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: September 28, 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 colors necessitate time-consuming setup work. In most coating stages one can choose between two parallel color tanks. This can either reduce the number of setups needed or enable setups concurrent with production. 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.

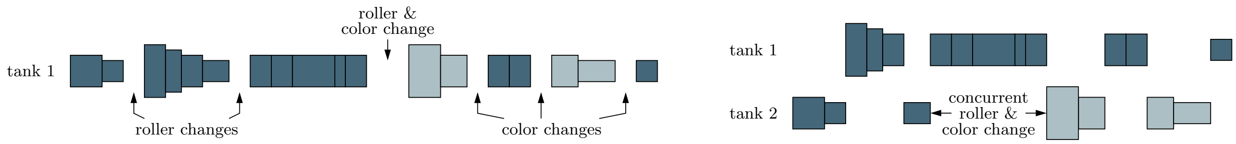
We present an optimization model for this integrated sequencing and scheduling problem. A core component is a graph theoretical model for concurrent setup scheduling. It is instrumental for building a fast heuristic which is embedded into a genetic algorithm to solve the sequencing problem. The quality 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; e.g., for the melt in the blast furnace or when rolling slabs in the hot and cold rolling mills to coils, to state only two examples. In all of these problems, however, *sequencing* is only one part of devising a complete production plan: For a given processing sequence, a *schedule* for performing certain tasks on workpieces or machinery also needs to be computed. The nature of this scheduling problem often depends largely on the processing sequence, hence the quest for good sequences can

*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 a different coating or is wider than its predecessor on that tank.

only be answered accurately by taking into account the cost of the ensuing scheduling problem as well. 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: In addition to sequencing coils, a challenging *concurrent setup scheduling* problem needs to be solved, and both problems strongly interdepend.

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, and the sheet metal used for car bodies already has an anti-corrosion coating before it arrives at the automotive plant for pressing. Already in the 1960s, associations were formed to promote the evolution of coil coating[1]. 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, coil coating has been investigated from chemical and engineering perspective in several publications, e.g., Delucchi et al. (1999), Deflorian et al. (2000), Duarte et al. (2005), Zhang et al. (2009).

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, modern coil coating lines are equipped with so-called *shuttle coaters* or *quick (color) change coaters*, offered by different manufacturers as standard machinery; see e.g., GFG Peabody, Bronx International, or SMS Siemag[2]. Shuttle coaters possess two separate tanks allowing for holding two different coating materials at the same time. The advantage is twofold: The shuttle can be used to switch between two coatings on the same coater at (essentially) no setup cost, or alternatively the unused tank can be set up while coating continues from the other tank in the meantime. We refer to this possibility to perform setup work *during production*, which is somewhat uncommon in scheduling literature, as *concurrent setup*; see Fig. 1 for an illustrative example.

Literature regarding optimization in the planning process for coil coating is scarce at best: To the best of our knowledge, only Tang and Wang (2009) consider planning for a coil coating line. They apply tabu search to a rather basic model without shuttle coaters. The present work is the first incorporating shuttles and concurrent setup work in a thorough mathematical investigation.

Our work is concerned with a somewhat standard coil coating line as operated by many major steel companies worldwide, consisting of a primer and a finisher, each comprising two coaters to coat the top and bottom side of a coil, cf.Meuthen and Jandel (2005), Delucchi et al. (1999). Each of the coaters may or may not be a shuttle coater, whose introduction significantly changes the

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

[2]`http://www.gfg-peabody.com/`, `http://www.bronxintl.com/`, `http://www.sms-siemag.com/en/1577.html`

flavor and the complexity of production planning: 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).

In order to precisely capture the scheduling step, we introduce the *concurrent setup scheduling* problem, which also fits other applications involving concurrent setup. By relating it to the independent set problem in certain generalized interval graphs called 2-union graphs, we obtain a dynamic program running in polynomial time for any fixed number of shuttle coaters, while proving NP-hardness when this number is part of the input. Moreover, we prove corresponding results for the independent set problem in special 2-union graphs, which are of interest in their own right.

The dynamic program inspires a fast and good heuristic for concurrent setup scheduling, which we embed into a genetic algorithm for sequencing. Altogether, we develop a 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 which we solve by branch-and-price.

Our algorithm has been added to PSI Metals' planning software suite[3], and is currently in use on a coil coating line with shuttle coaters 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[5]. Since most setup cost calculations are incorporated into our methods as a black box, our algorithm can be adapted easily to other coil coating lines with different setup rules and a different number of shuttle coaters.

2 Problem Formulation

Steel coils are a highly individualized product, and all non-productive time in coil coating depends on certain characteristics of coils. They usually have a length of 1–5 km, and their central attributes are naturally the coatings they receive in the four coating stages, chosen from a palette of several hundreds, and their width, usually 1–1.8 m. More intuitively, we will henceforth refer to coating materials as colors. We refrain from listing further coil attributes, since their list is very long; yet all relevant information is included in our calculations.

For a concise description of the optimization task, we shall briefly familiarize the reader with some coil coating terminology:

- 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.

[3] `http://www.psimetals.de/en/`
[4] `http://www.salzgitter-flachstahl.de/en/`
[5] This success has made this contribution a finalist of the 2009 EURO Excellence in Practice Award.

- 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. A *roller change* needs to be performed 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 former coil, the coating of the latter would bear imperfections.

Before entering production, each coil is unrolled and stapled to the end of its predecessor. In order to bridge non-productive time during setups, 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. 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. A schematic view of a typical coil coating line is depicted in Fig. 2.

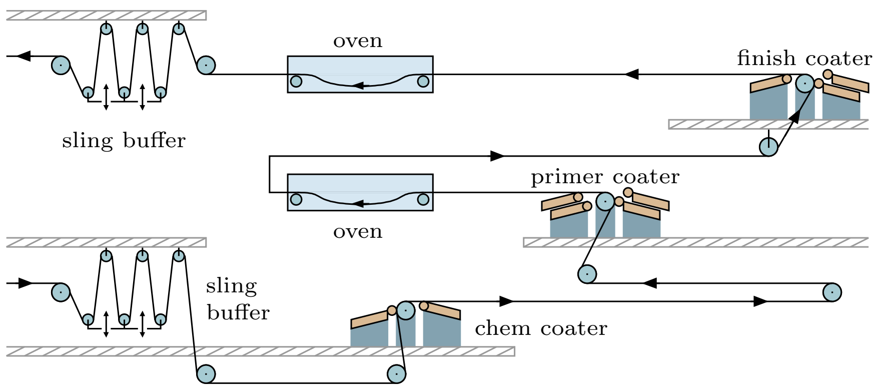


Figure 2: Schematic view of a coil coating line with chem, primer, and finish coater. Here, the chem and the bottom finish coaters are standard coaters, the remaining have shuttles.

An instance of our optimization problem comprises a set $[n] := \{1, \dots, n\}$ of coils to be coated. For the account in this paper, each coil j is characterized by its colors $c_j^{(k)} \in \mathbb{N}$ on coaters $k = 1, \dots, m$, its width $w_j \in \mathbb{R}^+$, and its processing time $p_j \in \mathbb{R}^+$, the time it takes for j to pass any given point on the coating line. Note that even though a coil passes the different stages of the coating line one after the other, it is feasible to think of a coil as being processed at all stages at once, as the time it takes a section of a coil to get from one stage to the next is negligible compared to coil lengths.

The optimization goal is to minimize the *makespan* for coating the given set of coils, i.e., the completion time of the last coil in the sequence. This is essentially equivalent to minimizing non-productive time, or *cost*, in the plan, which ensues for two reasons:

Transition Coils. In order to satisfy certain technical restrictions, transition coils of different lengths, widths etc. may be required in between two consecutive coils in the sequence. The duration of transition coils necessary for each pair of coils $(i, j) \in [n] \times [n]$ if run consecutively is explicitly specified in an instance as $s_{\text{loc}}(i, j)$, and we refer to it as *local cost*.

Figure 3: Optimum tank assignment for a fixed sequence of coils. The setup right before the last coil—none, in this example—depends on its predecessor on the tank, i.e., on the whole sequence.

Setups. Setups comprise color or roller changes. Both require roughly the same amount of time denoted by t_c. If coils $i, j \in [n]$ are run consecutively *on the same tank* of a coater k, setups of total length $s^{(k)}(i,j) = s^{(k)}_{cc}(i,j) + s^{(k)}_{rc}(i,j)$ have to be performed in between, comprising the sum of durations of a possible color and roller change, where

$$s^{(k)}_{cc}(i,j) = \begin{cases} t_c & \text{if } c^{(k)}_i \neq c^{(k)}_j, \\ 0 & \text{otherwise,} \end{cases} \qquad s^{(k)}_{rc}(i,j) = \begin{cases} t_c & \text{if } w_i < w_j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Due to the dependency of consecutive coils on a tank, it does not suffice to consider pairs of consecutive coils in the sequence in order to quantify necessary setup work. Larger sets of coils need to be taken into account—the entire sequence in the worst case, see Fig. 3. Hence, we refer to non-productive time ensuing due to setup work as *global cost*.

Finally, 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* for the set of setups $W = W(\pi, T)$ defined by π and T as described above. We elaborate on the structure of feasible schedules in the following section.

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 setups to be scheduled depends on T and π.

2.1 Concurrent Setup Scheduling

We will now focus on the subproblem of computing a schedule for a given sequence π. Deciding on a tank assignment T settles the predecessors of coils on the tanks, yielding the set $W = W(\pi, T)$ of setups to be performed. These can either be scheduled between coils during non-productive time, or *concurrently* to production on an idle tank. While the former results in an increase in makespan, the latter does not. Consequently, there are two possibilities to save cost by good scheduling: On the one hand, the tank assignment can preserve used colors and/or a rollers on an idle tank for later coils, thereby reducing the number of setups to be performed. On the other, setups can be performed concurrently with production on idle tanks, thereby keeping setups from increasing the makespan.

Setup schedules have to respect certain constraints in order to be feasible: While there may be several work teams to perform setups, at most one team can work on the same coater concurrently to production to ensure safety. Contrarily, during non-productive time, the teams work together to finish work as fast as possible. Moreover, concurrent setups are not allowed while transition coils are run, as transitions require careful monitoring. Finally, the execution of one concurrent setup may not be preempted by another. Note that as a result of these restrictions, transition coils can be ignored when scheduling concurrent setups.

In order to capture this scheduling problem more formally, we define the *Concurrent Setup Scheduling* (CSS) problem as follows. Consider a production process comprising m *cells*, each of which being equipped with two identical, interchangeable *tools*. A sequence of n jobs needs to be processed in a fixed order $\pi \in \Pi_n$, where each job j has to be processed for p_j time units utilizing one tool on each of the m cells simultaneously. If two jobs i and j are processed in consecution on the same tool of cell k, setup work taking time $s^{(k)}(i,j) \in \mathbb{R}_{\geq 0}$ has to be performed on the tool after job i is completed and before job j starts.

There are r *setup resources* available to perform setup work in two distinct ways: If performed during non-productive time, all r resources work together as one fast resource, achieving a speedup factor $\lambda \geq 1$. Alternatively, a setup can be performed *concurrently* to production on an idle tool by one of the setup resources, eliminating its impact on the makespan. Partial concurrent setups are possible and remaining setup work is finished during non-productive time, incurring a speedup as above.

A feasible solution to the CSS problem assigns one of the two tools to each job for every cell, and schedules all resulting setups, such that at most r concurrent setups are performed at a time during production, and only one setup at once during non-productive time. Concurrent setups may only be preempted by setups during non-productive time. The objective is to determine a plan (π, S) minimizing the *makespan*. Note that since all setup resources work together during non-productive time, the makespan of a solution is insensitive to deferring non-concurrent setups to the latest possible time. Hence, it suffices to only schedule concurrent setups explicitly.

In our application, coaters resemble the cells, and tanks are our tools. The number of setup resources is $r = 1$, i.e., there is one team of workers performing setups. Also, setup times have a special structure, cf.(1). Nevertheless, all of our results hold for CSS in general, i.e., arbitrary setup times and any r.

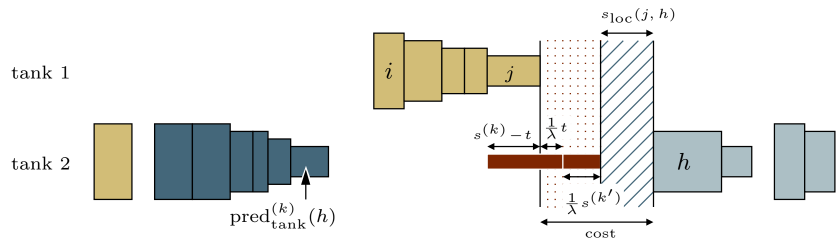
Moreover, our model and algorithms for CSS remain valid even in a more general setting where each setup task can only be performed by a subset of the resources. However, for the sake of a clear presentation and cleaner notation, we omit further comments on this case.

2.2 Cost Computation

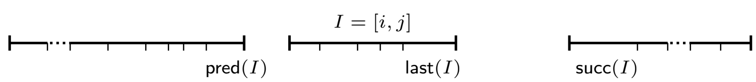
The cost of a coil coating plan decomposes into time for transition coils and setups during non-productive time. As cost computation is quite complicated, an illustrative example is given in Fig. 4(a). Again, concurrent setups incur no cost as they do not impact the makespan, and no setup may be performed during transition coils.

More formally, consider a sequence $\pi \in \Pi_n$ and a schedule $S = S(\pi)$ for π. Denote by $\mathrm{pred}^{(k)}_{\mathrm{tank}}(j)$ the predecessor of coil j on the same tank of coater k in S, and denote by $C(S)$ the total amount of concurrent setup in S. Then formally, the cost of plan (π, S) is given by

$$\sum_{j=2}^{n} s_{\mathrm{loc}}\big(\pi(j-1), \pi(j)\big) \;+\; \frac{1}{\lambda}\left(\sum_{k=1}^{m}\sum_{j=2}^{n} s^{(k)}\left(\mathrm{pred}^{(k)}_{\mathrm{tank}}(j), j\right) \;-\; C(S(\pi))\right),$$



(a) Picture of a partial schedule on coater k; $s^{(k)}$ is short for $s^{(k)}\left(\mathrm{pred}^{(k)}_{\mathrm{tank}}(h), h\right)$ in this figure. When coil j is finished, a period of non-productive time begins: Color and roller need to be changed on tank 2 of this coater, and this is only in part done concurrently, then finished during non-productive time. Additionally, more setups before h have to be performed on a different coater k', and this is done during non-productive time as well. All setups which are not concurrent incur speedup λ. Finally, a transition coil of length $s_{\mathrm{loc}}(j,h)$ needs to be run between coils j and h, and no setup can be performed during this time.



(b) Since transition coils do not allow concurrent setup, they can be ignored for scheduling. Thus, their length is not included in the length of intervals in our model.

Figure 4: Components of cost and tool intervals.

the sum of the duration of all transition coils and setup performed non-concurrently.

2.3 Related Work

Sequencing and scheduling problems with setup times and makespan minimization constitute a widely studied field, see Allahverdi et al. (2008) for a recent survey. In most of these problems, however, setups are purely *local*, i.e., depend only on two successive jobs on the same machine and thus are closely related to asymmetric traveling salesman problems, see Balas et al. (2008). Also more general problems with setups typically involve only local setups. One such example are *scheduling problems with communication delays and precedence constraints*, see Bampis et al. (1997), where a delay only occurs between pairs of jobs with a non-transitive precedence constraint among them when they are scheduled on different machines.

In contrast, our setup costs need to be calculated in view of several, possibly many preceeding jobs. Such *non local* setups have only sporadically been considered in scheduling, e.g., by Koulamas and Kyparisis (2008). They are more typical in multi product assembly lines. But here one no longer considers sequencing problems with makespan minimization, but aims at finding batches of products that minimize the cycle time of the robotic cells, see e.g., Rekieck et al. (2000).

Setups concurrent with production also occur in *parallel machine scheduling problems with a common server* considered by Hall et al. (2000): Before processing, jobs must be loaded onto a machine by a single server which requires some time (the setup time for that job) on that server. Good sequences of jobs on the parallel machines minimize the time jobs have to wait to be setup due to the single server being busy on a different machine. In this model, too, setups are purely local, and once sequences for the machines have been determined, scheduling the server constitutes a trivial task.

Altogether, we are not aware of any papers on sequencing or scheduling problems that come close to the combined sequencing and concurrent setup scheduling problem studied here. Our model and results may thus pave the way for other applications involving sequencing and concurrent setup scheduling as they may occur in complex production environments such as the automotive industry.

3 Interval Model for Concurrent Setup Scheduling

In our solution approach, which we describe in Sect. 5, we repeatedly solve instances of the CSS problem resulting from different sequences of coils. Hence, an efficient algorithm for CSS is of key importance. We develop a representation of solutions as a family of weighted 2-dimensional intervals, where the first dimension is related to a tool assignment and the second to performing concurrent setup work. We call two such intervals, or axis-parallel rectangles, *independent*, if their projections onto neither of the axes intersect. An optimal solution S to CSS 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 a fast heuristic algorithm and a dynamic programming approach for our application, yielding optimal coil coating plans 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 jobs to the same tool of a cell, and every resulting possibility to perform concurrent setup work, as weighted intervals called *tool intervals* and *setup intervals*, respectively. Then, we combine tool and setup 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 tool assignment and the concurrent setup work performed, compared to processing all jobs on the same tool without performing concurrent setup work.

setup intervals of all cells compete for the r available resources for concurrent setup work, i.e., a chosen setup interval for one cell impacts the choice of setup intervals on other cells. In contrast, we can choose tool intervals of different cells independently. We position the rectangles in the plane accordingly, such that there is a one-to-one correspondence between solutions to CSS on the one hand, and maximal independent sets of rectangles on the other. We denote by p_j the processing time of job j, and throughout this section, we assume that the jobs are numbered as they appear in the fixed sequence, i.e., $\pi = id \in \Pi_n$.

3.1 Tool Intervals

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

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

We call the last job before and the first job after I its predecessor $\mathsf{pred}(I)$ and successor $\mathsf{succ}(I)$, respectively. Also, let $\mathsf{last}(I)$ denote the last job in I. The weight $w^{(k)}_{\mathrm{coat}}(I)$ of a tool interval I of cell k comprises the change in cost before $\mathsf{succ}(I)$ resulting from processing all jobs in I with one tool and $\mathsf{pred}(I)$ and $\mathsf{succ}(I)$ with the other, as compared to running all these jobs from the same tool (and performing all necessary setup work before $\mathsf{succ}(I)$ during non-productive time). Thus, we have

$$w^{(k)}_{\mathrm{coat}}(I) \;=\; \frac{1}{\lambda}\Big(s^{(k)}\left(\mathsf{last}(I),\, \mathsf{succ}(I)\right) - s^{(k)}\left(\mathsf{pred}(I),\, \mathsf{succ}(I)\right)\Big)\,.$$

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

3.2 Setup Intervals

A setup interval I_s is always associated with a tool interval $I = [i, j]$, hence also with a particular cell k. Furthermore, it is tied to one of the r setup resources. It represents concurrent setup work performed from $\mathsf{pred}(I)$ to $\mathsf{succ}(I)$ on k's idle tool during jobs $i, \ldots, j$ by a certain setup resource. A setup interval's weight $w_{\mathrm{work}}(I_s)$ equals its length and describes the amount of concurrent setup work performed. So clearly, we always have

$$w_{\mathrm{work}}(I_s) \;\leq\; \min\Big\{p(I),\; s^{(k)}\left(\mathsf{pred}(I),\, \mathsf{succ}(I)\right)\Big\}\,.$$

3.3 Saving Rectangles

We now define *saving rectangles* $R = (I, I_s)$ as a combination of a tool interval I and a (possibly empty) setup interval I_s. The total weight of a saving rectangle R is

$$w(R) \;=\; w_{\mathrm{coat}}(I) \;+\; \tfrac{1}{\lambda}\, w_{\mathrm{work}}(I_s).$$

As for tool intervals, this weight may be negative.

Recall that a feasible solution to CSS shall correspond to the selection of a maximum weight independent subset of all possible saving rectangles. Quite naturally, both dimensions of a saving rectangle, i.e., tool and setup intervals, are associated with a sense of time; see Fig. 5. We now make use of this fact in order to position rectangles in the plane such that their projections onto one of the axes intersect, if and only if the savings in makespan represented by their weight *conflict*. i.e., cannot be realized jointly.

Two saving rectangles conflict if and only if one of the following holds:

- They belong to the same cell and their tool intervals intersect: Tool intervals of one cell, whose intersection contains a job j, lead to conflicting tool usage, since both intervals assign j to a different tools.
- They belong to the same setup resource and their setup intervals intersect: Each resource can perform only one setup at a time.

Consequently, when positioning rectangles in the plane, all rectangles associated with the same cell receive their own section of the x-axis, while all rectangles sharing the same setup resource are placed in a distinct section of the y-axis. Within sections, all rectangles are positioned in x and y direction according to their natural sense of time, see Fig. 5 for an illustration.

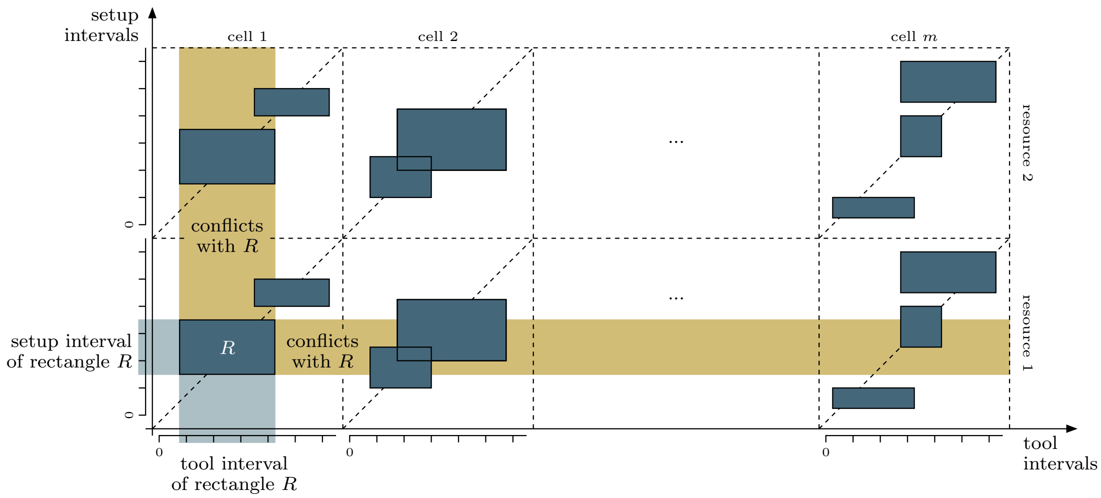


Figure 5: Some saving rectangles, appropriately positioned in the plane. Selecting rectangle R, i.e., running all jobs belonging to its tool interval on the same tool of cell 1 and performing concurrent setup during its setup interval, prohibits the selection of any rectangle whose projection onto one of the axes intersects that of R. The selection of non-disjoint tool intervals prevents a proper tool assignment, while intersecting setup intervals violate resource constraints. For simplicity, only two resources are visualized.

Now, any subset of pairwise independent saving rectangles, whose tool intervals cover all jobs on all cells, naturally corresponds to a feasible concurrent setup schedule with corresponding tool assignment. Moreover, if such subset has maximum weight, the corresponding schedule is optimal: It realizes the greatest savings possible as compared to running all jobs on the same tool.

By the construction above, saving rectangles also have a very specific structure. We build upon these properties when analyzing the independent set problem ensuing from CSS in Sect. 4:

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

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

Finally, note that for general instances of CSS, it may not be tractable to explicitly model all possibilities to perform concurrent setup work in an application, as the number of setup intervals could become super-polynomially large. In the case where all setup durations have a sufficiently large common divisor, however, we can prove a polynomial bound on the number of setup intervals that need to be considered for an optimal solution.

Lemma 1 *Given an instance of CSS, let τ denote the greatest common divisor of all setup durations $s^{(k)}(i,j)$. Then, there is a set D of at most*

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

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

Proof. Proof. We explicitly construct a set of potential start points sufficient for an optimal solution and then bound their number. In an optimal solution, setup intervals may of course start with every but the last job in the sequence, accounting for the first $n-1$ points necessary. Given a potential start point for a setup interval of length s, also the point exactly s after it is a potential start point. Since preemption is not allowed, any optimal selection of saving rectangles can be transformed to a solution where all setup intervals start at a point of the above form by shifting every selected setup interval to the left as far as feasibly possible.

Assuming w.l.o.g. that the fixed sequence of jobs is $\pi = id \in \Pi_n$, we now observe that moving through the sequence in order, the number of new potential start points during the processing of job i is bounded by $i(p_i/\tau)$. Thus, the total number of potential start points for setup intervals necessary in an optimal solution is

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

□ □

Thereby, the number of savings rectangles which need to be considered for an optimal solution is bounded by $k \cdot r \cdot n^2(N+1)$, the number of possibilities to pair each of the n^2 possible tool intervals of each cell with no or one setup interval of every setup resource. Note that N is polynomial in n, as long as the ratio $(\max_{i\in[n]} p_i/\tau)$ is bounded polynomially in n. This bound is also very coarse, since it takes into account all possibilities to pair *any* setup interval with a tool interval, while only setup intervals completely contained in it are really relevant.

In our application the duration of all setups is a multiple of t_c, hence Lem. 1 applies with $\tau = t_c$. In realistic instances, the maximum length of a coil is roughly twice t_c, so the number of setup intervals necessary for each cell is no more than $2n^2$.

3.4 Building Schedules from Maximal Independent Sets

Given an independent set of saving rectangles such that every job is covered by a tool interval on every cell, the construction of a feasible concurrent setup schedule with corresponding tool assignment is straight-forward: The schedule for concurrent setup work is given explicitly by setup intervals, interrupted by non-concurrent setups and transition coils, and at the end of each tool interval, we switch tools on the corresponding cell. 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.,

$$c(\mathcal{R}) \;=\; \sum_{i\in[n-1]} \left(s_{\mathrm{loc}}(i,i+1) \;+\; \sum_{k\in[m]} \tfrac{1}{\lambda}\, s^{(k)}(i,i+1) \right) \;-\; \sum_{R\in\mathcal{R}} w(R),$$

where the first part is the cost for running all jobs from the same tool, depending only on $\pi = id$.

Moreover, note that negative rectangle weights can be eliminated by adding a sufficiently large constant to the weight of each rectangle, weighted by its length. This does not change the structure of the problem, and we force any maximum weight independent set to be maximal such that the selected rectangles have tool intervals covering all jobs.

Finally, our model yields a fast algorithm for CSS when sufficient setup resources are available, i.e., concurrent setup can be performed simultaneously on all cells.

Lemma 2 *In the special case when $r \geq m$, CSS can be solved in polynomial time.*

Proof. Proof. By assigning a different resource to each cell, we can assume that two saving rectangles conflict if and only if they belong to the same cell and their tool intervals intersect. Thus, an optimal solution can be computed independently for each cell, and the problem reduces to finding maximum weight subsets of pairwise disjoint intervals (i.e., maximum weight independent sets in classic interval graphs), which can be done very efficiently (Gupta et al. 1982). □ □

We shall comment on the complexity status of CSS for all other cases of r and m in Sect. 4.

4 Independent Sets in Special 2-Union-Graphs

We now make a connection from the CSS problem 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.

Tool and setup 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 tool interval's dimension corresponding to the m different cells.

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 usual 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 $G = G_2$.

Considering the rectangles in CSS, property (3) is always satisfied, since setup intervals are subintervals of their tool intervals according to the natural sense of time. Due to separate resource sections along the y-axis, this holds for any number of resources.

First, we show that for variable m, the maximum independent set problem remains *NP*-hard for m-composite 2-union graphs, even in the unweighted case. Afterwards, we proceed to describe an exact dynamic programming approach running in polynomial time for any fixed m. Finally, we describe how both negative and positive results carry over to CSS.

Theorem 3 *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.

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.

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 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). $\square$ $\square$

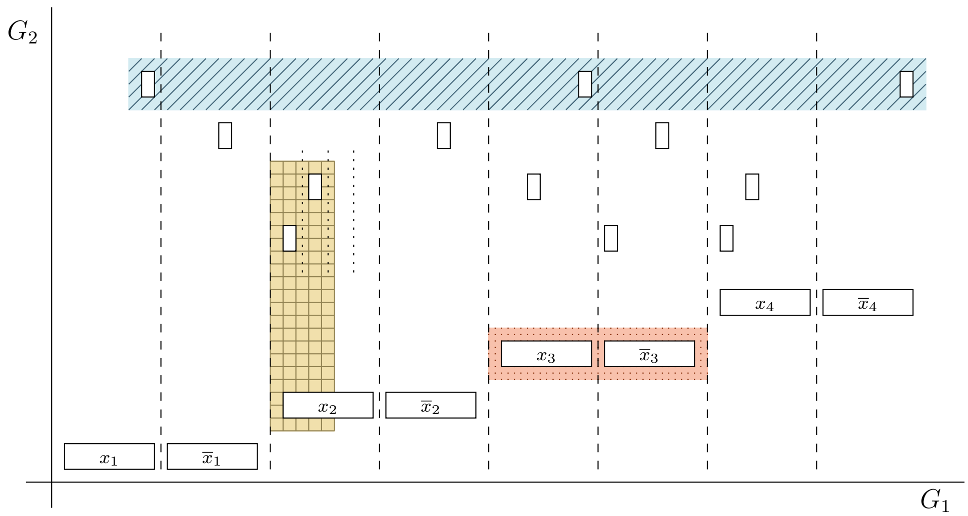


Figure 6: Two-dimensional interval representation of the graph constructed from the 3-SAT instance with four variables and clauses $(\overline{x}_2 \vee x_3 \vee \overline{x}_4)$, $(\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4)$, $(x_1 \vee x_2 \vee x_3)$, $(\overline{x}_1 \vee \overline{x}_3 \vee x_4)$. 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.

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. It is not immediately clear, however, that hardness of this problem entails hardness of CSS: It remains to show that the reduction can be performed such that the resulting interval representation is the interval model of a CSS instance. We give the proof of the following theorem in the appendix.

Theorem 4 *For the number of cells m part of the input, CSS is* NP*-hard for any fixed number $r < m$ of setup resources.*

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 is a constant.

Theorem 5 *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 $L_1, \ldots, L_m$ the minimal sections of the line containing all intervals in $C_1, \ldots, C_m$ 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

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

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 $P = (S_s, \ldots, S_t)$ 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 $e_L(u_i) \leq e_L(a_j)$, and from the fact that $S_a \cup S_v$ forms an independent set and $a_j \neq v_j$, $e_L(a_j) < s_L(v_j)$. Finally, this results in $e_L(a_j) < s_L(v_j) \leq e_L(u_i) \leq e_L(a_j)$, 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. 3), this approach may well be the best possible in terms of efficiency. Moreover, our interval model for CSS always leads to m-composite 2-union graphs. Hence, together with Lem. 1, we immediately obtain the following.

Corollary 1 *For an instance of CSS, let τ denote the greatest common divisor of all setup durations $s^{(k)}(i,j)$. When the number of cells m is fixed in advance, all instances of CSS where*

$$\frac{\max_{i \in [n]} p_i}{\tau}$$

is polynomial in the size of the input can be solved in polynomial time, even if the number of setup resources r is part of the input.

5 Algorithm

We now describe how we produce 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: Planners at SZFG require a plan covering 24–72 hours to be computed within 180 seconds. This is why we propose a special purpose heuristic, which is based on the insights gained in the previous sections.

Mainly due to global cost, an effective algorithm cannot rely 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.

5.1 Sequencing

We utilize a genetic algorithm for sequencing, which by its nature combines solutions, or *individuals*, in such a way that beneficial characteristics of solutions persist, while costly characteristics are eliminated (Aarts and Lenstra 1997). The set of solutions is commonly referred to as *population.* 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*, the population is first enlarged through crossovers and mutations, before a subset of all these individuals is selected for survival. See Meloni et al. (2003) for a recent successful application to a different production sequencing problem.

A key to the success of our algorithm lies in the construction of its initial population, which is highly diverse w.r.t. different beneficial aspects. 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 transition coils for the initial population.

Transition coils 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. For simplicity, we have omitted these criteria in our problem formulation in Sect. 2. This local cost incurred between two coils i and j has the following structure:

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

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

When considering only one single criterion, we may be able to avoid local cost completely by sorting the set of coils according to it, minimizing the largest occurring difference between subsequent coils. On the other hand, by the definition of s_{loc}, *one* set of transition coil can be used to bridge differences in *several* criteria at the same time, making the most of some unavoidable transition coils, 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}, transition coils becomes necessary due to some other criterion r_3, we know we could have used them 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 which would cause local cost 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 cost due to these criteria within certain parts of the sequence, while utilizing unavoidable transition 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.

Finally, optimizing a sequence w.r.t. local cost only amounts to solving an asymmetric traveling salesman problem, for which we use the Lin-Kernighan-Helsgaun (LKH) heuristic (Helsgaun 2000). We add an individual corresponding to an often 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):

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.

5.2 Scheduling

Given an individual from our genetic algorithm, i.e., a fixed processing order π for coils, we now solve the resulting instance of CSS in order to obtain a complete solution to the coil coating problem. Indeed, Cor. 1 yields a polynomial algorithm for this subproblem in our application. However, this 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.

5.2.1 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 setup intervals can be realized by the scarce work resource. Intuitively, the probability that concurrent setup work on different cells can be scheduled feasibly, i.e., one setup at a time, increases with the length of the associated tool interval. This is our heuristic's core idea for computing good tank assignments.

Instead of considering all coaters at once, we consider them separately. Recall that savings from tool intervals for different coaters can be realized independently in any case. Now, instead of explicitly adding a setup interval I_s to each tool interval I, we upper bound cost savings $w_{\text{work}}(I_s)$ from I's potential setup 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

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

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

With these new modified weights, it suffices to consider tool 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 an 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 setup $w \in W(\pi, T)$ 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 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 w with the earliest deadline for which $t \in [r_w, d_w]$.

5.2.2 Online Rule: First-in First-out (FIFO).

Finally, we have a look at the tank assignment rule which was previously in use at SZFG: 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 rule is its *online* character (and thus simplicity): the choice of tank depends only on the current and the previous coil; but this rule may produce suboptimal tank assignments.

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). 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 cost only. This reduces the coil coating problem to determining an optimal sequence with respect to local cost—which can be formulated as a (small) asymmetric traveling salesman problem, thus we denote the obtained bound by LB_{TSP}.

Global 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 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 cost. By means of an integer linear program we find a cheapest such concatenation. 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 σ, we denote by $t(\sigma) \in \{1, \ldots, \lceil n/\beta \rceil\}$ its time slot, and by $p(\sigma, j) \in \{1, \ldots, n\}$ the absolute position of coil j in subsequence σ. We subsume all possible subsequences of β coils in the set $\mathcal{S}$. We slightly abuse the set notation $j \in \sigma$ to express that coil j is contained in subsequence σ. 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 $z_\sigma \in \{0,1\}$ indicates whether subsequence $\sigma \in \mathcal{S}$ is selected or not. Each $\sigma \in \mathcal{S}$ has its local plus global cost c_σ which is computed as if all coaters were entirely clean and empty at the beginning of σ. Local cost $s_{\text{loc}}(i,j)$ between coils i and j is abbreviated $c_{i,j}$.

$$\min \sum_{\sigma} c_\sigma z_\sigma + \sum_{i,j \in [n]} c_{i,j} y_{i,j} \tag{4}$$

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

Note that in order to compute global 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.

The constraints are interpreted as follows. Each coil gets coated exactly once (5), each position is filled exactly once (6), and a subsequence σ 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 local 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, we would obtain a lower bound on the makespan for *any* tank assignment rule. At the moment, this idea is computationally intractable.

6.2 Pricing

Initially, the integer program (4)–(11) contains all $x_{p,j}$ and $y_{i,j}$ variables, but only some z_σ 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). All other z_σ 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_σ is

$$\bar{c}_\sigma = c_\sigma - \sum_{j \in \sigma} \mu_{p(\sigma,j),j} \ ,$$

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 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. Therefore, a straight forward depth first search enumerates the $O(n^\beta)$ subsequences to solve the pricing problem. A pruning criterion based on the dual variable values of coils not yet added to a subsequence 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

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

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

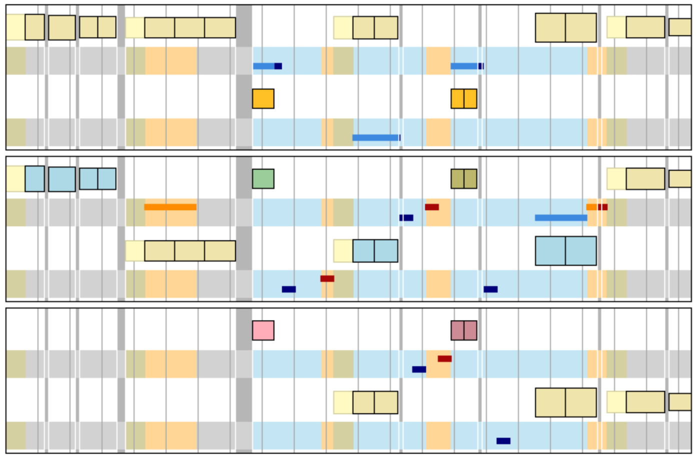


Figure 7: Detail of the visualization of an integer optimal solution to our integer program with subsequences with $\beta = 4$ coils each. Each block represents one shuttle coater with its two tanks; the dark rectangles reflect the coils in the coating sequence. It is clearly visible that the integer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

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 go live in an existing production environment, we were provided with realistic data right from the beginning. Planners at SZFG repeatedly validated our solutions, and addressed various issues regarding the accuracy of cost calculations and the practicability of work schedules, which were eventually solved.

The test instances fall in two categories. The first comprises sets of up to 120 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 20–40 coils 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 re-planning.

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 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

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 of 180 seconds for long-term planning.

In the construction heuristic for the initial population (Sect. 5.1), we use some coil properties which were omitted in the problem formulation in Sect. 2 for simplicity. All possible three-tuples of width, height, processing speed, and temperature in primer and finish oven are taken as criteria r_1, r_2, and r_3, accounting for about one third of our initial population of 200 individuals. In each generation we 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 worst 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. 5.2.1 and 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

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 $\{0.1, \ldots, 0.8\}$, the graph heuristic outperformed FIFO on 12 of the 16 instances, reducing 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.

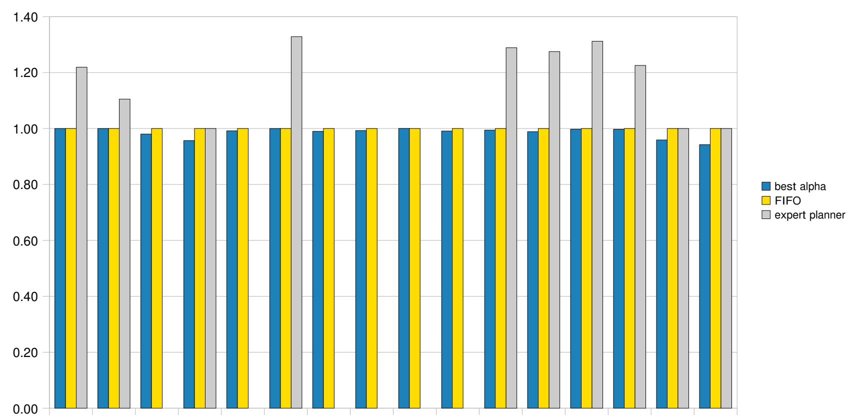


Figure 8: Comparison of normalized makespans for representative long-term instances. From left to right, makespans are for our solutions using the independent set heuristic with optimal choice of parameter α, our solutions using the FIFO online rule, as well as for a reference solution devised by an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

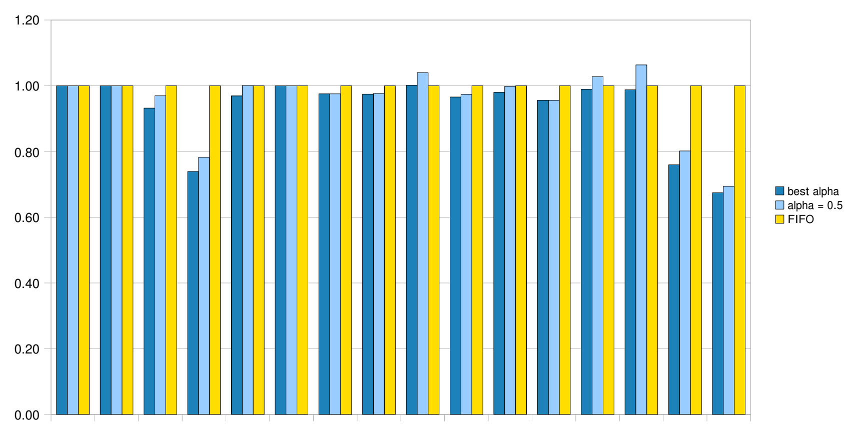


Figure 9: Comparison of normalized non-productive time included in our long-term solutions. From left to right, costs are for our algorithm when using the independent set heuristic with optimal choice of α, with fixed choice of $\alpha = 0.5$, and when using the FIFO online rule.

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.

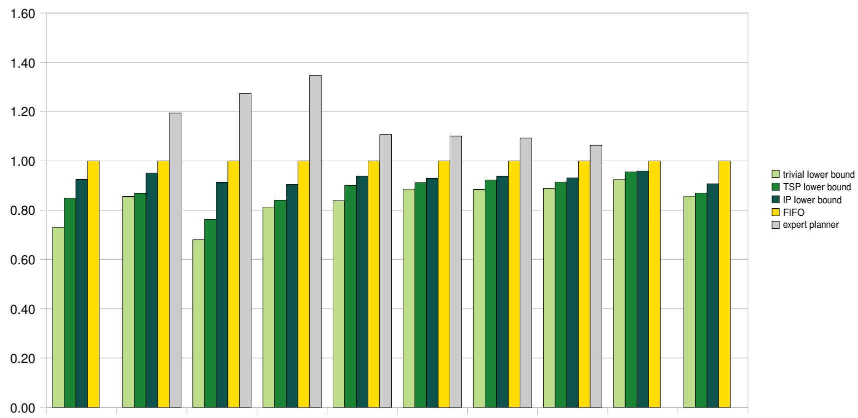


Figure 10: Comparison of normalized bounds and makespans for representative short-term instances when restricted to FIFO tank assignment. Bounds displayed from left to right are LB_{triv}, the sum of coil processing times, LB_{TSP}, the sum of processing times plus local cost in an optimal, local cost 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.

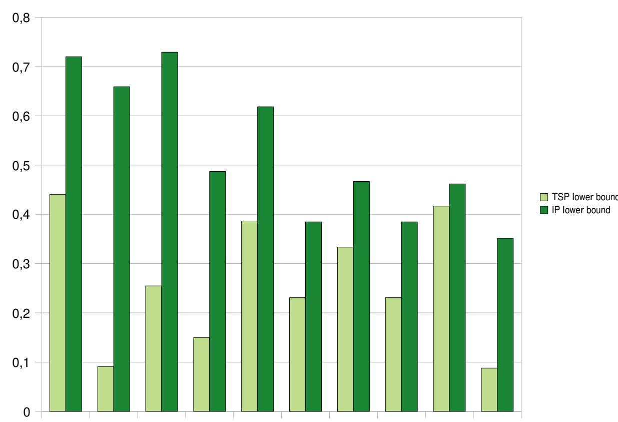


Figure 11: Percentage of the closed relative gap between the respective trivial lower bound and the optimum non-productive time included in our short-term FIFO solutions: Our bound from the branch-and-price approach respects global cost and closes considerably more of the gap than the TSP based bound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

If we concentrate on 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 cost only.

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.

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.

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.

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

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

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

Allahverdi, Ali, C.T. Ng, T.C.E. Cheng, Mikhail Y. Kovalyov. 2008. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research* **187**(3) 985 – 1032. doi:DOI:10.1016/

j.ejor.2006.06.060. URL http://www.sciencedirect.com/science/article/B6VCT-4MBBYT9-9/%2/1234fe3cd53aa7985be808af7cfe7893.

Balas, Egon, Neil Simonetti, Alkis Vazacopoulos. 2008. Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling* **11**(4) 253–262.

Bampis, Evripidis, Frédéric Guinand, Denis Trystram. 1997. Some models for scheduling parallel programs with communication delays. *Discrete Appl. Math.* **72**(1-2) 5–24.

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*. Association for Computing Machinery, New York, NY, USA, 151–158.

Deflorian, F., L. Fedrizzi, S. Rossi. 2000. Effects of mechanical deformation on the protection properties of coil coating products. *Corrosion Science* **42**(7) 1283–1301.

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.

Duarte, R.G., A.C. Bastos, A.S. Castela, M.G.S. Ferreira. 2005. A comparative study between cr(vi)-containing and cr-free films for coil coating systems. *Progress in Organic Coatings* **52**(4) 320–327.

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*, *Collections of the Mathematical Society J'anos Bolyai*, vol. 4. 571–584.

Hall, Nicolas G., Chris N. Potts, Chelliah Sriskandarajah. 2000. Parallel machine scheduling with a common server. *Discrete Applied Mathematics* **102**(3) 223–243. doi:10.1016/S0166-218X(99)00206-1.

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

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

Koulamas, Ch., G.J. Kyparisis. 2008. Single-machine scheduling problems with past-sequence-dependent setup times. *European Journal of Operational Research* **187**(3) 1045–1049. doi:DOI:10.1016/j.ejor.2006.03.066. URL http://www.sciencedirect.com/science/article/B6VCT-4M7VB41-D/%2/150293fa8d68a882b97ac061db5b76db.

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.

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

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

Rekieck, Brahim, Pierre De Lit, Alain Delchambre. 2000. Designing mixed-product assembly lines. *IEEE Transactions On Robotics And Automation* **16**(3) 268–280.

Schaefer, T.J. 1978. The complexity of satisfiability problems. *Proceeding of the 10th Annual ACM Symposium on Theory of Computing*. 216–226.

Tang, Lixin, Xianpeng Wang. 2009. Simultaneously scheduling multiple turns for steel color-coating production. *European Journal of Operational Research* **198**(3) 715–725.

Zhang, W., R. Smith, C. Lowe. 2009. Confocal raman microscopy study of the melamine distribution in polyester-melamine coil coating. *Journal of Coatings Technology and Research* **6**(3) 315–328.

A Proof of Theorem 4

In this hardness proof, we follow the main ideas from the proof of Thm. 3. Recall, that in this proof, the reduction from 3SAT was leading to an instance of the maximum independent set problem in m-composite 2-union graphs with very few rectangles. However, it is no clear how this instance corresponds to CSS. Hence, further investigations are necessary to show its hardness. Our proof is divided into four major steps: In Sect. A.1, we consider the special structure of m-composite 2-union graphs resulting from CSS instances. In general, these graphs contain a very high number of rectangles. In Sect. A.2, we investige a special CSS instance with only one cell, for which we show that it is sufficient to concentrate on a very limited set of rectangles in the corresponding 2-union graph. This graph will be of similar structure as the one in the proof of Thm. 3. In the main part of the proof in Sect. A.3, we reduce One-in-3SAT to CSS with one resource. Finally in Sect. A.4, we generalize our result to an arbitrary number of resources.

Throughout the proof, we speak of an interval being contained in another according to the natural sense of time, i.e., referring to the time axis starting at the beginning of each section; see Fig. 5.

A.1 Structure of m-Composite 2-Unions Graphs for CSS Instances

The m-composite 2-union graphs corresponding to instances of CSS exhibit a special structure: All possible tool intervals for the given sequence π need to be considered for each of the m cells. Moreover, for each tool interval I, we have to take into account all setup intervals of length

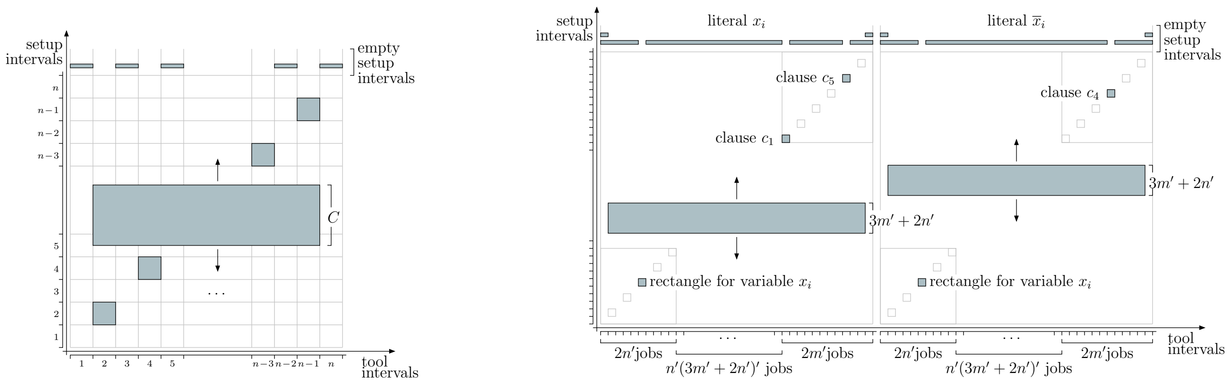
$$w_{\mathrm{work}}(I) = \min\left\{p(I),\ s^{(k)}\left(\mathsf{pred}(I),\, \mathsf{succ}(I)\right)\right\},$$

or shorter if the setup interval is started less that $w_{\mathrm{work}}(I)$ time units before I's end. We also need to consider the case of empty setup intervals. In fact, we can think of all setup intervals belonging to a tool interval I as only one rectangle which can be moved along the setup interval's time axis within the interval I. According to the number of setup resources r, we copy this rectangle representation of the m-composite 2-union graph to r different sections along the y-axis to obtain the final graph for the CSS instance. Note that all our argumentation also holds when considering only discrete start times of the setup intervals as given in Lem. 1. Still, for simplicity, we assume them to be arbitrary.

As described in Sect. 3.4, by adding a sufficiently large constant to each tool interval, weighted by its length, we can assume that every maximum weight independent set covers every job on every cell. For simplicity, we do not consider these additional weights in the following. Still, we assume that in every maximum weight independent set, all jobs are covered by a tool interval on every cell.

A.2 Zero-Cost Decision Variant for a Special Instance of CSS

We consider a zero-cost decision variant of CSS for a special instance with odd number of jobs n and $m = r = 1$. Later, we will use our investigations on this simple instance to construct a more elaborate one in our hardness proof. We assume $\pi = id$. For any even job j, we have processing



(a) Necessary saving rectangles for one cell of the instance from Sect. A.2 when considering the zero-cost decision variant of CSS. For simplicity, $p_j = 1$ for all j.

(b) Cells $2i - 1$ and $2i$ in CSS instance represent literals x_i and $\overline{x}_i$ from instance of One-in-3SAT. Grey rectangles mark positions of variable and clause rectangles on other cells. In this example, the clauses c_1 and c_5 contain literal $\overline{x}_i$, and c_4 contains x_i.

Figure 12: Instances of CSS which are used in the proof of Thm. 4.

time $p_j = 1$, and for odd j arbitrary $p_j \geq 1$. Setup costs are specified by

$$s(i,j) \;:=\; \begin{cases} 0 & \text{, if } j = i+1 \text{ and } i \neq 1, n-1 \\ M & \text{, if } j = i+1 \text{ and } i = 1, n-1 \\ 0 & \text{, if } j = i+2 \text{ and } i \text{ is even} \\ 1 & \text{, if } j = i+2 \text{ and } i \text{ is odd} \\ C & \text{, if } i = 1 \text{ and } j = n \\ M & \text{, otherwise} \end{cases}$$

with $C > 0$ and $M > \sum_i p_i$. The variable M is chosen sufficiently large, such that in any zero-cost schedule no setup of length M occurs; the length of any tool interval is simply not long enough to perform this setup work concurrently.

In the following, we will rule out rectangles which are sufficient to be considered when searching for a zero–cost schedule:

- Due to the size of M, we cannot chose any tool interval containing the jobs 1 and 2 or $n-1$ and n. Hence, every zero-cost solution chooses the tool intervals $[1]$ and $[n]$. The setup intervals are empty, since no setup has to be performed at the beginning and at the end.

- The tool interval $[2, n-1]$ can only be chosen if all resulting setup is performed concurrently, i.e., if its setup interval has length C.

- Consider tool intervals $[j]$ for $j = 2, 3, \ldots, n-1$. For odd j, no setup is necessary, and hence, the setup interval is empty. If j is even, a setup interval of length 1 is required. Since in this case $p_j = 1$, there is no flexibility in starting the setup interval.

- All other tool intervals require setup work of length M, and thus, they do not occur in zero-cost solutions.

Summarizing, it suffices to consider the tool interval $[2, n-1]$ and all intervals containing exactly one job, where only $[2, n-1]$ provides some flexibility for concurrent setup work; see Fig. 12(a). This leaves exactly two possible tool assignments: Either, we switch the tool after every job or we only switch after the first job and before the last job.

Finally note, that we can replace an even job i by p_i unit-size jobs $i_1, i_2, \ldots, i_{p_i}$ without changing the set of rectangles necessary to consider. We define the setup cost between consecutive replacement jobs to be 0, and transfer the setup cost to and from i to i_1 and i_{p_i}, respectively. All remaining setup cost, is set to M. With the same arguments as above, it is sufficient to assume that the jobs $i_1, i_2, \ldots, i_{p_i}$ form one interval instead of previously i. Thus, given a sequence of unit-size jobs, any subset $S \subset \{2, 3, \ldots, n-1\}$ with $|i-j| \geq 2$ for any $i \neq j \in S$ defines a CSS instance as described in this section. We interprete the jobs in S as formerly even jobs, and call them *unit rectangle jobs*.

A.3 Reduction of One-in-3SAT to CSS with one resource

We will now show the strong NP-hardness of CSS when he number of cells m is part of the input and when there is only one resource. We reduce from One-in-3SAT, a variant of 3SAT where one asks for a variable assignment, such that in every clause exactly one variable is true. This problem variant is known to be strongly NP-hard (Schaefer 1978). The main ideas of the reduction are taken from the proof of Thm. 3.

Consider an instance I' of One-in-3SAT with n' variables $x_1, \ldots, x_{n'}$ and m' clauses $c_1, \ldots, c_{m'}$. We reduce it to an instance I of CSS with $n := 2(n'+m') + n'(3m'+2n')$ unit-size jobs and $m := 2n'$ cells, one for each literal. Literals x_i and $\overline{x}_i$, $i \in \{1, \ldots, n\}$, correspond to cell $2i-1$ and $2i$, respectively. For each cell, we define the setup cost by specifying the unit rectangle jobs as in Sect. A.2: For x_i and $\overline{x}_i$, we choose job $2i$ on the cells $2i-1$ and $2i$. If literal ℓ is contained in the sth clause, then we choose job

$$2n' + n'(3m'+2n') + 2s - 1$$

on the cell corresponding to $\overline{\ell}$; see Fig. 12(b). We refer to the related rectangles as *variable rectangles* and *clause rectangles*, respectively, and set the setup cost C to $3m'+2n'$. We will show that there is a zero-cost schedule for I if and only if I' is a yes-instance of One-in-3SAT.

Assume, we are given a truth assignment for I'. For any true literal we choose a rectangle based on the tool interval $[2, n'-1]$, and arrange these n' rectangles without setup conflict from job $2n'+1$ til $2n'+1+n'(3m'+2n')$. On all remaining cells, we choose the variable rectangle and all clause rectangles. This does not lead to a conflict, since all of these variable rectangles belong to different variables, and in every clause exactly one literal is true. By Sect. A.2, the chosen rectangles represent a zero-cost schedule for I.

Consider the rectangle representation $\mathcal{R}$ of a zero-cost CSS schedule of I. By the amount of setup cost C, $\mathcal{R}$ cannot contain more than n' rectangles based on the tool interval $[2, n-1]$. Otherwise, not all setup work can be performed concurrently. Assume, that there is a variable for which both such rectangles are chosen. Then, by the above, there in another variable for which no such rectangle is chosen. Since at most one of the corresponding variable rectangles can be contained in $\mathcal{R}$, this leads to schedule with positive cost. Hence, for each variable exactly one rectangle based on $[2, n-1]$ is selected. We set the corresponding literals to true. On the remaining n cells, all variable and clause rectangles must be chosen to allow a zero-cost schedule. This is possible if and only if for each clause exactly one literal is not covered by a $[2, n-1]$-rectangle.

A.4 Generalization for an arbitrary number of resource

We generalize the result of Sect. A.3 for $r > 1$ resources by adding $r - 1$ cells to the reduction above. For these cells, we define the setup costs as follows: Between consecutive jobs, we have cost 0, except after the first and before the last job, where the cost is defined to be M. Between job 1 and n, we set the cost to $n - 2$. In all remaining cases, the cost is set to M. The only option to schedule the jobs on such a cell without cost is to switch the tank after the first and before the last job and to perform concurrent setup in the meantime, blocking one full resource.

Note, that with a similar construction, we can argue that the hardness result of Thm. 3 holds even when there is an arbitrary constant number of independent sections along the y-axis.