Finishing Line Scheduling in the steel industry

H. Okano
A. J. Davenport
M. Trumbo
C. Reddy
K. Yoda
M. Amano

A new solution for large-scale scheduling in the steelmaking industry, called Finishing Line Scheduling (FLS), is described. FLS in a major steel mill is a task to create production campaigns (specific production runs) for steel coils on four continuous processes for a one-month horizon. Two process flows are involved in FLS, and the balancing of the two process flows requires resolving conflicts of due dates. There are also various constraints along the timeline for each process with respect to sequences of campaigns and coils. The two types of constraints—along process flows and timelines—make the FLS problem very complex. We have developed a high-performance solution for this problem as follows: Input coils are clustered by two clustering algorithms to reduce the complexity and size of the problem. Campaigns are created for each process from downstream to upstream processes, while propagating upward the process timings of the clusters. Timing inconsistencies along the process flows are then repaired by scheduling downward. Finally, coils are sequenced within each campaign. The FLS system enabled a steel mill to expand its scheduling horizon from a few days to one month, and to improve decision frequency from monthly to daily.

1. Introduction

In a major steel mill in Japan, production scheduling for steel sheet products, also called *coil products*, was conducted by skilled human experts, and the scheduling horizon was limited to a few days. The steel mill had finished automating the scheduling of primary steelmaking, which covers the upstream processes of coil production, but their problem was how to determine accurate due dates for slabs for a longer horizon in order to utilize the automated scheduler more efficiently. Slabs created by the upstream processes are transformed to hot coils at a hot strip mill (HSM) and to cold coils at a cold mill (CM), and are finished with annealing and galvanizing processes (**Figure 1**). The whole scheduling task for coil production after the HSM, from the CM onward, is called *Finishing Line Scheduling* (FLS). In order to determine accurate due dates for slabs, therefore, the HSM and the finishing lines must be scheduled for a longer scheduling horizon. The steel mill first examined the feasibility of the existing capacity planning tools for the FLS and found them not applicable. The steel mill then created a special-purpose scheduling system, called the FLS system, with the support of an IBM team including the authors.

A schedule for the finishing lines consists of production campaigns[1] on each process and a coil sequence within each campaign. The schedule involves horizontal flows of production campaigns along timelines on each process and vertical flows of coils from upstream to downstream processes (**Figure 2**). For simplicity, in Figure 2, arrows are drawn for only a few pairs of coils, and only two campaign sequences are drawn for each process. There are actually more arrows, and there may be more than two machines of the same type, called *lines*, for each process. Each coil has several properties: width, thickness, length, campaign type, release date, due date, priority, grade, and so on. Among the coil properties, those other than release

[1] The production campaign is a production run with specific start and end times in which coils of a particular type are processed continuously on a process line.

0018-8646/04/$5.00 © 2004 IBM

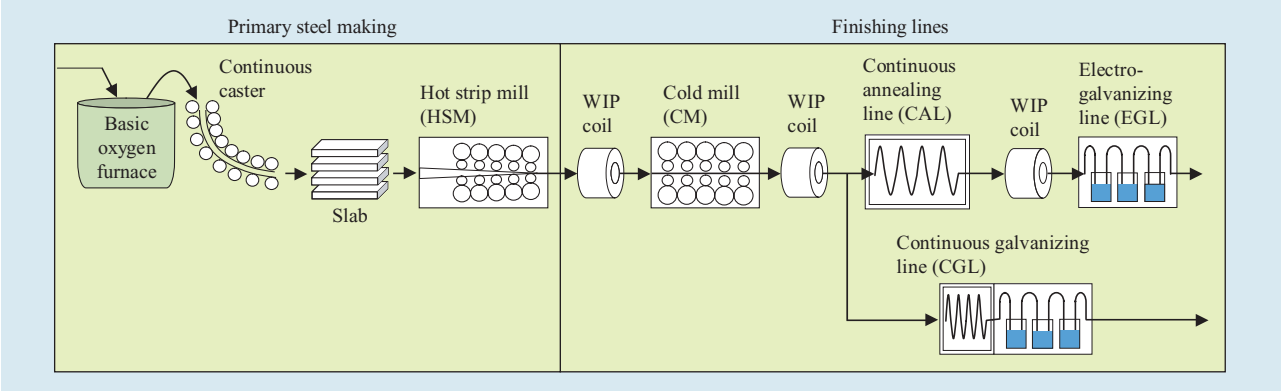

Figure 1

Process flow of primary steelmaking and coil making. Four processes targeted for scheduling in the finishing lines are shown. Work in process (WIP) indicates that there is inventory stored at each location in the flow.

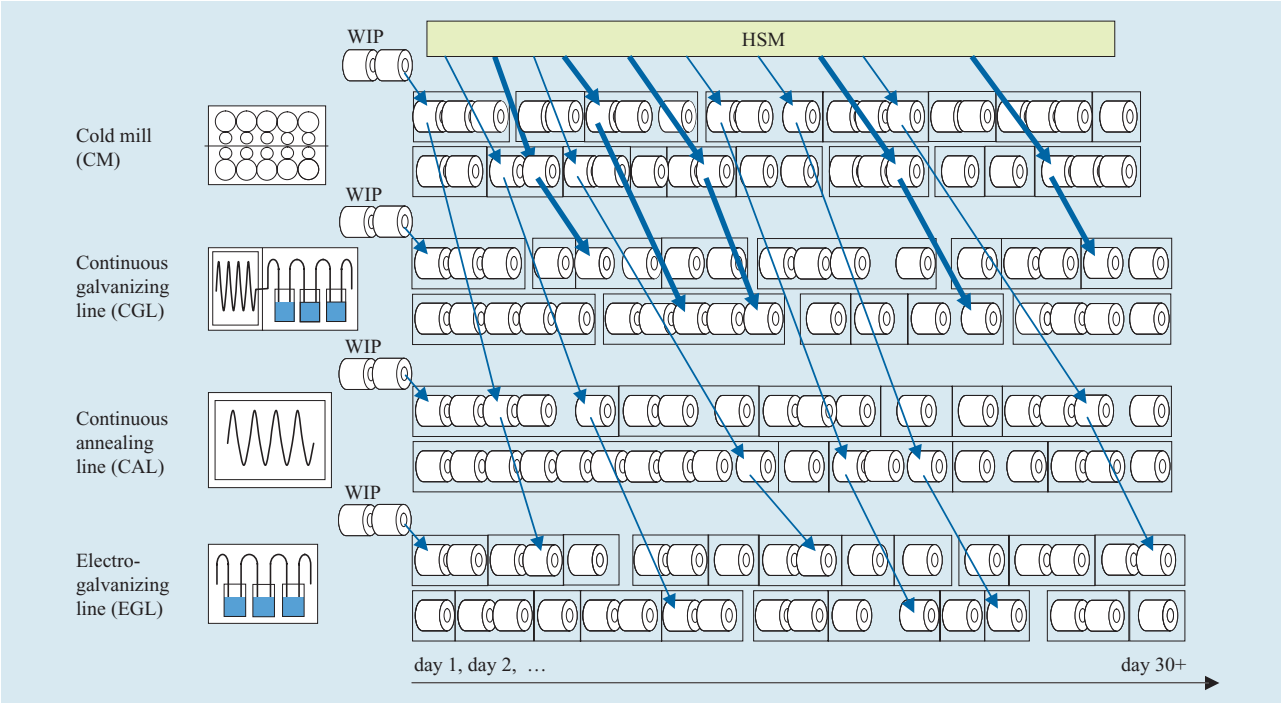

Figure 2

Schematic rendering of schedules for four processes. Rectangles shown for the four processes represent campaigns, and the objects drawn in the campaigns represent coils. Arrows between coils show that they are the same coils which should be processed from top to bottom. Thick arrows represent coils going from CM to CGL, while thin arrows represent coils going from CM to CAL, then to EGL.

and due dates, priority, and grade are all process-dependent. Note that campaign types on one process are not related to those on other processes. As shown in Figure 2, therefore, two coils in the same campaign on CM may (usually) be assigned to different campaigns on lower stream processes. The number of coils assigned to a campaign is typically about 50 to 500. The task of sequencing the coils assigned to each campaign taking into account the several constraints between coils is called the *sequencing* (**Figure 3**). In this task, the *vertical relationships*

should be taken into account, which means that a coil must be processed after it has been through any preceding prerequisite processes. To create campaigns for a particular process, it is usually necessary to create several campaigns of the same type in order to meet the due dates and satisfy the constraints on the minimum and maximum campaign sizes. Therefore, creation of campaigns involves the partitioning of coils into several campaigns with the same type so that the sequencing problems for those campaigns become feasible. Whether or not an instance of the sequencing problem has a feasible solution, in this paper, is called *sequenceability*. The task of creating campaigns for each process and partitioning coils into campaigns so that every campaign is sequenceable is called *campaign allocation*. The vertical relationships should be taken into account in this task as well. There are several types of campaigns, and some campaign types require setups for rollers, galvanizing liquid, and so on. Specific campaign transitions minimizing setup costs are preferred, and some campaign transitions also require minimizing setup times. Taking into account the setup costs and times, the problem requires observing the preferable sequences of campaign types, called the *campaign templates*. The problem also requires creating specific types of campaigns within specific time frames in each month or in each week. Such a time frame is called a *chance*, and the types of campaigns that must be scheduled in chances are called the *chance campaigns*. Note that the campaign templates, chances, and minimum and maximum campaign sizes render the allocation problem so difficult that simple capacity-planning tools which do not consider relationships between campaigns along the timelines cannot be used.

Some papers have been published about HSM scheduling and production scheduling for primary steelmaking involving a continuous caster and HSM. However, searches find no literature specifically on FLS. Therefore, papers on scheduling algorithms for HSM and primary steelmaking and those on job-shop scheduling which are relevant to FLS are reviewed here. Yasuda et al. [1] describe HSM production scheduling that involves decisions on which slabs are rolled in which sequence and the orders to which slabs are assigned. They address a two-stage solution. The first stage generates a rough schedule of width, thickness, and reheating temperature transitions for a campaign, i.e., the interval between roll exchanges. The second stage assigns orders to slabs and sequences the order–slab pairs according to the rough schedule. Fang and Tsai [2] and Lopez et al. [3] address analogous HSM scheduling problems and respectively describe a genetic algorithm and a tabu search approach. The sizes of their problems are a few hundred slabs, and they do not address optimal ordering of campaigns. In contrast to the HSM scheduling problem, the size of the

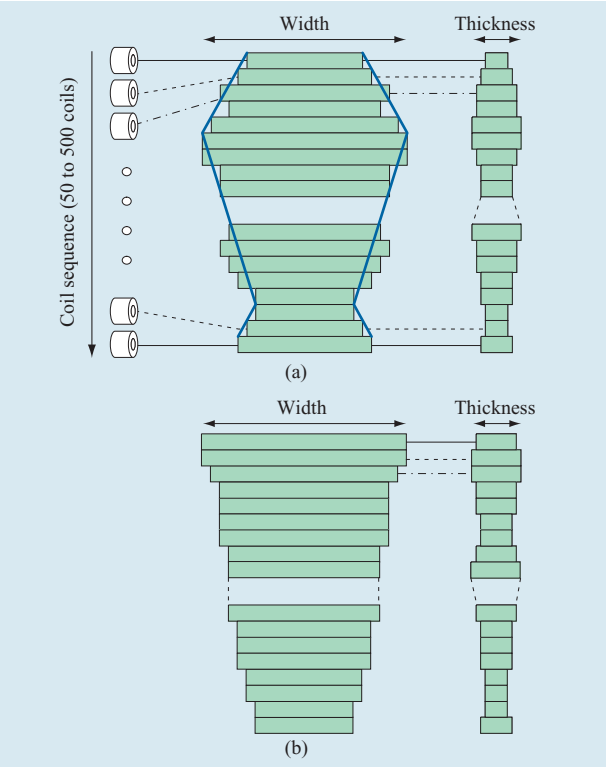

Figure 3

Width and thickness transitions of a coil sequence: (a) Typical profile; (b) wide-to-narrow profile.

FLS problem is about 5,000 coils on each process, and FLS addresses minimization of the setup cost and time between campaigns. In addition, FLS involves constraints on the process timings of coils in the four processes.

Numao and Morishita [4] address short-term scheduling problems in primary steelmaking and describe a system called *Scheiker*, which is a cooperative scheduling tool that interacts with a human expert to create schedules for material flow from the basic oxygen furnace to the continuous caster. The human scheduler first creates a global schedule using the Scheiker graphical user interface. Scheiker then utilizes the rules in its expert system kernel to identify local constraint failures and make repairs to correct the failures. If the failures are not fully repaired, the human scheduler modifies the schedule to rectify the identified failures. Scheiker then again identifies constraint failures and resolves them. This cooperative scheduling continues until all constraints are met and the objectives are achieved. In the FLS system, all of the scheduling is done exclusively by the engine for a one-month horizon. The human experts may make small changes for short-term horizons during actual execution.

Lee et al. [5] address integrated scheduling at continuous casters and HSMs. They address two techniques for caster scheduling and a third approach for integrated caster and HSM scheduling. The first of these generates caster schedules by clustering similar orders and then finding a sequence of these clusters using a heuristic search algorithm based on the beam and branch-and-bound search. These sequences are further improved by a genetic algorithm. The second approach is based on a solution framework, called asynchronous teams or A-teams [6, 7], which can exploit several solution techniques for finding multiple Pareto-optimal solutions. The third approach is based on clustering and a heuristic search. An analogous approach using A-teams was also described by Gao et al. [8]. In contrast, the FLS problem involves twice as many processes as their problem does, and has a branch of the process flow which their problem does not address.

Wein and Chevalier [9] discuss dynamic decision policies for assigning due dates, releasing jobs from a backlog, and sequencing jobs. Their problem assumes that the job shop is modeled as a multiclass queuing network, and the objective is to minimize both the work-in-process (WIP) inventory and the due date lead time (DDLT, the due date minus arrival date) of jobs. They proposed a two-step approach. The first step releases and sequences jobs, ignoring DDLT and focusing instead on efficient system performance. DDLT is then taken into account in the second step to set the due dates. In FLS, coils (jobs) are grouped into campaigns whose lengths vary from about a few hours to a week. Each coil in a campaign is associated with a release date from the preceding process and a due date for the succeeding process, and its DDLT is sometimes shorter than the campaign length. Therefore, once coils are assigned to campaigns, the time windows of campaigns are relatively short, and thus the sequence of campaigns is so restricted that the simple queuing network model does not apply. Therefore, the assignment of coils into campaigns and campaign sequencing in FLS should be solved at the same time.

Leachman et al. [10] address a large-scale and multiple-process scheduling problem in semiconductor manufacturing. Their scheduling algorithm for each process refers only to the WIP at hand, and the system is controlled by determining target cycle times and WIP levels for individual processes and by periodically releasing new jobs into the fabrication line. In their problem, each device is typically scheduled to be set up once per shift, which means that the campaign lengths are only one shift long. In contrast, in FLS, campaigns can be longer than the size of the WIP that exists when campaigns begin. Therefore, we use off-line scheduling algorithms that refer to all of the coils that will be released during a one-month horizon, not only to the WIP at hand.

In this paper, we describe a four-step approach for the FLS problem: 1) Input coils are clustered by two clustering algorithms to reduce the complexity and size of the problem. 2) Campaigns are created for each process from downstream to upstream processes, while propagating the process timings of the clusters upward. 3) Timing inconsistencies along the process flows are then repaired by scheduling downward. 4) Finally, coils are sequenced within each campaign. Campaign allocation, Steps 2 and 3, addresses resolving of the due date conflicts on the CM. Note that the process flow branches at the CM to CGL-bound and EGL-bound (Figure 1), and that the CM has to produce CGL- and EGL-bound coils in proper proportion, taking into account the due dates at the lower stream processes. Our algorithm propagates time windows upward to provide the CM with the timing requirements on the lower stream processes so that the CM can be scheduled accordingly.

The remainder of the paper is organized as follows: The FLS problem is presented in the next section. The solution framework for the FLS system and numerical experiments are described in Section 3. Details of each of the algorithmic components are presented in Sections 4 through 7. Finally, our conclusions are summarized in Section 8.

2. The FLS problem

The input data to the problem is the number of WIP coils on each of the four *continuous processes* and the number of coils that will be released from HSM to CM during the scheduling horizon. The problem is to create production campaigns to which coils are assigned for each line of the processes, and to sequence coils within each campaign so that productivity and product quality are maximized and tardiness is minimized.

Productivity means gross production per month divided by cost, where the term *gross* indicates the exclusion of parts of sheets which cannot be delivered because of trimming or scars. Maximization of gross production per month is addressed in the FLS problem as minimization of the gaps between campaigns, and maximization of productivity is addressed directly as campaign templates and indirectly as sequencing penalties. The gaps between campaigns include setup times and downtimes of processes due to both planned maintenance and lack of WIP inventory. When the WIP inventory in front of a process for each of the campaign types is less than the minimum campaign size, the process has to stop until enough WIP is supplied from upstream processes. Such a situation lowers productivity; therefore, the upstream processes should take into account the amount of WIP at lower stream processes. Setups of campaigns that require extra costs for changing of galvanizing liquid are undesirable, and campaign templates are designed by the human experts in

the steel mill to avoid such expensive setups. For example, a setup that requires changing only one galvanizing pot is preferred to a setup that requires changing many. The campaign templates are sequences of campaign types and downtimes that require only small setups. (Downtime is treated as a special campaign type.) Sequencing penalties include width differences between consecutive coils to represent minimization of trim loss. Note that the continuous process is a process in which all of the input coils within a campaign are welded together to make a long strip, processed at one time, and cut into coils after processing. When the widths of consecutive coils differ considerably, two corners of the wider coil are cut off to create a smooth profile (**Figure 4**). Therefore, in order to increase productivity, it is preferable to make width profiles of campaigns as smooth as possible. (In addition, when the difference in widths or thickness is very large, the welded juncture is weak, limiting the allowable differences of width and thickness.) There are also constraints on minimum campaign sizes, which are set to decrease the number of changes of rollers or galvanizing liquid.

Product quality is improved when the steel sheets are properly galvanized and are not scarred. Maximization of product quality is the production of as much high-grade product as possible without any problems. Note that each order is associated with a grade, and high-grade products (for example, for the outer panels of cars) are more expensive than low-grade products. Once a steel sheet for a high-grade product develops problems, it must be reassigned to a lower-grade order, whose profit margin is lower than those of high-grade orders. Also, in continuous processes, the rollers are worn by processing coils; processing narrower and wider coils in that order causes a transferring of scars, called *edge marks*, to the wider coils at the positions of both sides of the narrower coils (Figure 4). Therefore, in order to increase product quality, the width profiles of campaigns for high-grade products should be wide-to-narrow [Figure 3(b)]. Also, campaigns for high-grade products should not be placed immediately after downtimes, when the process machines are not yet stable. This preference is reflected in the campaign templates. There are constraints on maximum campaign sizes, which are set to avoid scars due to worn rollers and to keep product quality high.

Minimization of tardiness means that as many products as possible should be shipped before the due date.

Input and output

The input data of the FLS problem is the following:

- A set of processes $P = \{CM, CAL, EGL, CGL\}$ indexed by p.
- A set of lines L_p on process p indexed by l.

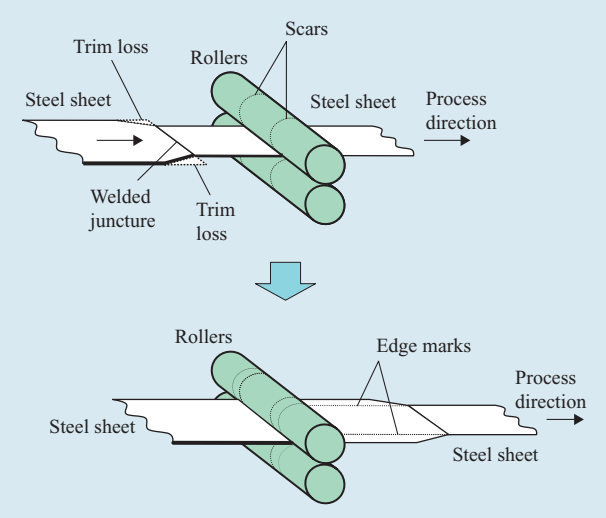

Figure 4

Edge marks and trim loss.

- A set of coils C_p on process p indexed by i.
- The previous campaign on each line of the processes.
- Any planned downtimes.

The previous campaign is a pre-allocated campaign which should be followed by newly created campaigns; planned downtimes are time periods in which campaigns cannot be created.

Associated with each coil i are

- A campaign type c_i.
- A set of assignable lines $L_i \subseteq L_p$.
- A processing time l_i.
- A priority: high or low.
- A release date r_i if coil i has no preceding process.
- A due date d_i if coil i has no succeeding process.
- A corresponding coil u_i in the preceding process.
- A lead time $l(u_i, i)$ to be allocated after the preceding process.
- Any process-dependent sequencing properties.

The preceding process is the process immediately upstream in the finishing lines from which coil i comes; the succeeding process is the process immediately downstream to which coil i goes. Each coil is assignable to only specific lines L_i. The process-dependent sequencing properties include width, thickness, annealing temperatures, and surface treatment types.

Associated with each campaign type c are the minimum length min_c and the maximum length max_c. For each line, the following constraints are given: campaign template,

chance campaigns, chance positions, and sequencing constraints. The campaign template is defined as a graph with campaign types as nodes and campaign transitions as directed edges. The chances for campaign type c are time windows at specific positions in the timeline: $[\mathrm{EST}_c^t, \mathrm{LFT}_c^t]$ for the tth chance.[2] Sequencing constraints include the allowable differences of width and thickness, width profile, setup costs between coils, and transition speed of the annealing furnace temperature. Most of these constraints can be represented as a distance function between each pair of coils.

The output data for the FLS solution is

- A set of campaigns K_l on each line l.
- A set of coils C_k assigned to each campaign k.
- A start time τ_k for each campaign k.
- A start time τ_i for each coil i.

With respect to the output data, the ordering of campaigns on line l is denoted by Π_l and the ordering of coils for campaign k is denoted by π_k.

Constraints and preferences

The output schedule should satisfy the following constraints and preferences (the latter are marked with an asterisk):

$\mathrm{EST}_i \le \tau_i$	[release date]
$\tau_{u_i} + l_{u_i} + l(u_i, i) + \tau_i$	[vertical relationship]
$\tau_{\pi_k(j)} + l_{\pi_k(j)} = \tau_{\Pi_k(j+1)}$	[continuous sequence]
$\tau_{k_1} + l_{k_1} + l(k_1, k_2) \le \tau_{k_2}$, $k_1 = \Pi_l(j), k_2 = \Pi_l(j+1)$	[setup time between campaigns]
coil i must be assigned to a campaign of type c_i	[campaign type]
coil i must be scheduled on line $\in L_i$	[assignable line]
$\mathrm{EST}_{c(k)}^t \le \tau_k \le \mathrm{LFT}_{c(k)}^t - l_k$ for any chance t	[chance campaign]
$\tau_i + l_i \le \mathrm{LFT}_i$	[due date]*
$min_{c(k)} \le l_k \le max_{c(k)}$ for all campaigns k	[min–max campaign sizes]*
campaign orderings follow campaign templates	[campaign templates]*
coil orderings follow sequencing constraints	[sequencing constraints]*

for each coil i and each line l of processes, where $l(u_i, i)$ denotes the lead time between coil i and its preceding coil u_i, $l(k_1, k_2)$ denotes the setup time between two campaigns k_1 and k_2, and $c(k)$ and l_k respectively denote a campaign type and the length of campaign k. Note that preferences may be violated when there is no feasible schedule. The due date constraint is marked as a preference because there may be no feasible schedule that satisfies both the due dates and the other constraints. The campaign templates are shown as a preference for the same reason.

Objectives

The objectives to be minimized for the FLS problem are shown in **Table 1**, in which the elements are listed basically in the order of importance assigned by human experts. The first three objectives (marked with +) are counted as weighted sums. Tardiness for coils with different priorities is treated with different weights. The other objectives are considered in the order shown. When an input coil (or cluster) has an EST at a much later position, the makespan[3] is determined by that coil, disregarding the other parts of the schedule. From a practical point of view, schedules with fewer gaps at early parts of the horizon are preferred, and the makespan is not very important. The minimization of gaps between campaigns reflects this preference (**Figure 5**). The sequencing constraint violation is listed last because detailed coil sequences are required for a horizon of only a few days. Except for the first few days in the beginning of a schedule, the primary objective in terms of sequencing is to guarantee sequenceability, which is not explicitly shown in Table 1.

System targets

The targets of the FLS system are as follows: 1) Create a one-month schedule within one hour of running time; 2) minimize tardiness; 3) maximize productivity; and 4) maximize product quality. The first target makes the size of the problem very large. The number of coils to be processed by all of the continuous processes in one month is 20 to 25 thousand. The issue arising here is how to reduce the problem complexity and size by clustering the coils or by decomposing the problem without significantly affecting the solution quality. The second target involves a tradeoff between tardiness and the constraints in campaign allocation. Tardiness can be minimized by creating small campaigns or by ignoring the campaign templates. However, setup costs will become large in such a schedule. The third target involves a line balancing problem. As shown in Figure 1, the process flow in the steel mill branches after CM, with one branch going to EGL and the other to CGL. Unless CM produces EGL-bound and CGL-bound coils in proper proportions, either

[2] The terms *EST* and *LFT* respectively indicate *earliest start time* and *latest finish time*.

[3] The term *makespan* indicates the duration from the beginning of the scheduling horizon to the completion of the last coil on each line.

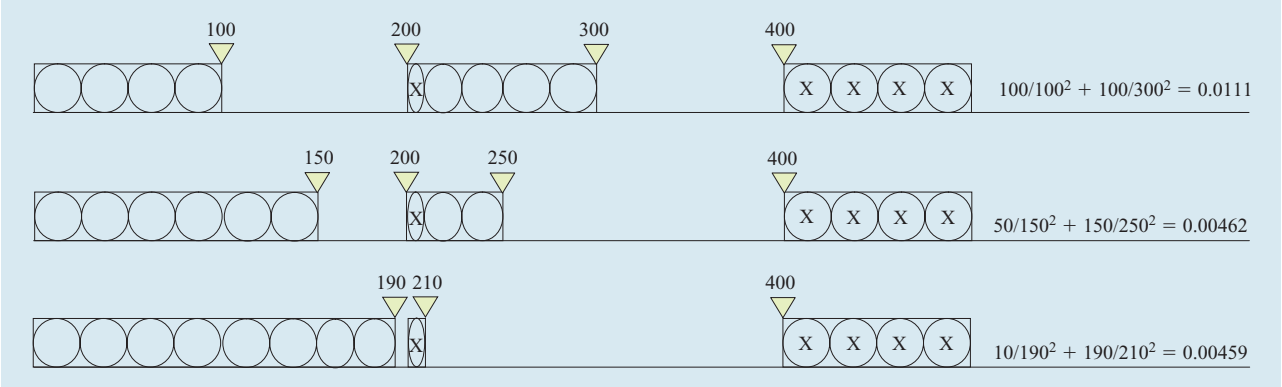

Figure 5

Example of gap penalty. The bottom schedule is the best and has the smallest gap penalty. Coils or clusters marked with X have earliest start times (ESTs) at their positions, and they cannot be moved earlier.

Table 1 Objectives to be minimized for the FLS problem.

$\Sigma_{p\in P}\Sigma_{i\in C_p} \max\{\tau_i + l_i - \mathrm{LFT}_i, 0\}$	[tardiness]+
$\Sigma_{p\in P}\Sigma_{l\in L_p}\Sigma_{k\in Kl} \max, \{min_c(k) - l_k, 0\}$	[minimum campaign size violation]+
$\Sigma_{p\in P}\Sigma_{l\in L_p}\Sigma_j[\tau_{k_2} - (\tau_{k_1} + l_{k1})]/\tau_{k1}^2,\ k_1 = \Pi_l(j),\ k_2 = \Pi_l(j+1)$	[minimize gaps between campaigns]+
$\Sigma_{p\in P}\Sigma_{l\in L_p}$ campaign template violation	[campaign template violation]
$\Sigma_{p\in P}\Sigma_{l\in L_p}\Sigma_{k\in Kl} \max\{l_k - max_{c(k)}, 0\}$	[maximum campaign size violation]
$\Sigma_{p\in P}\Sigma_{l\in L_p}$ sequencing penalty	[sequencing constraint violation]

EGL or CGL may be starved for coils and stop. Such stops (downtimes) should be avoided because they lower productivity. The consideration of the different product flows to avoid downtimes is called *line balancing*. The fourth target relates to the sequencing constraints and also to the maximum campaign size and the campaign templates.

3. Solution framework

In order to reduce the problem complexity and size, the FLS system incorporates two types of clustering: outlier and performance. Dimensions, time windows, and other parameters usually constrain the neighborhood of coils that can be scheduled together, and many of these neighborhoods are large for most typical coils. Such coils are likely to be sequenceable even if they are randomly assigned to campaigns. In contrast, coils with a small neighborhood or no neighbors, the *outlier coils*, should be put into campaigns along with near-neighbor coils which connect them to normal ones. By creating clusters of outlier and connecting coils, it is possible to separate the sequenceability problem from campaign allocation, and the problem complexity is reduced. This type of clustering is called *outlier clustering*. Normal coils and the outlier clusters are further clustered by *performance clustering* in order to reduce the problem size.

The FLS problem has two interrelated problems: horizontal and vertical. The horizontal problem involves campaign allocation and sequencing, and the vertical problem involves line balancing and vertical relationships. Handling them at the same time, however, is not practical because of the complexity. The FLS system thus adopts a horizontal engine for campaign allocation which runs independently for each process. The vertical problem is addressed by *time window propagation* and *upward* and *downward scheduling*. A time window $[\mathrm{EST}_i, \mathrm{LFT}_i]$ is assumed for each cluster i, which means that cluster i should be scheduled at the earliest start time (EST) or later, but (preferably) no later than the latest finish time (LFT). The time windows are set based on the release and due dates and the lead times, as shown in **Figure 6(a)**. The time window propagation defines a narrowed and non-overlapping time window for each cluster [**Figure 6(b)**], so that the horizontal engine for each process can work independently without worrying about the vertical relationships. For example, after creating the

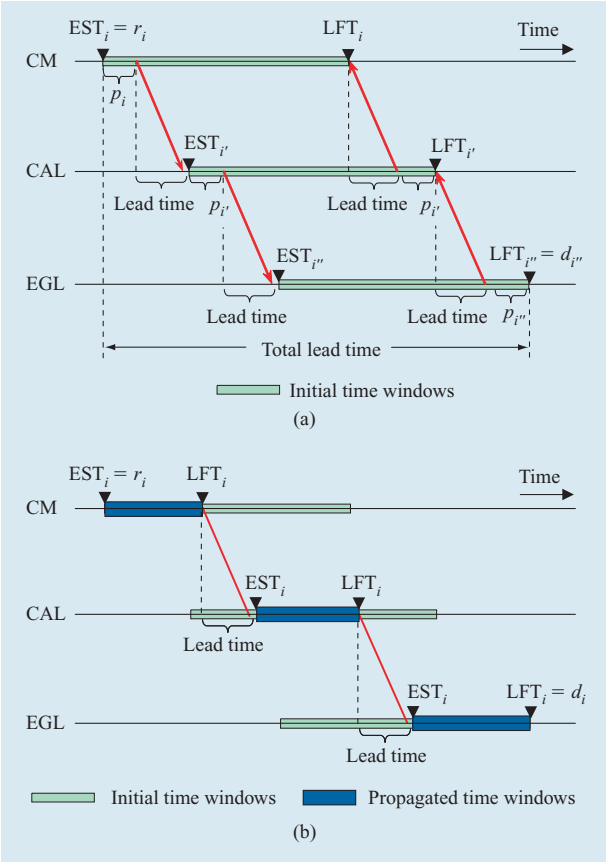

Figure 6

(a) Initial time windows for a cluster on three processes. The earliest start times (ESTs) are calculated from upstream to downstream processes, and the latest finish times (LFTs) are calculated in the opposite direction. (b) Propagated time windows for a cluster on three processes. The propagated time windows are calculated inside their initial time windows so that they do not overlap one another. The initial time windows are those depicted in Part (a).

line schedule for CM, the process timings of the clusters on CM are propagated downward, as illustrated in **Figure 7**.

The line balancing problem and tardiness minimization require resolving conflicts of due dates on CM. These are addressed in two steps: upward and downward scheduling. In upward scheduling, campaign allocation is performed from downstream to upstream processes, one by one, using the ESTs of initial time windows for the input clusters. As processes are scheduled, the process timings of clusters on lower stream processes are propagated upward to set the LFTs for the preceding clusters. When campaigns are allocated for CM, the timing requirements on all of the lower stream processes are represented as propagated time windows, and thus the horizontal engine can take line balancing into account by referring to

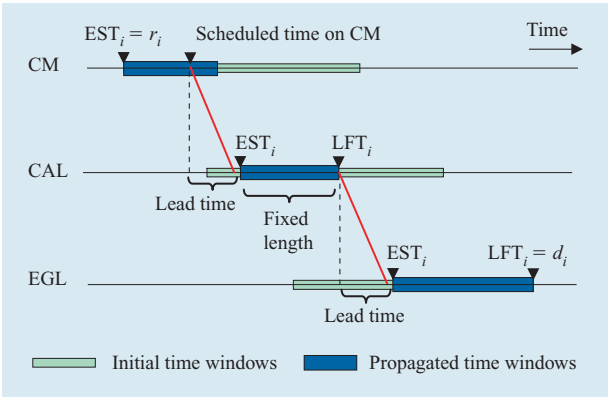

Figure 7

Propagated time windows for a cluster after scheduling CM. The scheduled time on CM of cluster i, τ_i, is propagated downward to recalculate the time windows on CAL and EGL.

the propagated time windows. Note that, when upward scheduling is completed, there may be vertical relationship violations because the campaign allocation uses the initial time windows and they do not include margins for preceding processes. The purpose of the upward scheduling is not to obtain feasible solutions, but to obtain LFTs on the CM that reflect the timing requirements on the lower stream processes. A feasible solution is then created by scheduling downward for the CGL, CAL, and EGL.

Sequencing for each campaign is performed after creating the campaigns. Sequencing problems are so process-dependent that the FLS system adopts different sequencing engines for each of the processes. The solution framework of the FLS system is summarized as follows: 1) upward scheduling for downstream-to-upstream processes, 2) downward scheduling for upstream-to-downstream processes, and 3) sequencing for each campaign, where the upward and downward scheduling are performed for each process by the horizontal engine. This horizontal engine consists of the following steps:

1. Outlier clustering.
2. Performance clustering.
3. Campaign allocation.
4. Time window propagation.

Theoretically, the whole problem can be represented as a single optimization problem, but handling such a huge problem is not practical with respect to both development and maintenance. We thought it would be preferable to decompose the problem into smaller components for which detailed heuristics can be captured easily. Therefore, we designed the campaign allocation step to run independently for each of the processes. The listed algorithmic components are described in the following

sections except for the time window propagation, which has already been discussed. The inputs for the steps of campaign allocation and time window propagation are clusters, not coils. In their descriptions, however, the notations for coils defined in the subsection on input and output are used for clusters.

Numerical experiments

Numerical experiments were conducted using a problem instance made for evaluation. The problem setting is as follows: There are two lines for each process: CM, CGL, CAL, and EGL. For each line of CM, 120 clusters were released at random between day 1 and day 15. The length of each cluster on CM is three hours. WIP clusters amounting to ten days are supplied for each CM line. The clusters on CM lines were associated with succeeding process lines equally among each line of CGL and CAL. The clusters bound for CAL were also associated with succeeding process lines equally among each of the EGL lines. The length of each cluster on CGL, CAL, and EGL is six hours. WIP clusters amounting to five days were supplied for each of the CGL, CAL, and EGL lines. The lead time between processes was set to one day. The total lead time for EGL clusters, i.e., LFT on EGL minus EST on CM (Figure 6), was set to 12 days, and the total lead time for CGL clusters, i.e., LFT on CGL minus EST on CM, was set to eight days.

The clusters on CM were randomly associated with one of the campaign types A, B, C, D, and E. No campaign template was specified for CM. The setup time between CM campaigns was set to zero. The clusters on CGL were randomly associated with one of the campaign types G, H, I, and chance. The H type is for high-grade products, which should not be placed immediately after G or chance. The setup time between G, H, I, and chance was set to eight hours. Clusters amounting to three days were associated with the chance, and their time windows were set between day 11 and day 15. The clusters on CAL were associated with one of the campaign types P and Q. No setup time was specified for switching between them. The clusters on EGL were randomly associated with one of the campaign types V, W, X, Y, Z, and chance. The campaign template was specified as V $\rightarrow$ W $\rightarrow$ X $\rightarrow$ Y $\rightarrow$ Z. The setup times from Z to V and between the chance and the others was set to eight hours. The campaign templates for CGL and EGL were represented as the distance matrices shown in **Tables 2** and **3**, where the distance is a virtual cost between campaign types. The distance matrix defines a directed graph, consisting of nodes as the campaign types and weighted arcs as transitions between them, in which minimum cost paths represent the campaign template. There were no planned downtimes in this experiment. The maximum campaign sizes for CM, CGL, and EGL were one day, 14 days, and two days, respectively.

Table 2 Distance matrix for the campaign types on CGL.

	G	*H*	*I*	*Chance*
G	0	10	5	10
H	10	0	1	10
I	10	1	0	10
Chance	10	10	10	0

Table 3 Distance matrix for the campaign types on EGL.

	V	*W*	*X*	*Y*	*Z*	*Chance*
V	0	1	10	10	10	10
W	10	0	1	10	10	10
X	10	10	0	1	10	10
Y	10	10	10	0	1	10
Z	1	10	10	10	0	10
Chance	10	10	10	10	10	0

The maximum campaign size was not specified for CAL. The minimum campaign size was set to 12 hours for all of the processes.

Figure 8 shows the results of the numerical experiments. In the upward scheduling shown in Figure 8(a), EGL and CGL were first scheduled with initial time windows set to their clusters. Let τ^{EGL} and τ^{CGL} denote scheduled times of the clusters on EGL and CGL, respectively. CAL was then scheduled with the clusters associated with LFTs propagated upward from EGL as $LFT^{CAL} = \tau^{EGL} - \mathrm{DAY}$. CM was scheduled last, with the clusters associated with the LFTs propagated upward from CGL and CAL as $LFT^{CM} = \min\{\tau^{EGL} - 2\mathrm{DAY}, \tau^{CAL} - \mathrm{DAY}\}$ and $LFT^{CM} = \tau^{CGL} - \mathrm{DAY}$. In the downward scheduling, the length of the time window on each process was set to four days (Figure 7). Figure 8(a) shows the result of the upward scheduling. The slanted vertical lines in the figure represent points at which two connected clusters involve a vertical relationship violation, which may happen because the LFTs are not always observed. Note that, in this step, clusters are tentatively sequenced within each campaign in order to check the vertical consistency. After the downward scheduling, the violations were repaired and a feasible solution was obtained [Figure 8(b)].

A naive solution that schedules only downward involves gaps (downtimes) on CAL between day 4 and day 5 [Figure 8(c)]. The proposed solution that schedules upward and downward does not involve gaps on CAL and EGL [Figure 8(b)]. This is because, in the proposed solution, the CM could consider the timing requirements of lower stream processes. When the extent of vertical

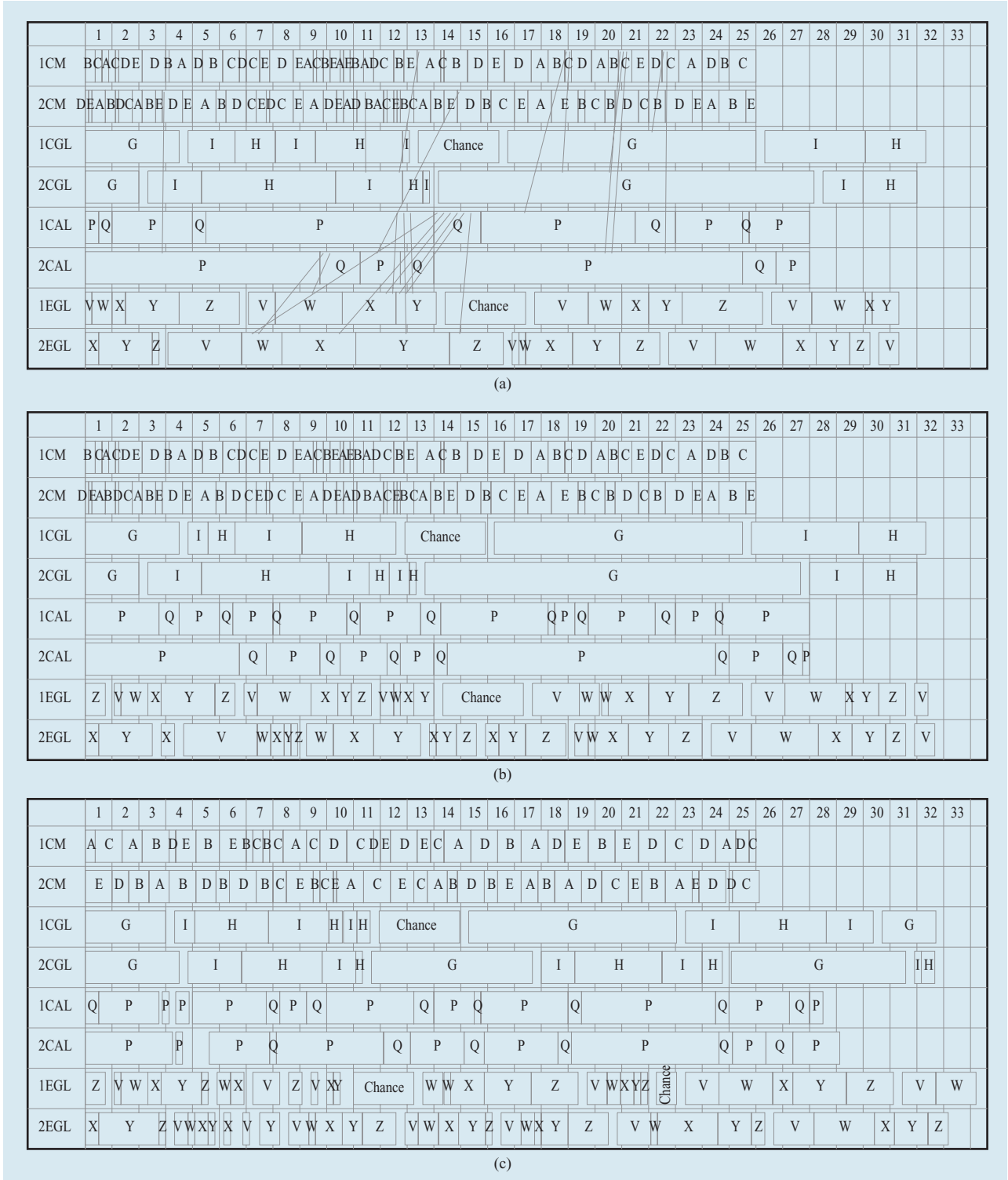

Figure 8

(a) Schedule obtained by upward scheduling. The slanted vertical lines represent vertical relationship violations. (b) Schedule obtained by the proposed solution method. Timing requirements of the lower stream processes are propagated upward [Part (a)], and then the vertical relationship violations are resolved by downward scheduling. (c) Schedule obtained by using downward scheduling only. Observe the undesirable gaps between campaigns on CAL and EGL.

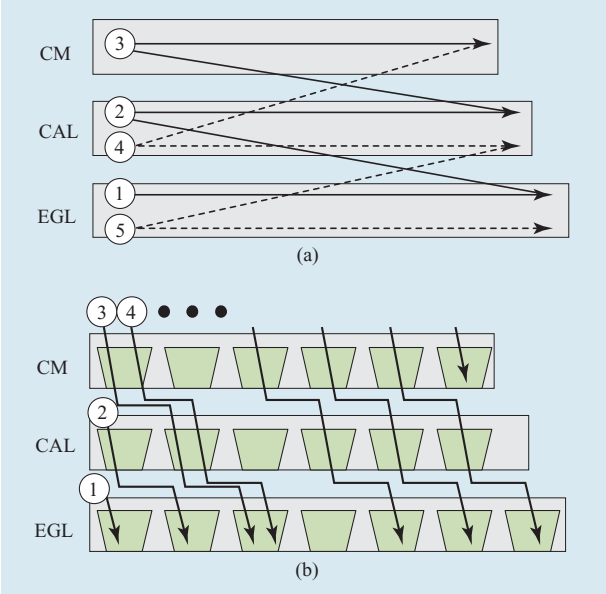

Figure 9

Comparison of (a) the proposed solution (upward and downward scheduling using a horizontal scheduler); (b) the capacity planning approach (put clusters into the daily buckets in order of due dates).

relationship violations is small, as in the schedule shown in Figure 8(a), the downward scheduling step may be replaced by local search on the violated clusters.

Comparison with capacity planning approach

In the proposed solution, schedules are created by running the campaign allocation engine for each process as depicted in **Figure 9(a)**. The campaign allocation engine can take into account the campaign templates, such as "H should be placed after I," while the vertical relationships are addressed by the time window propagation. An alternative approach we have considered is off-the-shelf capacity planning programs. Placing daily buckets for each process, such tools can iteratively assign associated coils to the buckets in the ascending order of their due dates [**Figure 9(b)**]. For example, a coil processed by CM, CAL, and EGL is assigned to the earliest assignable buckets at CM, CAL, and EGL at the same time. Campaigns can be created afterward according to the contents of the buckets. A drawback of this approach is that it cannot consider the constraints along timelines, that is, the campaign templates. We considered applying local search for the created campaigns, but such an approach seemed to make the solution complicated. Therefore, we gave up this approach and sought for other approaches which are suitable to address the constraints along timelines.

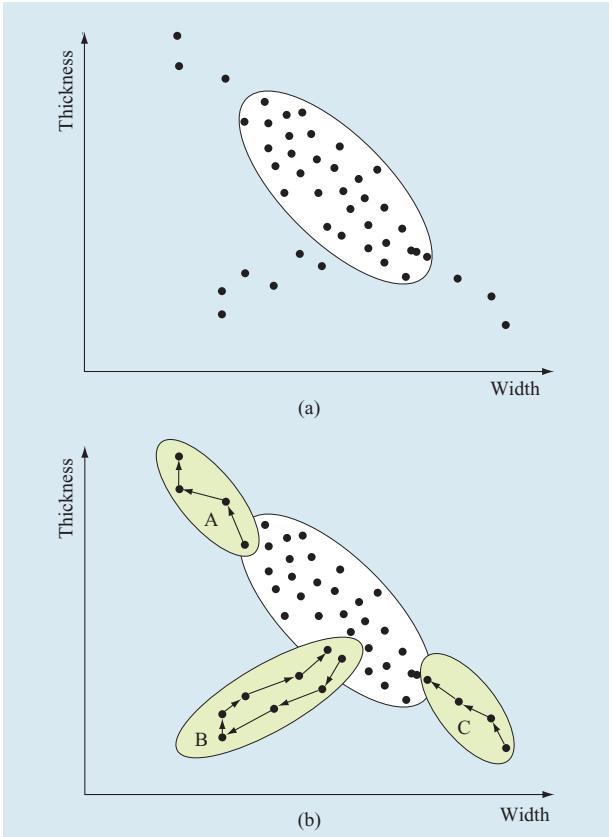

Figure 10

(a) Width/thickness plot of coils of a particular campaign type, showing middle range of coils (oval area) and outliners (points outside the oval). (b) Output of outlier clustering. The shaded ovals represent clusters connecting outlier coils to middle-range coils.

4. Outlier clustering

The FLS system performs the allocation of coils to campaigns independently of the sequencing of coils within a campaign. However, within a campaign, there are constraints that must be satisfied with respect to the way that coils can be sequenced. These constraints are usually based on the dimensions of the coils, such as width and thickness, but may take into account line-specific characteristics as well, such as the annealing temperature for CAL. The majority of coils have properties such that it is, in general, fairly easy to satisfy the sequencing constraints. However, there are often a small number of coils within certain campaigns which have atypical properties, such as very wide or very narrow coils. Such coils usually cause problems during sequencing, since the number of coils with which they can feasibly be sequenced is quite small. We call these *outlier coils*.

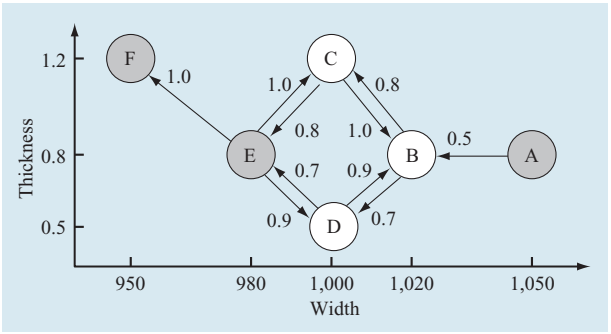

Figure 11

Example of a transition graph for outlier clustering. High-priority coils are represented by shaded nodes, low-priority coils by light nodes. Edge weights represent transition costs with respect to sequencing constraints.

In order to avoid possible sequencing problems with outlier coils after allocation, the horizontal engine runs an "outlier clustering" algorithm before allocation, which associates outlier coils with other coils that can be sequenced with them. All of the coils within an outlier cluster are then allocated to the same campaign during the allocation phase of the engine. **Figure 10(a)** illustrates a typical distribution of coils with respect to their width and thickness properties. The majority of the coils lie within what we call a "middle range," presented within the oval area on the graph. Coils within the middle range are compatible with many other coils with respect to how their width and thickness properties satisfy the sequencing constraints with other coils. Thus, whatever the campaign to which the middle-range coils are allocated, the expectation is that it should not be too difficult to feasibly sequence these coils. Coils outside the middle range are outlier coils, which will be hard to sequence. The primary goal of outlier clustering is to form clusters which connect outlier coils to coils in the middle range. The output of outlier clustering for the example illustrated in Figure 10(a) is presented in **Figure 10(b)**. The shaded ovals represent clusters of outlier coils created by the outlier clustering algorithm. The directed arcs in the solution graph represent a feasible direction of sequencing between two coils. (Note that the sequencing constraints are asymmetrical, since in general wide to narrow sequences of coils are preferred. In order to guarantee that clusters are sequenceable, therefore, we sequence coils within clusters.) Three outlier clusters were created here. Cluster A forms a path of coils away from the middle-range coils toward the narrowest coil. Such a cluster would probably be placed at the end of a sequence of coils in a campaign. Cluster B forms a path connecting coils from and back to the middle range. Cluster C connects the widest coils to the middle range.

Transition graph

The primary data structure for representing coils and constraints in outlier clustering is the transition graph. A transition graph is associated with a particular campaign type on a particular process. A transition graph for a campaign type is constructed from all of the coils with the same campaign type on a particular process.

In the transition graph, nodes represent coils. Nodes are labeled with the priority of the coil: high or low. A directed edge from node A to node B indicates that it is feasible to sequence the coil represented by node A directly before the coil represented by node B in some campaign with respect to the sequencing constraints. For all of the processes, these include width and thickness constraints, feasibility with respect to assignable lines ($L_A \cap L_B \neq \phi$), and overlaps of time windows ($[\mathrm{EST}_A, \mathrm{LFT}_A] \cap [\mathrm{EST}_B, \mathrm{LFT}_B] \neq \phi$). For the annealing line, the temperature constraint is also checked. Edges are included for only feasible transitions. Each edge is labeled with a transition cost, which lies in the range between 0.0 and 1.0. When this cost is 0.0, the two coils represented by the adjacent nodes have exactly the same properties, such as width and thickness. When the cost is 1.0, the difference between the width and thickness properties is the maximum allowable while still ensuring feasibility. For example, when the widths of coils B and C are 1,020 mm and 1,000 mm, respectively, and the maximum widening and narrowing allowances are 20 mm and 30 mm, respectively, the transition cost from B to C with respect to width is 20/30 = 0.67, and that from C to B is 20/20 = 1.0.

An example of a transition graph is illustrated in **Figure 11**. The maximum widening and narrowing allowances are the same as for the example above, and the maximum allowable thickness difference is 0.4 mm. The transition costs shown in the figure are the average values of the costs with respect to width and thickness.

Greedy clustering algorithm

A simple greedy algorithm is used to perform outlier clustering. The algorithm works by first creating a priority queue, where each entry in the queue contains the following information about each coil: 1) the coil identifier, 2) the scheduling priority of the coil, 3) whether the entry is for an incoming or outgoing connection for this coil, and 4) the number of possible incoming or outgoing connections to other coils. The priority queue sorts these entries in order of decreasing scheduling priority, followed by increasing number of incoming or

outgoing possible connections. For the transition graph example given in Figure 11, the outlier data for the coils represented in this graph, sorted in order of priority, would be the following:

1. { A, high, incoming, 0 },
2. { F, high, outgoing, 0 },
3. { F, high, incoming, 1 },
4. { A, high, outgoing, 1 },
5. { E, high, incoming, 2 },
6. { E, high, outgoing, 3 },
7. { C, low, incoming, 2 },
8. { D, low, incoming, 2 },
9. { B, low, outgoing, 2 },
10. { C, low, outgoing, 2 },
11. { D, low, outgoing, 2 },
12. { B, low, incoming, 3 }.

The greedy clustering algorithm works by successively removing entries from the priority queue; for each entry, if there are any available incoming or outgoing connections for the selected coil, it selects a connection and records it in the cluster. The logic for selecting a connection takes into account a number of factors, such as attempting to minimize the edge cost of the selected connection and maintaining a consistent direction for the direction of the cluster toward or away from the middle-range coils. The transition graph is then updated, which involves removing edges from the graph representing connections between coils which are no longer possible. For example, if coil D is assigned to be an incoming connection to coil B, it can no longer be an incoming connection to coil E. Other edges are removed to prevent cycles occurring in the transition graph, which would lead to infeasible orderings in clusters. Propagation of time windows is also performed within the graph as coils are added to outlier clusters. This may also result in edges being removed from the transition graph if they come to represent connections that become infeasible with respect to the time windows of the connecting coils. Note that after propagation, the data stored in the priority queue may no longer be current, since the number of possible connections for a coil may have changed. We need to recompute this data for any coils affected by propagation, which may also result in the priority queue order being changed.

The greedy algorithm could be extended to perform a more rigorous depth-first backtracking search, but in practice this was not found to be necessary for the problems for which we used the clustering algorithm. In the few cases in which clustering could not connect all outlier coils to the middle range, a number of relaxations could be attempted, such as relaxing some of the time windows of the coils or bringing in substitutable coils from other campaign types to help make connections.

5. Performance clustering

Performance clustering takes a set of coils on a process and groups near-neighbor coils into clusters that can be treated as indivisible units by other algorithms in the FLS system. The performance-clustering algorithm uses the transition graph to find the closest pair of coils and groups them one by one, while putting more priority on clustering such that the resulting transition graph of clusters has more edges between clusters so that the loss of scheduling flexibility is limited. The purposes of performance clustering are to reduce the problem size for subsequent algorithms, and to create clusters of at least the minimum campaign size.

The input data for the performance clustering is

1. A set of coils of a particular campaign type.
2. A set of clusters created during outlier clustering.
3. The maximum cluster duration.
4. The maximum allowable edge cost for connecting coils and/or clusters.

The output of performance clustering is a set of clusters. The coils in a cluster are explicitly sequenced, making it straightforward to determine the effective time window of the cluster. Clusters created by previous clustering steps are respected during performance clustering; that is, all of the coils in an input cluster retain their relative order in the output cluster.

The *multiple-fragment* clustering algorithm operates by creating a transition graph in which nodes represent clusters and directed edges between each pair of nodes indicate that one can feasibly merge the adjacent clusters into a single cluster. The algorithm steps are as follows:

1. Create a cluster for each input coil.
2. Create a node for each cluster (including input clusters).
3. Create edges between clusters by evaluating coil sequences, time windows, and maximum cluster size constraints. Save the costs associated with feasible edges in the priority queue.
4. Obtain the next edge (i, j) from the priority queue. If there is no next edge, terminate.
5. If both i and j are contained in the same cluster, discard this edge to avoid creating a cycle, and go to Step 4.
6. Merge cluster i into cluster j.
7. Remove all of the out-edges of i and all of the in-edges of j.
8. Determine the time window and the size of the newly combined cluster $\{ij\}$.

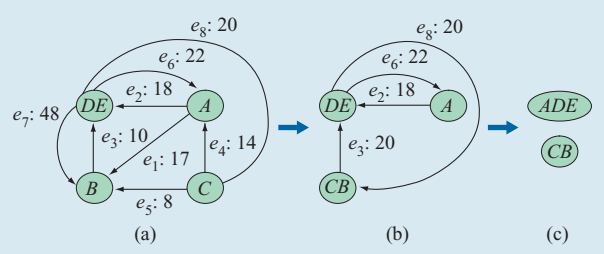

Figure 12

Multiple-fragment clustering algorithm.

9. Re-evaluate all of the in-edges and out-edges of $\{ij\}$, remove infeasible edges, and update the priority queue costs for feasible edges.
10. Go to Step 4.

Note that the multiple-fragment algorithm is a well-known heuristic for the *traveling salesperson problem* (TSP), which starts with each point as a fragment consisting of a single point and patches the closest pairs of fragments one by one without making any points of degree three or small loops (see for example [11]).

As an example, consider an input with three coils A, B, and C, and one two-coil cluster $\{DE\}$. After creating clusters $\{A\}$, $\{B\}$, and $\{C\}$, suppose that evaluating all pairs of edges between clusters results in the graph shown in **Figure 12(a)**. We choose the lowest-cost edge e_5, and so merge cluster C into cluster B. All of the in-edges of B and the out-edges of C are removed. Suppose that after re-evaluation we find that edge e_3 changes its cost to 20. The new graph is shown in **Figure 12(b)**. We again choose the lowest-cost edge e_2 and form cluster $\{ADE\}$. Suppose that on re-evaluation the remaining edges are not feasible; we then terminate with the graph shown in **Figure 12(c)**.

The cost of linking two clusters in performance clustering is dynamically re-evaluated in the multiple-fragment algorithm in order to take into account time window flexibility, while the edge cost used in the outlier clustering is static. The difference between the maximum EST and the minimum EST over all coils in a cluster gives the minimum amount of time the earliest available coil must wait to be processed. This wait time, the ESTGap, must be no greater than MaxESTGap for the process. Similarly, the difference between the maximum LST and the minimum LST over all jobs in a cluster, the LSTGap, must be no greater than MaxLSTGap for the process. For the purposes of edge scoring, we define the time window component to be the ratio of the ESTGap and the MaxESTGap. This quantity is added to the geometric component described in the outlier clustering section to obtain the edge score. A useful extension to the edge cost function is to consider sequencing flexibility loss as well. Recall that in the discussion above, when two clusters are merged, the in-edges of the target cluster and the out-edges of the source cluster are removed. Given two edges e_1 and e_2 having the same cost, we can decide between them on the basis of the extent to which committing each edge would reduce the overall sequenceability of the set of clusters. One measure of this is the sum of the out-degree of the source and the in-degree of the target. We call this measure the *disruptiveness* of the edge. The more accurate but significantly more expensive measure is to actually evaluate the feasible edges remaining after committing e_1 and compare this to the feasible edges remaining after committing e_2 instead.

6. Campaign allocation

Campaign allocation is a task to create campaigns and assign given clusters—outlier and performance—to the campaigns for each process, taking into account the constraints and preferences described in Section 2, except for those for sequencing. The detailed sequencing constraints are not considered in this task to reduce the complexity of the problem and to keep it independent of the processes. The campaign templates are not directly considered, but they are represented as distances between each pair of campaign types and reflected indirectly as minimization of the total distance between allocated campaigns. We assume that each cluster has the same properties as a coil, and that C_p denotes a set of input clusters for process p. The output of campaign allocation is a partition of C_p into lines, a partition of the clusters into campaigns for each line l, and the process timing τ_i of each cluster i.

The algorithm used for campaign allocation is based on the traveling salesperson problem with time windows (TSPTW), and implementation ideas are taken from previous studies in this area (see for example [12]). The variation of the TSPTW discussed here is the multiple TSPTW with color (campaign-type) constraints, where "multiple" means that plural lines are considered at the same time.

Solution representation for campaign allocation

Campaign allocation solutions are represented by sets of disjoint clusters and orderings of clusters for each of the lines $l = 1, \cdots, |L_p|$. Given an ordering of clusters π_l, the output schedule is created as follows: The process timing τ_i for the first cluster, $i = \pi_l(1)$, is calculated as a continuation from the previous campaign, and that for the other clusters, $i = \pi_l(j), j > 1$, are calculated as continuations from the previous clusters, $\pi_l(j-1)$, within the ordering. First, tentatively set τ_i to the finishing time

of the previous cluster. If the campaign types of the previous and current clusters are different, the setup time between them is added to τ_i. If $\tau_i < \mathrm{EST}_i$, τ_i is reset to EST_i. If i is associated with chance terms and τ_i is not contained in any of the chance terms, the closest chance term after τ_i is found. Let t be the EST of the closest chance term after τ_i. If $t - \tau_i \leq \theta$, τ_i remains as it is; otherwise τ_i is moved to t, where the threshold θ is a tuning parameter. The gaps created between clusters by EST or chance terms are filled with other clusters during the local search phase described later.

Construction of campaigns

For the initial solution for a process, the best solution is selected from the solutions created by the *nearest neighbor method with randomization*. In this method, campaigns with random sizes are placed at the earliest available positions, as illustrated in **Figure 13**. The type of a newly placed campaign is selected with some randomization based on the distance from campaigns which have already been allocated. Note that the distances between campaigns are defined to represent the campaign templates, and they include many ties. The randomization is intended to break ties randomly to create various solutions.

The size of the newly placed campaign is set to a random value, with consideration of the minimum and maximum sizes of the campaign. The size is also limited by the next chance timing, as in the campaign labeled 4 in Figure 13. If the earliest position is associated with a chance term, e.g., the position of campaign 6, the associated chance campaign is placed without this randomization to ensure that chances are handled properly. Clusters with the same campaign type as the newly placed campaign are selected and assigned to the campaign in order of decreasing priority, followed by increasing EST. To implement this procedure in linear time to the number of the input clusters, for each cluster i, the other clusters $C_p \backslash \{i\}$ are sorted in advance in decreasing order of priority, in increasing order of EST, and in decreasing order of the distance from i. The sorted list is called the *neighbor list*. When clusters are assigned to a new campaign, the first cluster is selected on the basis of the neighbor list of the last cluster of the previous campaign. The set of clusters to fill in the new campaign are selected in the order of the neighbor list of the first cluster. Finally, the best solution among the created solutions is selected, and the resulting orderings $\pi_1, \cdots, \pi_{|L_p|}$ are passed to the local search described in the next subsection.

Local search for campaign allocation

Local search applies two types of neighborhood operations: 2-opt and path-move. The 2-opt operation reverses a path of clusters within a campaign [**Figure 14(a)**],

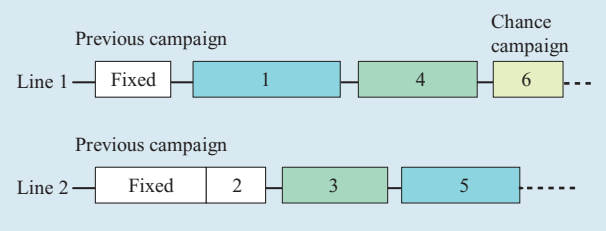

Figure 13

Construction order of campaigns for a process with two lines. The campaigns labeled 1 through 6 are created in that order. Campaign 6 is a chance campaign whose start timing is given as input, and the size of campaign 4 is bounded accordingly.

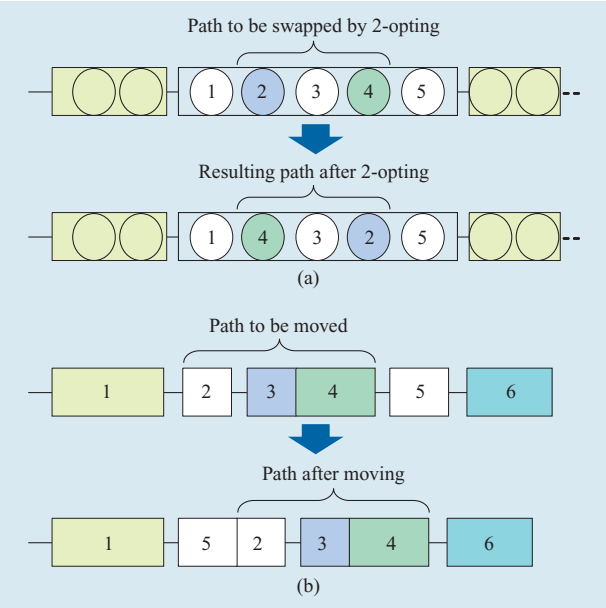

Figure 14

(a) 2-opt operation; cluster 2 is the base, and cluster 4, $\mathrm{EST}_4 \leq \mathrm{EST}_2$, is the partner. (b) Path-move operation for a path of campaigns within a line. Path-moving across lines or for a cluster is also possible.

and the path-move operation moves a cluster or a path of campaigns within a line or across lines [**Figure 14(b)**]. The latter shows a move operation for a path of campaigns within a line. The path-move operation for a cluster is used so that the due date violations of individual clusters are minimized, and small gaps between clusters are filled.

The local search in campaign allocation uses the following framework: An *active cluster list* is initialized with all of the input clusters. A base cluster is removed at random from the list, and the neighborhood operations

with respect to the base are examined. That is, candidate 2-opt and path-move operations, which cut edges that have the base as one of the ends are examined one by one. While examining the neighborhood operations, the first one that reduces the objective value is accepted, and the remaining candidate neighborhoods are discarded. After each application of the neighborhood operations (that is, when the operations have reduced the objective value), the clusters at both ends of the two cut edges in 2-opting or the four cut edges in path-moving are inserted into the active cluster list. The local search continues until the list becomes empty. The objective function is defined on the basis of the objectives described in Section 2, except for the sequencing constraint. The candidates for a 2-opt with a base are created as follows: The partner clusters for the base are first chosen from clusters within the same campaign, whose ESTs are earlier than the base and which are placed after the base. The candidate paths to be reversed by 2-opting are the paths from the base to each of the partners. Note that the rule for choosing partners is designed to minimize tardiness by 2-opting. The candidates for a path-move with a base are created as follows: The partner clusters for the base are first chosen as the base itself and the last clusters of the campaigns on the same line placed after the base. The candidate paths to be moved are the paths from the base to each of the partners, that is, either the base itself, the latter half of the campaign from the base, or consecutive campaigns. The destinations considered for the candidate paths are the transition points of the campaigns. As campaigns are moved by local search, the distances between campaigns are minimized, and the orderings of the campaigns are gradually refined so that the campaign templates are satisfied.

7. Sequencing

Finding the minimum cost ordering π_k of coils C_k assigned to each campaign k is called *sequencing*. The sequencing is done for every campaign in the FLS system, and it is also called from a schedule-editing program used by the human experts. The FLS system creates a layout of campaigns and orderings of coils for a one-month horizon, and the schedule for a few days from the beginning is passed to the human experts. The human experts finally fix the orderings of the coils considering the situations changed after running the FLS system, such as priority changes and delays or failures in coil making in the preceding processes. Therefore, the sequencing program should be fast enough that a sequence of a few hundreds of coils can be obtained within a minute. Note that the input for sequencing is not clusters, but coils. Thus, the scalability of the sequencing programs (sequencers) cannot be ensured by the performance clustering described in Section 5.

In this task, detailed sequencing constraints which differ greatly for different processes should be taken into account, and so different sequencers were implemented for each of the processes. Although the detailed constraints differ greatly for each process, they share the same type of constraints regarding allowable differences of width and thickness. For most of the campaign types on every process, the overall profile of widths should be wide to narrow in order to avoid edge marks, as described in Section 2 [see Figure 3(b)]. For campaign types or processes whose wide-to-narrow preference is strong, the sequencing problem resembles the simple sorting problem; for those whose wide-to-narrow preference is weak, it resembles the traveling salesperson problem with time windows (TSPTW). We selected the TSPTW as a base model for all of the sequencers, and implemented process-dependent constraints on top of the base program.

In the next two subsections, the base TSPTW program and the proximity search method used in the base program are described. The detailed description of the CGL sequencer, which is regarded as the most difficult process in the finishing lines in terms of sequencing, appears in another report [13].

TSPTW-based sequencing

The TSPTW is a well-known problem for which various heuristics have been proposed by many researchers [12]. In particular, the subject problem has a distinct structure similar to that of the geometric TSP, which is known to be easier than non-metric TSPs. Note that polynomial-time approximation schemes are known for the former problem, whereas no performance guarantee is possible for the latter class of problems. Note also that for the TSPTW, even finding a feasible solution is known to be NP-hard, but in practice it is as easy as the TSP when the time windows are wide. In the sequencing problem, the TSP "cities" to be visited are coils each having a specific width and thickness. In terms of both width and thickness, distance and maximum difference between each pair of coils are defined. By mapping the coils on a plane of width and thickness as shown in Figures 10(a) and 10(b), the sequencing can be seen as a problem to find the minimum-cost path that visits all of the coils on the plane. The time windows of the coils are relatively wide compared to the campaign length, so that the geometric characteristics are dominant in sequencing.

The algorithm used in the base program for sequencing has the same structure as that used for campaign allocation described in Section 6, consisting of a construction heuristic and local search. The solution representation is an ordering π_k. The process timing τ_i of each coil i is calculated by scanning π_k from the beginning to the end as $\tau_{\pi_k(j)} = \max\{\mathrm{EST}_{\pi_k(j)}, \tau_{\pi_k(j-1)} + l_{\pi_k(j-1)}\}$. As described in Section 2, there must be no gaps in process

timing between consecutive coils. The objective function is defined to penalize such gaps so that gaps are filled during the local search. The objective function is a linear combination of penalties for gaps, tardiness, edge marks, transition costs, and various process-dependent constraints.

The construction heuristic used for the sequencers is the *addition heuristic*. In this heuristic, the first and the last coils, $\pi_k(1)$ and $\pi_k(|C_k|)$, are determined first, and the current solution is created from those coils. The coil nearest to the current solution is then selected from among the unvisited coils and inserted into the position in the current solution which least increases the objective value. This continues until all of the coils have been added. This procedure can be efficiently implemented using a priority queue to hold the nearest unvisited coil from each of the coils in the current solution. The local search used for the sequencers applies the 2-opt and path-move operations in the same way as in campaign allocation. The structure of the local search is also the same as for campaign allocation; i.e., an active list of base coils is used. A significant difference from the campaign allocation is the use of the geometric characteristics, as described in the next subsection.

Using the geometric structure

In the sequencers, queries for the nearest-neighbor coils or for near-neighbor coils within some radius from a given coil are used many times. For this purpose, we use a two-dimensional *k*-d tree [14] where the dimensions represent width and thickness. Before the *k*-d tree is created, the dimensions of the input coils are normalized by the average allowable differences of width and thickness so that the rectangle of the allowable differences on the width/thickness plane becomes a square. This rectangle is called the transition rectangle (**Figure 15**).

When the allowable differences of width and thickness are a and b, respectively, the search radius is set to $(r/2)\sqrt{a^2 + b^2}$ for finding the near-neighbor coils. The tuning parameter r is a radius factor used to limit the search area. The coils found by the *k*-d tree inside the circle are examined one by one to check whether a transition from the search point (center of the circle) is permissible, taking into account all of the properties. **Figure 16** shows the performance of the CGL sequencer plotted against the radius factor. (A computer with a 1-GHz processor and a 512-MB memory was used.) The figure shows that the implementation using the *k*-d tree is efficient, and it is possible to meet the time requirements by setting the search radius appropriately.

O(1)-time feasibility check in sequencing

In the construction and local search heuristics of the sequencers, the objective function takes $O(n)$ time to evaluate the exact cost of a coil sequence, where $n = |C_k|$ is the number of coils in the sequence. The same amount of time is required even if a coil sequence is updated only locally because a local change might affect the whole sequence for some constraints such as tardiness, temperature transitions, wide-to-narrow constraint, and so on. For example, it must scan the sequence forward and backward to compute the target furnace temperature for the continuous annealing line. Though such a linear time algorithm might seem practical, the reality is that the objective function is evaluated within a loop of $O(n)$ or higher complexity, thus making the whole computational cost more than quadratic, which is impractical.

To make the computation faster, we adopted a two-stage evaluation: 1) In the first stage, the cost of the new sequence is "estimated" in $O(1)$ time. 2) If the estimated

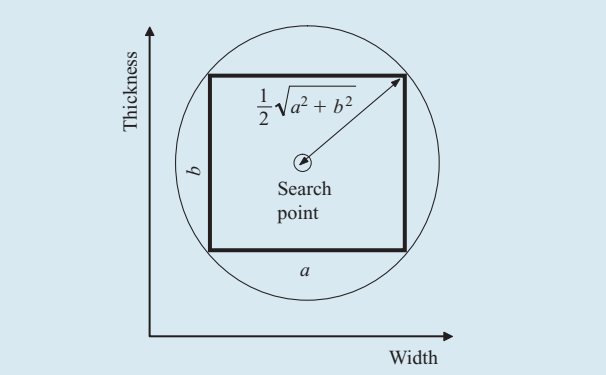

Figure 15

Transition rectangle.

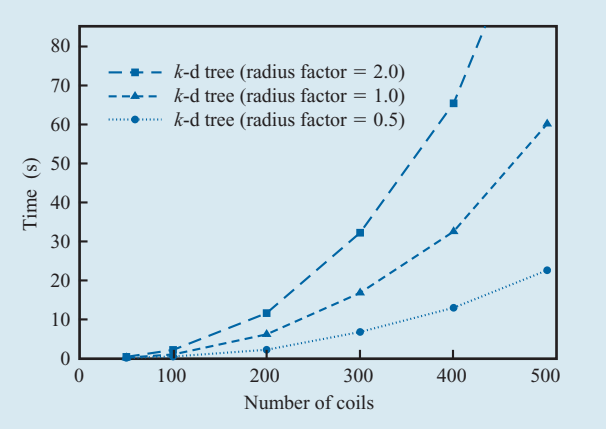

Figure 16

Performance of the sequencer.

cost is lower than the cost of the original sequence, the cost of the new sequence is then exactly evaluated using $O(n)$ time; otherwise, the new sequence is discarded without actually evaluating the objective function. Note that, for time windows, an $O(1)$-time feasibility check algorithm which does not reject feasible solutions is already known [15], whereas in our logic, which considers various constraints, the first stage may reject feasible solutions in rare cases. Most of the neighborhood solutions examined in local search will be worse than the current one, especially when the search is near a local optimal solution. Therefore, rejecting obviously worse solutions in the first stage can greatly speed up the sequencers.

To estimate the new sequence in $O(1)$ time, we propagate information forward and backward through the sequence and store additional information for each of the coils. Given the changed parts of the sequence, the cost of the sequence is estimated in the first stage by referring only to the additional information around the changed parts. As a result, we were able to make the sequencers 15 to 50 times faster without making the solution quality worse.

8. Conclusions

A new solution for the finishing line scheduling (FLS) problem in a major steel mill has been described. The production schedules for finishing lines involve various types of production campaigns because the number of kinds of steel sheet products produced in the steel mill is very large. Therefore, minimization of the setup cost and time between campaigns is one of the main requirements of the FLS problem. The problem also requires ensuring the timing consistencies of coils across four processes. Capacity planning tools are not applicable to this problem because of these complex constraints. To the best of our knowledge, there is no prior study in the literature that addresses the FLS problem as of the time at which this paper is being written.

Issues in designing the solution were the complexity and size of the problem. In order to reduce the problem complexity, we separated the problems of sequenceability and timing consistency from campaign allocation by using outlier clustering for sequenceability, and by using time window propagation and upward and downward scheduling for timing consistency. In order to reduce the problem size for campaign allocation, we introduced performance clustering. The scalability of the sequencers was ensured by the radius factor and the $O(1)$-time feasibility check. These approaches made it possible to scale our solution up to large amounts of input data, which, in the case of the subject steel mill, comes to 20 to 25 thousand coils, while satisfying the one-hour limit for running time.

By using the FLS system, the steel mill can provide the primary production scheduling system with accurate due dates of coils for a one-month horizon. The systems integration has just been completed, and the cost savings have not yet been determined. The expected merits yielded by the FLS system are the following: plan lead time reduction, inventory reduction, workforce reduction, capability for quick rescheduling, and accurate due date quotation, where the plan lead time means the time required from order input to production, the inventory means the number of work-in-process (WIP) coils, and the workforce is for sequencing and for the creation of *chance tables*. The steel mill created rough production campaigns of daily buckets for a one-month horizon, called a chance table, to use for due date quotation or for workforce planning. Since the FLS system can automatically create a production schedule more precise than the chance table, creation of the chance table itself is no longer necessary. The FLS system has allowed the steel mill to create schedules with a one-month horizon every day, and has increased the potential decision frequency from monthly to daily.

Acknowledgments

The authors would like to express sincere gratitude to Mr. Yuhichi Shibata at IBM Business Consulting Services for his valuable comments and suggestions on the manuscript.

References

1. H. Yasuda, H. Tokuyama, K. Tarui, Y. Tanimoto, and M. Nagano, "Two-Stage Algorithm for Production Scheduling of Hot Strip Mill," *Oper. Res.* **32,** 695–707 (1984).
2. H.-L. Fang and C.-H. Tsai, "A Genetic Algorithm Approach to Hot Strip Mill Rolling Scheduling Problems," *Proceedings of the International Conference on Tools with Artificial Intelligence*, IEEE, Piscataway, NJ, 1998, pp. 264–271.
3. L. Lopez, M. W. Carter, and M. Gendreau, "The Hot Strip Mill Production Scheduling Problem: A Tabu Search Approach," *Eur. J. Oper. Res.* **106,** 317–335 (1998).
4. M. Numao and S. Morishita, "Co-operative Scheduling and Its Application to Steelmaking Industries," *IEEE Trans. Indust. Electron.* **38,** 150–155 (1991).
5. H.-S. Lee, S. S. Murthy, S. W. Haider, and D. V. Morse, "Primary Production Scheduling at Steelmaking Industries," *IBM J. Res. & Dev.* **40,** 231–252 (1996).
6. S. N. Talukdar, L. Baerentzen, A. Gove, and P. S. de Souza, "Asynchronous Teams: Cooperation Schemes for Autonomous Agents," *J. Heuristics* **4,** 295–321 (1998).
7. S. N. Talukdar, P. S. de Souza, and S. Murthy, "Organizations for Computer-Based Agents," *Int. J. Eng. Intell. Syst.* **1,** 75–87 (1993).
8. H. Gao, J. Zeng, and G. Sun, "Multi-Agent Approach for Planning and Scheduling of Integrated Steel Processes," *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 2002, pp. 427–432.
9. L. M. Wein and P. B. Chevalier, "A Broader View of the Job-Shop Scheduling Problem," *Manage. Sci.* **38,** 1018–1033 (1992).
10. R. C. Leachman, J. Kang, and V. Lin, "Slim: Short Cycle Time and Low Inventory in Manufacturing at Samsung

Electronics," *Interfaces* **32,** 61–77 (2002).
11. J. L. Bentley, "Fast Algorithms for Geometric Traveling Salesman Problems," *ORSA J. Computing* **4,** 387–411 (1992).
12. D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study," *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra, Eds., John Wiley and Sons, Inc., New York, 1997, pp. 215–310.
13. H. Okano, T. Morioka, and K. Yoda, "A Heuristic Solution for the Continuous Galvanizing Line Scheduling Problem in a Steel Mill," *Research Report RT-0478*, 2002; IBM Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan.
14. J. L. Bentley, "K-d Trees for Semidynamic Point Sets," *Proceedings of the Sixth Annual ACM Symposium on Computational Geometry*, 1990, pp. 187–197.
15. G. A. P. Kindervater and M. W. Savelsbergh, "Vehicle Routing: Handling Edge Exchanges," *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra, Eds., John Wiley and Sons, Inc., New York, 1997, pp. 337–360.

Received October 15, 2003; accepted for publication May 28, 2004; Internet publication September 17, 2004

Hiroyuki Okano *IBM Research Division, IBM Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan (okanoh@jp.ibm.com).* Mr. Okano joined the IBM Tokyo Research Laboratory in 1990 after receiving B.S. (1988) and M.S. (1990) degrees in information science from the Tokyo University of Agriculture and Technology. After five years of research on user interfaces and systems software, Mr. Okano joined the operations research group and started work on combinatorial optimization. Since then, he has been working on developing solutions to several optimization problems in logistics and the manufacturing industry. His research interests include scheduling and optimization, local search techniques, and stochastic sensitivity analysis.

Andrew J. Davenport *IBM Research Division, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (davenport@us.ibm.com).* Dr. Davenport received his M.Sc. and Ph.D. degrees in computer science from the University of Essex, United Kingdom. He spent four years working as a Research Scientist in the Enterprise Integration Laboratory at the University of Toronto, Canada, where he focused on developing new constraint programming technologies to solve complex planning and scheduling problems in industry. In 1999 he joined the IBM Research Division, where he is a Research Staff Member in the Department of Mathematical Sciences. He is currently working on projects applying constraint programming, mathematical optimization, and artificial intelligence technologies to a range of customer problems in electronic commerce, production planning, and scheduling.

Mark Trumbo *Dragonfly Consulting, 3-320 Chapel Street, Ottawa, Ontario K1N 7Z3 Canada (marktrumbo@rogers.com).* Mr. Trumbo joined IBM in 1988 after receiving a B.S.E.E. degree from the University of Washington; he joined the Manufacturing Research Department at the IBM Thomas J. Watson Research Center in 1990. After receiving an M.A.Sc. degree from Simon Fraser University in 1994, Mr. Trumbo returned to the IBM Research Division to concentrate on planning and scheduling for the steel industry; over the course of several years, he successfully implemented inventory application, slab design, and cast design systems for a major Japanese steel producer. Now an independent consultant, Mr. Trumbo continues to collaborate closely with IBM Research and IBM Global Services to provide solutions for customers in the metals industry.

Chandra Reddy *IBM Research Division, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (creddy@us.ibm.com).* Dr. Reddy joined IBM in 1998 after receiving a Ph.D. degree in computer science from Oregon State University. Since then, he has been working on developing solutions to several optimization and scheduling problems arising in the steel industry. Several of these solutions have been successfully deployed and are in production at two of the world's leading steel makers. His research prior to joining IBM focused on both theoretical and practical aspects of machine learning of compact hierarchical control knowledge for planning and problem solving. Dr. Reddy's research interests include scheduling and optimization, machine learning for planning and classification, data mining, and artificial intelligence.

Kunikazu Yoda *IBM Research Division, IBM Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan (yoda@jp.ibm.com).* Mr. Yoda received a B.S. degree in applied mathematics and physics and an M.S. degree in applied systems science from Kyoto University in 1994 and 1996, respectively. He joined the IBM Research Division in 1996, initially working on data mining. He was subsequently involved in a variety of projects including network security, optimization, autonomic computing, and grid computing. Mr. Yoda's research interests lie in the area of reliable distributed systems, fault-tolerant distributed algorithms, and discrete algorithms in optimization problems.

Masami Amano *IBM Research Division, IBM Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan (amanom@jp.ibm.com).* Mr. Amano received B.S. and M.S. degrees in information science from Kyoto University. He joined the IBM Research Division in 2001 to work on optimization technologies such as production scheduling and transportation scheduling. His current interest is optimization for supply chain management in the manufacturing industry.