G = G_1$, while in the latter, G_1 has no edges, so $G = G_2$.

First, we show that for variable m, or in terms of scheduling, the number of machines part of the input, the maximum independent set problem remains *NP*-hard for m-composite 2-union graphs, even in the unweighted case. We proceed to describe an exact dynamic programming approach running in polynomial time for any fixed m.

Theorem 1 *For m part of the input, the maximum independent set problem in m-composite* 2-*union graphs is strongly* NP-*hard, even in the unweighted case.*

Proof. Proof. We give a reduction from 3-Sat (Cook 1971). Let I denote an arbitrary 3-Sat instance with n variables $x_1, \ldots, x_n$ and r clauses $c_1, \ldots, c_r$. We may assume w.l.o.g., that no clause contains two literals of the same variable. With $m := 2n$, we now construct an m-composite 2-union graph G, such that G admits an independent set of cardinality $n + r$, if and only if there is a truth assignment for the variables of I such that all clauses are satisfied.

We construct G as the edgewise union of two interval graphs G_1 and G_2. 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	$[2r \cdot (i-1); 2r \cdot (i - \frac{1}{2}))$	$[i-1; i)$
$\overline{x}_i$	$[2r \cdot (i - \frac{1}{2}); 2r \cdot i)$	$[i-1; i)$

Table 1: Intervals associated with literals in the two 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, while s is 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.

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 $n + r$. 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 a triangle, so similarly, S may contain at most r of these nodes. Consequently, S

Occurrence in clause	Interval in G_1	Interval in G_2
c_j	$[s+\frac{j-1}{r}; \frac{s+j}{r})$	$[n+j-1; n+j)$

Table 2: Intervals associated with occurrences of literals in clauses in the two 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.

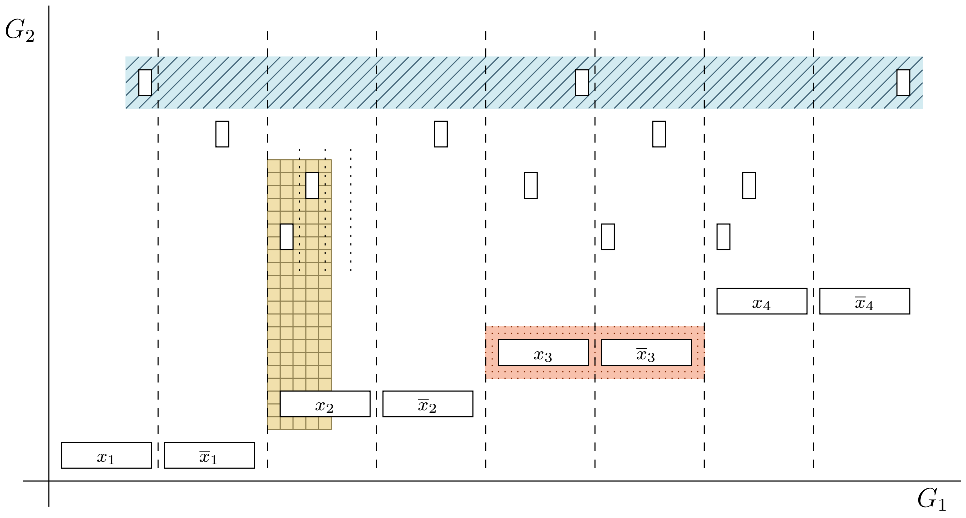


Figure 6: Two-dimensional interval representation of the graph constructed from the 3-Sat instance with four variables and clauses $(\overline{x}_2 \vee x_3 \vee \overline{x}_4)$, $(\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)$, $(x_1 \vee x_2 \vee x_3)$, $(\overline{x}_1 \vee \overline{x}_3 \vee x_4)$.
The 2-intervals associated with the two literals of variable x_3 are highlighted with dots, those belonging to the fourth clause $(\overline{x}_1 \vee \overline{x}_3 \vee x_4)$ with stripes. The adjacency of the occurrences of literal $\overline{x}_2$ in the first two clauses with x_2 is highlighted with squares.

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.

Finally, for $m = 2n$, the graph G is also m-composite: Firstly, no interval in G_1 crosses the end points of the intervals corresponding to literals, so these intervals define m disjoint sections of the line containing the connected components of G_1. Secondly, edges in G_2 either join two literals of the same variable, or two occurrences of two literals in the same clause. In both cases, the adjacent nodes' intervals in G_1 have identical section-relative intervals by our construction, hence G also satisfies property (3). $\square$ $\square$

Note that this reduction actually demonstrates that the maximum independent set problem is even *NP*-hard in 2-union graphs of a very specific structure: The graph constructed is the edgewise union of a collection of pairwise disjoint triangles and edges on the one hand, and a collection of disjoint stars on the other.

Nevertheless, let us now turn to positive results. We propose the following dynamic programming algorithm to compute a maximum weight independent set in m-composite 2-union graphs, which runs efficiently when m, i.e., the number of machines in the scheduling interpretation of the problem, is a constant.

Theorem 2 *For any constant m, a maximum weight independent set in m-composite 2-union graphs can be computed in polynomial time by dynamic programming.*

Proof. Proof. Let the graph G again be represented as the edgewise union of two interval graphs G_1 and G_2. We define a *state* in our algorithm to be a m-tuple

$$S_v = (v_1, \dots, v_m),$$

where v_i is either a node whose interval in G_1 lies in component C_i, or empty. Note that the nodes of one state are thereby independent in G_1, and $(n+1)^m$ is a rough upper bound on the number of states. We call a state *feasible*, if its nodes also form an independent set in G_2 (and hence in G).

We now define a directed acyclic graph $\mathcal{S}$ on the set of feasible states, and prove that a longest path from the empty state, whose components are all empty, to some other state corresponds to a maximum weight independent set in G. Since the number of states is polynomially bounded, this will yield the result.

We call two states S_u, S_v *compatible*, if $S_u \cup S_v$ forms an independent set in G. As in Def. 3, we denote by $L_1, \dots, L_m$ the minimal sections of the line containing all intervals in $C_1, \dots, C_m$ respectively, and assume again w.l.o.g. that all L_i have equal length. For the i-th component v_i of S_v, let $s_L(v_i)$ and $e_L(v_i)$ denote the start and end point of the L_i-relative interval of v_i^1 in G_1; set $s_L(v_i), e_L(v_i)$ to zero when v_i is empty. We define the edge set of $\mathcal{S}$ as

$$E(\mathcal{S}) := \{(S_u, S_v) : S_u, S_v \text{ compatible and } e_L(u_i) \leq e_L(v_j) \quad \forall i, j = 1, \dots, m\}.$$

So in a sense, when following a path in $\mathcal{S}$, the corresponding intervals in all components of G_1 are traversed simultaneously and in a certain monotone fashion.

Now $\mathcal{S}$ is clearly acyclic, and we argue that any path $P = (S_s, \dots, S_t)$ in $\mathcal{S}$ defines an independent set $S_s \cup \cdots \cup S_t$ in G: First, the union of two subsequent states in P is independent by definition of $E(\mathcal{S})$. Furthermore, the union of all states in P is independent in G_1 by construction. It remains to show that the union of any two non-subsequent states in P is independent in G_2 as well.

For the sake of contradiction, assume there is a state $S_u \in P$ and a node $u_i \in S_u$ which, in G_2, shares an edge with a node v_j contained in a state succeeding S_u on P. Let S_v denote the first such state. Since adjacent states form independent sets by definition of $E(\mathcal{S})$, there must be a state S_a in between S_u and S_v on P which contains neither u_i nor v_j, so in particular $a_j \neq v_j$. Also, by our assumptions, $S_u \cup S_a$ and $S_a \cup S_v$ form independent sets.

Now, by property (3), we have

$$s_L(v_j) \leq e_L(u_i) \leq e_L(v_j).$$

On the other hand, from the definition of $E(\mathcal{S})$, we get

$$e_L(u_i) \le e_L(a_j),$$

and from the fact that $S_a \cup S_v$ forms an independent set and $a_j \ne v_j$

$$e_L(a_j) < s_L(v_j).$$

Finally, this results in

$$e_L(a_j) < s_L(v_j) \le e_L(u_i) \le e_L(a_j),$$

a contradiction. Consequently any path in $\mathcal{S}$ defines an independent set in G.

We now define the length of an edge (S_u, S_v) as the weight gained by a path, i.e., an independent set, through its traversal:

$$\ell((S_u, S_v)) := \sum_{x \in S_u \cup S_v} w_x - \sum_{x \in S_v} w_x.$$

Since for any independent set in G, its nodes can be ordered by the end points of their section-relative intervals in G_1, any maximum weight independent set also has a representation as a path in $\mathcal{S}$, concluding our argument. □ □

Note that in view of the hardness result for variable m (Thm. 1), this approach may well be the best possible in terms of efficiency.

5 Algorithm

We 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 24–72 hours must be computed within 180 seconds. This is why we propose a special purpose heuristic, which could not have been developed without the insights gained in the previous sections. Moreover, as we detail in Sect. 6, 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 due to global setup cost, an effective algorithmic scheme cannot be based on local decisions or optimal substructures alone. Instead, meta heuristics generating many *complete* solutions quickly are a natural approach. In their course, the cost of many sequences needs to be evaluated, so the scheduling needs to be performed very often as well. Hence, fast algorithms for this subproblem are required. We describe these ingredients in the following subsections.

5.1 Sequencing

We utilize a genetic algorithm for sequencing, which by its nature combines solutions in such a way that beneficial characteristics of solutions found persist, while costly characteristics are eliminated. The set of solutions maintained by such algorithms is commonly referred to as *population*. An algorithm's defining operations are so called *crossovers* and *mutations*. In a crossover, the combination of two parents from the current population brings forth a new individual, while a mutation creates a new individual by modifying one already present. In an iteration, or *generation* of the algorithm, its population is first enlarged through crossovers and mutations, before a subset of all these individuals is selected for survival. See Aarts and Lenstra (1997) for a general overview of the topic and Meloni et al. (2003) for a recent successful application to a different production sequencing problem.

The key to the success of our algorithm lies in the construction of its initial population, which exploits the special cost structure of our problem. Computational experience has shown that not so much the absolute quality of the initial solutions, but much more their diversity w.r.t. different beneficial aspects lead to rapidly evolving populations.

Recall that the cost of a solution essentially computes as the sum of local and global cost, where the former only depends on the sequence, while the latter is greatly affected by the scheduling. In a sense, we would like to eventually avoid global cost by switching tanks smartly, so we focus on local cost for the initial population.

As described in Sect. 2.1.1, local setup ensues due to scrap coils which need to be inserted into the sequence in order to bridge differences in certain criteria r of subsequent coils, like their weight per meter, thickness, etc. This share of the cost incurred between two coils i and j has the following structure:

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

where δ_r denotes the length of local setup necessary to bridge the difference between i and j in criterion r.

When considering only one single criterion, we may be able to avoid such setup cost completely by sorting the set of coils according to it, minimizing the greatest occurring difference between subsequent coils. On the other hand, by the definition of s_{loc}, scrap coils can be used to bridge differences in several criteria at the same time, making the most of some unavoidable local setup, in a sense. These two ideas are the essence of the algorithm generating our initial population.

To create an individual $\pi \in \Pi_n$, we first pick a criterion r_1 by which we sort the set of coils to be coated. To break ties, which occur frequently for some criteria, we pick a second criterion r_2. If in the ensuing sequence, call it π_{12}, local setup becomes necessary due to some other criterion r_3, we know we could have used it to bridge differences in any other criterion at the same time, in particular differences in r_1 and r_2.

This suggests the following algorithm to build a smarter π from π_{12}: We choose a fixed criterion r_3 different from r_1 and r_2. Then, we add coils to π in the order of π_{12}, skipping those before which local setup would ensue in π due to differences in r_3. After reaching the end of π_{12}, we add the first unused coil from π_{12} to the end of π and repeat the process regarding only the coils in π_{12} which have not been added to π yet.

We can now create different individuals for the initial population by different choices of r_1, r_2, and r_3, and each individual is likely to avoid local setup due to these criteria within certain parts of the sequence, while utilizing unavoidable scrap coils to bridge differences in multiple criteria at once. Thereby, we obtain a broad variety of completely different individuals, each composed of sections which are attractive regarding their own part of the objective. They turn out to provide a particularly diverse basis for constructing sequences with low overall cost by simple crossover and mutation operations.

Finally, as mentioned before, optimizing a sequence w.r.t. local cost only amounts to solving an asymmetric traveling salesman problem, which we can solve optimally in most cases by the Lin-Kernighan-Helsgaun (LKH) heuristic (Helsgaun 2000). We add an individual corresponding to this mostly optimal tour to the initial population as well, before completing it with a number of random sequences.

During a run of our algorithm, we maintain a constant population size across generations. For each individual, we solve the scheduling subproblem as described in the next section in order to assess its cost. Individuals with better makespans survive. Mutations are conducted by inverting a random consecutive subsequence of an individual. For crossovers, we implement a classic mechanism for sequencing problems originally proposed by Mühlenbein et al. (1988), which we shall describe only briefly:

Upon the selection of two individuals from the current population, a *donor* d and a *receiver* r, a random consecutive subsequence of random length is taken from the donor to form the basis for the construction of the new individual, or *offspring* s. We complete s by continuing from its last element i with the elements following i in the receiver, until we encounter an element already contained in s. Now we switch back to the donor and continue completing s from d in the same way, going back to r again when encountering an element already added to the offspring. If we can continue with neither r nor d, we add to s the next element from r which is not in s yet, and try again.

Naturally, algorithmic parameters like population size or the amount of crossovers and mutations performed per generation can have a great effect on a genetic algorithm's performance, and good settings have to be determined through extensive testing. We provide details on the parameter settings used in our implementation in Sect. 7.1.

5.2 Scheduling

Given an individual from our genetic algorithm, i.e., a fixed processing order π for coils, we now solve the scheduling subproblem in order to obtain a complete solution to the coil coating problem. We first describe how an optimal concurrent setup schedule and tank assignment can be computed by a dynamic programming approach which runs in polynomial time as long the number of coaters considered is constant. In order to satisfy the strict runtime requirements of the application, we then proceed to devise a fast heuristic based on ideas from the former exact algorithm. Finally, we describe an intuitive tank assignment rule which was previously in use at SZFG, and to which we compare our results in Sect. 7.2.

5.2.1 Optimal Solutions via Dynamic Programming.

We now utilize the dynamic programming algorithm computing a maximum weight independent set in an m-composite 2-union graph from Thm. 2 to solve the integrated tank assignment and concurrent setup scheduling problem in polynomial time. There are two main issues to be addressed. Since every possibility to perform a concurrent setup job needs to be represented as a work interval, we need to argue that the number of savings rectangles we need to consider in order to obtain an optimal plan is polynomially bounded. These rectangles will be the nodes of the m-composite 2-union graph. Also, as described in Sect. 3.2, a coater interval may have two work intervals associated with it, one for changing a color, and one for changing a roller. Hence, we have to deal with a generalized notion of 2-intervals, and we have to demonstrate that this does not harm correctness or runtime of the dynamic program.

The following lemma bounds the number of savings rectangles which need to be considered in order to obtain an optimal plan.

Lemma 3 *There is a set D of at most*

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

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. 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. Since preemption is not allowed, 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 $\pi = id \in \Pi_n$, we now 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

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

□ □

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

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

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, and due to the fixed sequence, the number of coater intervals is bounded by n^2.

In realistic instances, the maximum length of a coil is roughly twice the time to perform a single concurrent setup task, so this ratio is roughly two. The above bound is also very coarse, since it takes into account all possibilities to pair *any* work interval with a coater interval, while only work intervals completely contained in it are really relevant. Theoretically however, extreme cases, which would have many coils with extremely short processing time followed by a very long coil, cannot be ruled out.

It remains to show that the dynamic program can handle our generalized notion of 2-intervals, namely the case where two disjoint work intervals are associated with a coater interval. Recall that in the proof of Thm. 2, a feasible state in the dynamic program is an m-tuple

$$S_v = (v_1, \ldots, v_m),$$

where each component v_i is either empty or a savings rectangle, and all v_i form an independent set. Two states are compatible if the union of their components still forms an independent set, and two compatible states are adjacent in the state graph if in each component, their savings rectangles are identical, or their coater intervals are disjoint. Edges in the state graph are weighted with the savings gained by added savings rectangles, so any longest path from the empty state corresponds to a maximum weight independent set, i.e., a selection of pairwise compatible savings rectangles of maximum cumulated weight.

Clearly, none of these steps is affected by whether none, one, or two work intervals are associated with a coater interval—additional work intervals merely need to be taken into account when checking for feasibility and compatibility of states. The key property that work intervals are subintervals of their coater interval remains valid regardless.

Hence, by applying Thm. 2, we have an algorithm computing an optimal schedule for a sequence of n coils in time

$$\mathcal{O}((n^2(N+1)^2)^m),$$

which is polynomial in n as long as m, the number of shuttle coaters, is constant and $(\max_{i\in[n]} p_i/\lambda\, t_c)$ is bounded by a polynomial in n.

5.2.2 Heuristic Solutions.

The dynamic programming approach is obviously not feasible for practical use due to its runtime. Nevertheless, we can use its ideas to define a fast heuristic algorithm based on our independent set model for concurrent setup scheduling. At the end of this section, we also introduce the simple scheduling rule in use at SZFG so far.

Independent Set Heuristic. The complexity of our exact algorithm stems from the need to consider interval selections for all coaters simultaneously in order to ensure that savings from all selected work intervals can be realized by the scarce work resource. Intuitively, the probability that indeed all of the corresponding concurrent setups can be scheduled feasibly, i.e., one setup at a time, increases with the length of the associated coater interval. This is our heuristic's core idea for computing good tank assignments.

Instead of considering all coaters at once, as in the exact algorithm, we consider them separately. Recall that savings from coater intervals for different coaters can be realized independently in any case, since they mainly correspond to savings from keeping a color or a roller for later use. Now, instead of explicitly adding work intervals I_c, I_r to each coater interval I, we upper bound cost savings $w_{\text{work}}(I_c, I_r)$ from I's potential work intervals according to how much concurrent setup work for one coater we expect to be able to schedule during I, and add this value to the weight $w_{\text{coat}}(I)$ of I. More precisely, we define the new weight of a coater interval as

$$w'_{\text{coat}}(I) := w_{\text{coat}}(I) + \min\{\alpha|I|, |I_c| + |I_r|\},$$

where $\alpha \in [0,1]$ is a parameter. When choosing $\alpha = 0$, concurrent setup work is assumed impossible. For the contrary extreme $\alpha = 1$, all potential concurrent setup work is assumed to be scheduled for each coater interval.

With these new modified weights, it suffices to consider coater intervals alone. As a consequence, similar to the case of sufficient work resources mentioned in Sect. 3.4, computing a tank assignment T reduces to finding a maximum weight independent set in a regular interval graph, which can be dealt with very efficiently. In order to compute a feasible concurrent setup schedule for this tank assignment, we use an earliest-deadline-first strategy as a simple scheduling rule. As before, each global setup task $w \in W(\pi, T)$ is associated with the coil i it is performed 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 in T; 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.

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. For scheduling the tasks in the resulting set $W(\pi, T)$, the same earliest-deadline-first rule as above is used.

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. 6, 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.

As a final note, we also experimented with local search heuristics on tank assignments. Once π and T were computed for an individual as above, we would compute concurrent setup schedules for T as well as for certain neighboring, i.e., slightly modified tank assignments and choose the best resulting complete solution. However, in all but the fewest cases, T's neighbors failed to improve the final solution compared to T, so this approach was eventually abandoned.

6 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. 5.2.2). 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 β, 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 is given implicitly.

6.1 An Integer Program: Concatenating Short Subsequences

The logic of the model is to assign subsequences of β coils each to $\lceil n/\beta \rceil$ *time slots*, each consisting of β consecutive positions, except the last which may possibly be shorter. For a subsequence s, we denote by $t(s) \in \{1, \ldots, \lceil n/\beta \rceil\}$ its time slot, and by $p(s, j) \in \{1, \ldots, n\}$ the absolute position of coil j in subsequence s. We subsume all possible subsequences of β 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 (Barnhart et al. 1998, Desrosiers and Lübbecke 2005).

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 $s_{\mathrm{loc}}(i, j)$ 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. 5.2.2), 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.

$$\min \sum_{s} c_s z_s + \sum_{i,j \in [n]} c_{i,j} y_{i,j} \tag{4}$$

$$\begin{aligned}
\sum_{p \in [n]} x_{p,j} &= 1 && j \in [n] && (5)\\
\sum_{j \in [n]} x_{p,j} &= 1 && p \in [n] && (6)\\
\sum_{\substack{s \ni j:\\ p(s,j)=p}} z_s &= x_{p,j} && p, j \in [n] && (7)\\
x_{\mathrm{last}(t),i} + x_{\mathrm{first}(t+1),j} &\leq 1 + y_{i,j} && i,j \in [n],\ t = 1, \ldots, \lceil n/\beta \rceil - 1 && (8)\\
x_{p,j} &\in \{0,1\} && p, j \in [n] && (9)\\
y_{i,j} &\in \{0,1\} && i, j \in [n] && (10)\\
z_s &\in \{0,1\} && s \in \mathcal{S} && (11)
\end{aligned}$$

The constraints are interpreted as follows. Each coil gets coated exactly once (5), each position is filled exactly once (6), and a subsequence s needs to have coil j in position p if and only if the corresponding $x_{p,j}$ indicates so (7). The precedence constraints (8) 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 last(t) in t and coil j in the first position 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 (4) 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 could 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.

6.2 Pricing

Initially, the integer program (4)–(11) 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 (7) are the only ones which contain these variables; as a consequence the corresponding dual variables $\mu_{p,j} \in \mathbb{R}$ are the only relevant ones for calculating their reduced cost (which must be negative in order to profitably add the variable to the problem). The reduced cost of z_s is

$$\bar{c_s} = c_s - \sum_{j \in s} \mu_{p(s,j),j} \ , \tag{12}$$

and the pricing problem is to find a subsequence of minimum, or at least negative, 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 the pricing 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.

6.3 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 problem which are explicitly present in this extensive column generation formulation 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 (5), one for each coil.

7 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, while developing our algorithms we experimented on instances provided by planners at SZFG, and in the 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.

The test instances provided by SZFG fall in two categories. The first comprises sets of coils for a planning horizon of up to 72 hours. These instances are solved well ahead of time for long-term planning. The second category is made up of coil sets for a shorter time horizon of at most 24 hours. These instances usually occur when unplanned changes in demand or availability of coils necessitate short-term replanning. A typical batch for such short planning horizon consists of 20–40 coils, while batches for long-term planning comprise up to 120 coils.

For many test instances, SZFG provided a plan devised by their expert planners for comparison with our solutions. In most cases we were able to obtain unexpected improvements over these

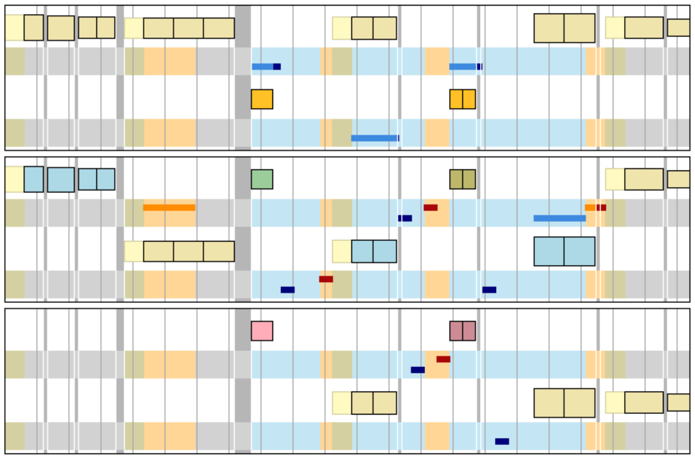


Figure 7: Detail of the visualization of an integer optimal solution to our integer program with subsequences with $\beta = 4$ coils each. Each block represents one shuttle coater with its two tanks; the dark rectangles reflect the coils in the coating sequence. It is clearly visible that the integer 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.

plans. We cannot report on the precise numerical characteristics of our data and results due to non-disclosure agreements, hence normalized numbers are presented.

7.1 Some Implementation Details

As stated before, the runtime limit prescribed by our industry partner was 180 seconds. 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. For most short-term instances, our algorithm finds its best solutions after less than 30 seconds, while solutions keep improving towards the runtime limit for long-term planning.

In the construction heuristic for the initial population described in Sect. 5.1, we use all possible three-tuples of width, height, processing speed, and temperature in primer and finish oven as criteria r_1, r_2, and r_3, accounting for about one third of our initial population of 200 individuals. We keep population size constant across generations, and in each generation create 100 new individuals each from both mutations of random individuals and crossovers of two randomly chosen parents. This results in a total population of 400, of which 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 the worse 90% of the population and replace it with a new initial population. This usually leads to further improvement of the best individuals in subsequent generations.

Regarding scheduling, we performed extensive testing using the heuristics described in Sect. 5.2.2. We report on results for the independent set heuristic for different values of the parameter $\alpha \in [0, 1]$ in comparison to the simpler FIFO rule.

For lower bound computations, we implemented a branch-and-price algorithm to solve the integer program (4)–(11) within the publicly available SCIP framework (Achterberg 2007). Its implementation is not tuned to performance as we used it as a proof-of-concept only, so we do not report computation times for lower bounds (which were huge).

7.2 Interpretation of Results

Much to SZFG's planners' surprise, our algorithm produces plans with makespan reductions of up to 25%, on average over 13%, compared to reference solutions actually gone to production in both long-term and short-term planning, cf. Figs. 8 and 10. Expert planners also assert that our solutions "look very different" from theirs while retaining operability. Together with the fact that significant savings are also realized over manual plans which "looked optimal," we take this as evidence that our rigorous mathematical analysis of the savings potential of shuttle coaters fully paid off. It should also 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. Hence, cost savings are now realized to their full extent in day-to-day production.

7.2.1 Long-Term Instances

As expected, our independent set heuristic for setup scheduling proves superior to the simpler FIFO online rule. We were unable, however, to find a uniform choice of the parameter α suitable for all instances alike. When determining the best setting for α by trying all values in $\{0.1, \dots, 0.8\}$, the graph heuristic outperformed FIFO on 12 of the 16 instances, reducing setup cost by up to 30% (over FIFO). This translates to makespan savings of up to 6%, cf. Fig. 8. When fixing α to 0.5, the independent set heuristic remains similarly superior to FIFO on 8 instances, while incurring an increase in makespan of at most 1% in four cases, cf. Fig. 9.

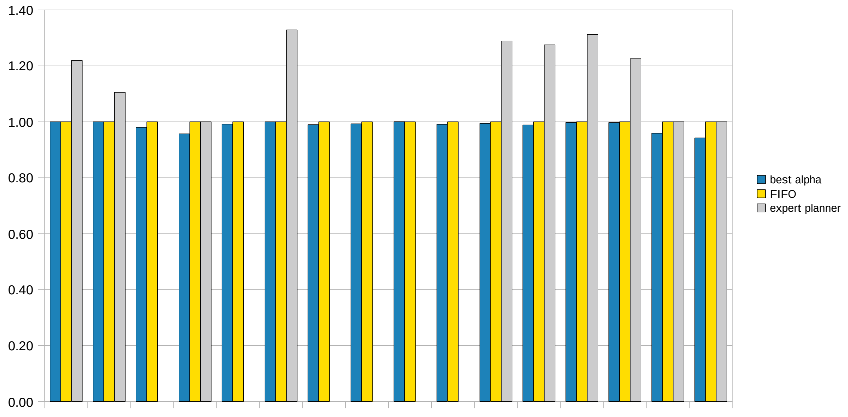


Figure 8: Comparison of normalized makespans for representative long-term instances. From left to right, makespans are for our solutions using the independent set heuristic with optimal choice of parameter α, our solutions using the FIFO online rule, as well as for a reference solution devised by an expert human planner where it was available.

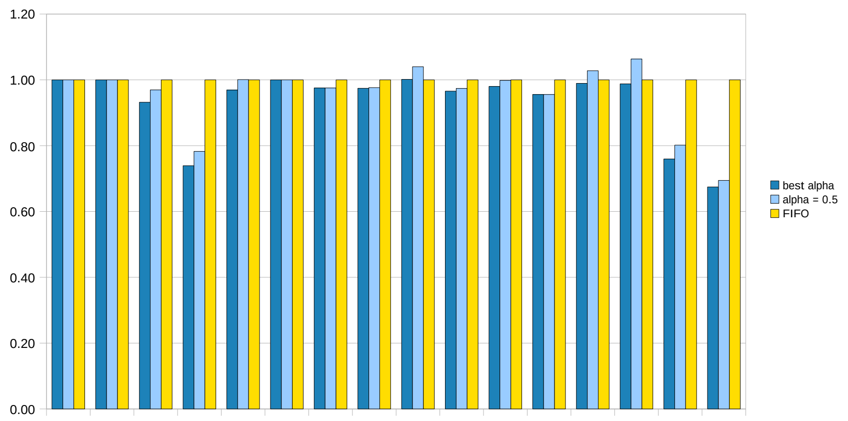


Figure 9: Comparison of normalized non-productive time included in our long-term solutions. From left to right, costs are for our algorithm when using the independent set heuristic with optimal choice of α, with fixed choice of $\alpha = 0.5$, and when using the FIFO online rule.

7.2.2 Short-Term Instances

For all short term instances, we succeeded in computing lower bounds by our branch-and-price approach, proving our solutions to be within at most 10% of makespan optimality, cf.Fig. 10. Yet, we did not solve all instances to integer optimality and also used short subsequences only ($\beta \leq 6$), so the lower bound is certainly improvable. We are thus convinced that the actual quality of our heuristic solutions is even better than currently proven.

If we concentrate on setup cost—in contrast to makespan—it can be seen from Fig. 11 that the integer programming lower bound is able to close much more of the gap to the upper bound than the TSP bound which is optimal w.r.t. local setup only. This illustrates again why our integrated consideration of sequencing and scheduling is indeed the key to successful optimization in this application.

Finally, the superiority of the independent set heuristic to FIFO is less significant in short-term planning. While both heuristics were on a par for most instances, small improvements over FIFO were observed in three cases.

8 Summary and Conclusions

We have developed an exact mathematical model for the complex integrated sequencing and scheduling task in coil coating with shuttles and implemented optimization software solving it in practice. Our approach fulfills all requirements regarding speed and robustness for its application in the production environment. Our model takes into account all relevant aspects of production reality, including the subtasks of sequencing the coils, and scheduling the color tanks and the scarce work resources to perform setup work. 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, Germany.

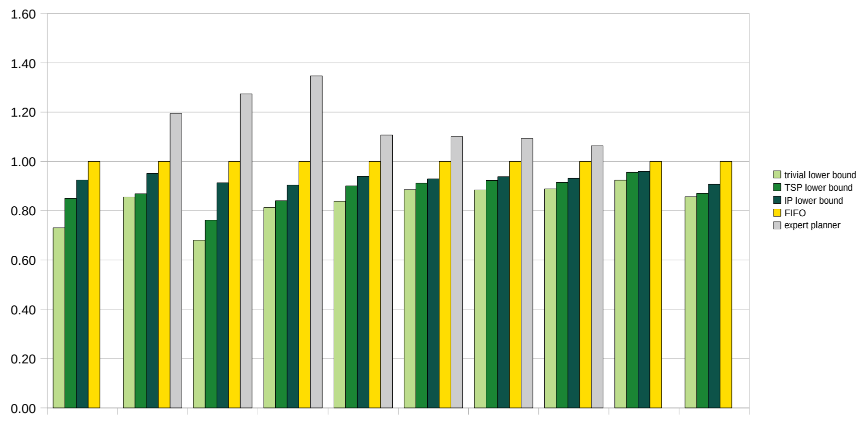


Figure 10: Comparison of normalized bounds and makespans for representative short-term instances when restricted to FIFO tank assignment. Bounds displayed from left to right are LB_{triv}, the sum of coil processing times, LB_{TSP}, the sum of processing times plus the local cost of an optimal, local cost based TSP solution, and our IP bound LB_{IP}. Makespans are given for our solutions obtained using the FIFO online rule for scheduling, as well as for a reference solution devised by an expert human planner where it was available.

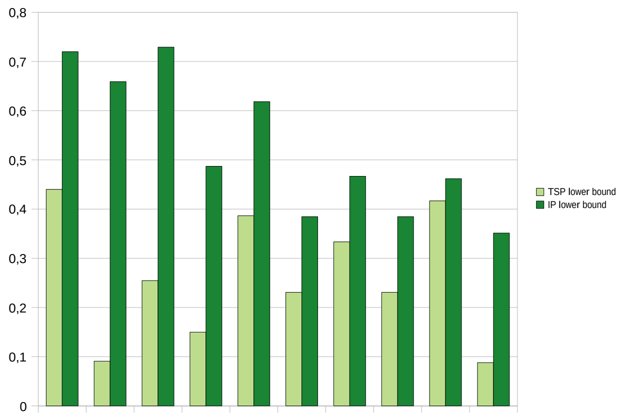


Figure 11: Percentage of the closed relative gap between the respective trivial lower bound and the optimum non-productive time included in our short-term FIFO solutions: Our bound from the branch-and-price approach respects global setup cost and closes considerably more of the gap than the TSP based bound which only considers local setup cost.

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 finally got fully acquainted with operating their machines only through our theoretical investigations.

For the scheduling subproblem, we have developed a graph theoretic model allowing for the fast computation of optimal solutions in an environment where sufficient work resources are available. For the present resource-constrained case, this model leads to a heuristic algorithm which is significantly superior to the simpler FIFO rule, in particular for long-term instances.

Finally, we have developed and implemented an integer programming model of a combinatorial relaxation of the problem, which we are able to solve via branch-and-price for short-term planning instances. The solutions yield lower bounds on the optimal makespan of our short-term instances when using an online tank assignment rule, proving that our heuristic solutions have an optimality gap of no more than 10%. Our optimization module has been in use for day-to-day planning at Salzgitter Flachstahl GmbH since December 2009.

References

Aarts, E., J.K. Lenstra, eds. 1997. *Local Search in Combinatorial Optimization*. John Wiley & Sons.

Achterberg, T. 2007. Constraint Integer Programming. Ph.D. thesis, Technische Universität Berlin. `http://opus.kobv.de/tuberlin/volltexte/2007/1611/`.

Bar-Yehuda, R., M.M. Halldórsson, J. Naor, H. Shachnai, I. Shapira. 2002. Scheduling split intervals. *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 732–741.

Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations Research* **46**(3) 316–329.

Butman, Ayelet, Danny Hermelin, Moshe Lewenstein, Dror Rawitz. 2007. Optimization problems in multiple-interval graphs. *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 268–277.

Cook, S.A. 1971. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing*. 151–158.

Delucchi, M., A. Barbucci, G. Cerisola. 1999. Optimization of coil coating systems by means of electrochemical impedance spectroscopy. *Electrochimica Acta* **44**(24) 4297 – 4305.

Desrosiers, J., M.E. Lübbecke. 2005. Selected topics in column generation. *Operations Research* **53**(6) 1007–1023.

Gupta, U.I., D.T. Lee, J. Y.-T. Leung. 1982. Efficient algorithms for interval graphs and circular-arc graphs. *Networks* **12** 459–467.

Gyárfás, A., J. Lehel. 1969. A Helly type problem in trees. *Combinatorial Theory and its Applications*, *Coll. Math. Soc. J'anos Bolyai*, vol. 4. 571–584.

Helsgaun, K. 2000. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European J. Oper. Res.* **126**(1) 106–130.

Kaiser, Tomás. 1997. Transversals of d-intervals. *Discrete & Computational Geometry* **18**(2) 195–203.

Meloni, C., D. Naso, B. Turchiano. 2003. Multi-objective evolutionary algorithms for a class of sequencing problems in manufacturing environments. *IEEE International Conference on Systems, Man and Cybernetics* **1** 8–13. doi:10.1109/ICSMC.2003.1243784.

Meuthen, B., A.-S. Jandel. 2005. *Coil Coating*. Vieweg+Teubner.

Mühlenbein, H., M. Gorges-Schleuter, O. Krämer. 1988. Evolution algorithms in combinatorial optimization. *Parallel Computing* **7** 65–85.