Scheduling Foundry Production without Mold Inventory

**Brent Austgen**  
Department of Mathematics, Rose-Hulman Institute of Technology, Terre Haute, IN, USA, brent.austgen@gmail.com

**Dana Hofheins**  
Pentair Water Solutions, Melbourne, Australia, Dana.Hofheins@pentair.com

**Allen Holder**  
Department of Mathematics, Rose-Hulman Institute of Technology, Terre Haute, IN, USA, holder@rose-hulman.edu

**Pankaj Upadhyay**  
Pentair Water Solutions, Melbourne, Australia, Pankaj.Upadhyay@pentair.com

**Abstract**  
A collection of mathematical programs is used to schedule production at a foundry that casts products without an inventory of sand molds. The problem of sequencing products is divided into two sequential sub-problems. The first distributes weekly demands to individual hours, and the second optimally sequences products per hour. Both phases are accomplished by solving mixed integer linear programs, and no specialized heuristic is needed. The resulting schedules detail the order in which casting molds should be created and how they should be split between two pouring lines so that products can be cast consecutively down each line. The two phases combine to reduce other production delays such as those caused by waiting for the molten metal to cool. Software development and implementation are discussed in addition to results on real data.

**Keywords** Operations Research; Foundry Scheduling; Coopr; Pyomo

---

1. Introduction

We consider the problem of optimizing production schedules at Pentair Water Solutions' Currimbin foundry in Queensland, Australia, which manufactures a wide array of parts in support of Pentair's business interests. Products are made by pouring molten metal into sand molds, called boxes, and sequencing these boxes to improve the manufacturing efficiency is the objective of this project. We present a two-phased, mathematical programming model that adheres to several manufacturing requirements. Results show marked potential improvements in efficiency, and initial projections suggest possible annual savings of half million dollars per year.

Foundry operations are highly variable, and the operations research community has investigated several production paradigms. See [1, 2, 3, 4, 5, 6, 7, 8, 9, 11] for similar research in the casting industry, and see [12] for a review of production scheduling in the related steel industry. The foundry we study receives its demands from a larger corporate partner, and such foundries are said to be captive. Moreover, the foundry has a single furnace, making it a small foundry, and it uses a single molten mix to create products. Much of the recent industrial engineering and operations research literature on related problems instead addresses...
market-driven foundries that need to react to market requests. Market-driven foundries mandate schedules that, at any given time, assign single products to particular furnaces. Different products in the market-driven setting require different metallurgical blends, and the guiding principals used to optimize production in this setting are to 1) reduce furnace downtime caused by altering the metallurgical blend, and 2) satisfy promised delivery dates.

The problem we solve for the Currumbin foundry differs from those in the literature. First, captive foundries typically separate the creation of molds from the sequencing of castings since molds can be created apriori and stored until needed. Mold creation in this setting is restricted by storage and/or creation time. Of course, mold inventory needs to be congruent with the production schedule, which is designed to maximize furnace use. The streamlined production process at the Currumbin foundry instead foregoes a mold inventory and casts products directly after molds are created. Hence, optimizing the casting sequence is intimately dependent on optimizing the sequence in which molds are created. Second, products do not need to be uniquely assigned to appropriate metallurgical blends, and instead, different products can be cast without costly delays necessitated by changing the mixture in the furnace. Third, the demands placed on the Currumbin foundry are not, at least directly, driven by the unknown and highly variable nature of the open market.

While the scheduling problem at the Currumbin foundry is in some ways simple due to there being a sole furnace with a sole metallurgical blend and to the fact that weekly demands are reasonably assigned by the larger Pentair corporation, the lack of a mold inventory to buffer the process of creating molds from the process of casting products presents a unique challenge. Moreover, while all products are created with the same molten metal, they differ with regard to the temperatures at which they must be cast, which means that the production sequence needs to agree with the cooling dynamics of the molten metal.

The production workflow we consider is depicted in Figure 1. Molds are created by compacting sand around a pattern. The machine that changes patterns holds two, one that is in use and another that can be swapped for a different pattern or that can be rotated into use. About four molds can be made with the active pattern in the time required to swap the inactive pattern. It is therefore ideal to consecutively create multiple molds with the active pattern to minimize the prolonged delays that might be incurred by waiting for a pattern to be swapped before continuing with the creation of molds. Molds proceed in sequence toward the furnace, at which point they are divided between two pouring lines. Molten metal is transported to the molds via a ladle, and products are cast by pouring the metal into the molds once the metal reaches an appropriate temperature. The ladle can alternate between the two pouring lines, but this movement costs an undue loss of heat. Castings proceed to a cooling area and are separated from the molds once they have sufficiently cooled. The warm sand is then recycled for new molds after being processed by a tumbler.

We sequence products to limit delays in mold creation and to bypass the current practice of using the ladle across the two pouring lines. There are two interlinked sequencing problems. One is to order the products at mold creation, which is not as straightforward as a rule like “make at least four boxes of each product at a time.” One reason is that many products have a demand below four boxes. Another reason is that products can be sequenced in strings of less than four and still not cause a delay. For example, six boxes of product A and two boxes of product B could be sequenced without a delay as,

\[ A, A, B, B, A, A, A, A. \]

Since the two boxes of product B are followed by four boxes of product A, the pattern for product B could be swapped without a delay while the last four boxes of product A are created.

Any forthcoming sequence of molds should be the interleave of two, not necessarily contiguous, subsequences for the two pouring lines. These subsequences must be ordered so that the products can be cast consecutively down each pouring line. This restriction has three implications, and the products must be sequenced so that...
• temperature requirements per ladle pour are in non-increasing order,
• the temperature requirements align with the rate at which the molten metal cools as it is poured, and
• the amount of metal required to cast a collection of products per ladle pour must not exceed the ladle’s capacity.

An important observation is that these requirements are defined in terms of the ladle, and it is this observation from which we build.

Our solution methodology divides the optimization process into two sequential pieces, each of which is handled by solving a collection of mixed-integer linear programs (MIPs). The input to the decision process is a week’s demands as detailed by a list of products, product weights (the amount of metal required per part), product temperatures, and the time required to pour a casting. The unit is that of a box, which contains a mold that might make several units of a single product. Each box is associated with a unique product, the number of products produced, the temperature at which the product must be cast, the amount of sand used to construct the mold, and the number of boxes required to satisfy the week’s demands.

Our first stage distributes the week’s demands over individual work hours. We use an hour as the standard unit of production because all measures of efficacy, cost, and labor are naturally stated in terms of hours. For instance, one of management’s goals is to have the foundry process 21 boxes per hour, a desire that helps guide the distribution of demands. The second stage of the scheduling process decides how to optimally sequence each hour’s products to remove ladle crossovers and to limit delays in mold creation. We optimally solve a MIP for each hour to construct the schedule. The MIP is defined in terms of ladle pours that innately impose several of the manufacturing demands, and this design strategy transitions the enforcement of several manufacturing demands from traditional mathematical constraints to the computational burden of constructing the collection of product subsequences that satisfy manufacturing limitations.

The difficulty of solving real-world, large-scale scheduling problems is well established [3, 10], and heuristics are commonplace, see for example the variety of heuristics used in [2, 4, 7, 8, 9, 11]. One of the benefits of our methodology is that the MIPs can be solved with off-the-shelf software, and hence, we do not require a tailored heuristic to efficiently generate quality schedules. We have worked with several free and commercial solvers, and our computational effort clearly shows that the MIP of the first stage is more difficult than
the MIPs of the second. While all solvers we tested were able to efficiently solve the MIPs of
the second stage, Gurobi\textsuperscript{©} has distinguished itself by solving the MIP of the first. Moreover,
the second stage MIPs are degenerate, and among the solvers we tested, Gurobi’s solutions
in the second stage were more practical to the application than other (free) solvers. For
these reasons all numerical work was accomplished with version 5 of Gurobi.

The following section presents the MIPs associated with the two stages of design. Sections 3 and 4 address software design and computational results. We conclude in Section 5. The software of this work is called Production Scheduling Optimization (ProSO), and it is available at http://holderfamily.dot5hosting.com/aholder/research/

2. A Two-Phased Mathematical Programming Model

The software is designed to be used per week at the Currimbin foundry, which routinely
equates to about 1,350 boxes over 100 different product types. The number of boxes is
divided by the desired rate, which is 21 boxes per hour, and for an average week this
calculation shows a planning horizon of 60 to 70 hours – including a few extra if needed.
The first stage of the planning process distributes the weekly demands into these 60+ hours
so that each hour attempts to receive a collection of products that can be well scheduled
in the second stage. All subsequences satisfying the manufacturing demands are calculated
once hours are populated with demands. These subsequences are ordered lists of boxes that
can be cast with a single ladle pour, and they are stored in a specialized tree that permits
efficient queries. The second stage problem is to optimally select a collection of ladle pours
for each hour.

2.1. Stage I: Distribute Demands

The MIP of Stage I is defined in terms of the sets and parameters listed in Table 1. The
decision variables are,

\begin{itemize}
\item \(x_{ph}\) – integer amount of product \(p\) to make in hour \(h\)
\item \(y_{ph}\) – binary variable to indicate if product \(p\) is made in hour \(h\),
\item \(z_{th}\) – a (real) variable upper bound that helps to ensure that hour \(h\)
\item \(f_{p^{'}h}\) – binary variable to indicate if product \(p^{'}\) produces
\item \(s_{h}\) – binary variable to indicate if hour \(h\) is used in the production
\end{itemize}


The objective is a scalarization of two sub-objectives. The first term expresses the desire
to distribute demands to the individual hours in groups of size at least \(\beta\) because this is the
quantity that permits pattern changes without concern. The second term encourages the
temperature distribution of the products in each hour to resemble the distribution of the

Table 1. Sets, indices, and parameters for Stage I’s MIP.

<table>
<thead>
<tr>
<th>Index Sets</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>set of products, indexed by $p$</td>
</tr>
<tr>
<td>$P'$</td>
<td>subset of products whose (weekly) demands are at least $\beta$ boxes (see parameters below), indexed by $p'$</td>
</tr>
<tr>
<td>$T$</td>
<td>set representing unique pouring temperatures, indexed by $t$</td>
</tr>
<tr>
<td>$H$</td>
<td>set of production hours, indexed by $h$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_p$</td>
<td>weekly box demand for product $p$</td>
</tr>
<tr>
<td>$w_p$</td>
<td>weight of metal for product $p$</td>
</tr>
<tr>
<td>$k_p$</td>
<td>pouring temperature for product $p$</td>
</tr>
<tr>
<td>$u_t$</td>
<td>value of pouring temperature $t$</td>
</tr>
<tr>
<td>$r_t$</td>
<td>percentage of weekly demand poured at temperature $t$</td>
</tr>
<tr>
<td>$B_{min}$</td>
<td>minimum number of boxes to produce per hour</td>
</tr>
<tr>
<td>$B_{max}$</td>
<td>maximum number of boxes to produce per hour</td>
</tr>
<tr>
<td>$D_{max}$</td>
<td>maximum number of boxes of a single product to produce per hour</td>
</tr>
<tr>
<td>$C_{max}$</td>
<td>maximum number of different product types to produce per hour</td>
</tr>
<tr>
<td>$W_{max}$</td>
<td>maximum weight of metal to be poured per hour</td>
</tr>
<tr>
<td>$\beta$</td>
<td>the number of boxes that can be created while swapping a pattern</td>
</tr>
<tr>
<td>$\eta$</td>
<td>objective weight</td>
</tr>
</tbody>
</table>

weekly demands. The MIP model is,

$$\max \left\{ \sum_{p'} \sum_{h} f_{p'h} - \eta \sum_{th} z_{th} \right\}$$

s.t.

$$d_p = \sum_{h} x_{ph}, \quad \forall p \in P$$

$$B_{min} s_h \leq \sum_{p} x_{ph} \leq B_{max} s_h, \quad \forall h \in H$$

$$-z_{th} \leq \sum_{p: k_p = u_t} x_{ph} - r_t \sum_{p} x_{ph} \leq z_{th}, \quad \forall t \in T, \forall h \in H$$

$$x_{p'h} \geq \beta f_{p'h}, \quad \forall p' \in P', \forall h \in H$$

$$x_{ph} \leq D_{max}, \quad \forall p \in P, \forall h \in H$$

$$\sum_{p} y_{ph} \leq C_{max}, \quad \forall h \in H$$

$$y_{ph} \leq x_{ph} \leq d_p y_{ph}, \quad \forall p \in P, \forall h \in H$$

$$\sum_{p} w_p x_{ph} \leq W_{max}, \quad \forall h \in H$$

$$f_{p'h} \leq y_{p'h} \quad \forall p' \in P', \forall h \in H$$

$$x_{ph} \in \mathbb{Z}_+, \quad \forall p \in P, \forall h \in H$$

$$s_h, f_{p'h}, y_{ph} \in \{0,1\}, \quad \forall p \in P, \forall p' \in P', \forall h \in H$$

$$z_{th} \in \mathbb{R}_+, \quad \forall t \in T, \forall h \in H,$$

where $\mathbb{Z}_+$ and $\mathbb{R}_+$ are the nonnegative integers and reals, respectively. We have experimented with numerous variations of Stage I’s MIP, many of which are discussed in Section 3.
2.2. Stage II: Construct Production Sequences

The sets and parameters defining the MIP of Stage II are listed in Table 2. The collection of products and their demands, i.e. \( P^h \) and \( d_p^h \), are decided by Stage I. All sets and parameters that have a superscript \( h \) depend on the specific hour being scheduled. The set of slots is fixed across all hours due to the intrinsic restriction of there being two pouring lines and a furnace capacity of 2,700 kg per hour. Since the ladle holds 1,000 kg, we have less than 3 full ladle pours per hour, and so \( S = \{1, 2, 3\} \) indexes the three pouring slots of each hour.

The initial calculation of Stage II constructs a data tree that enumerates all subsequences that can be poured with a single ladle fill. Each node in the tree is defined by the state of the ladle, which is the temperature and weight of the molten metal. Downward branches correspond to casting products, which reduces the ladle’s temperature and weight. Additional boxes are added to a production sequence only if they satisfy the temperature and weight requirements of the product. A path from the root node to any other node is a potential ladle pour, and since the manufacturing demands are imposed as the tree is constructed, we do not have to logically constrain the MIP to enforce manufacturing protocols.

We consider a three product example to illustrate a data tree. Assume that a box of product A requires 45 kg of metal and a pouring temperature of \( 1390^\circ \pm 5^\circ \). Likewise assume that products B and C have respective weight requirements of 50 kg and 32 kg and temperature requirements of \( 1380^\circ \pm 5^\circ \) and \( 1370^\circ \pm 5^\circ \). We assume that pouring a box of product A decreases the metal’s temperature by \( 3^\circ \), of product B by \( 4^\circ \), and of product C by \( 2^\circ \) (these are a bit exaggerated for illustrative purposes). A partial data tree is depicted in Figure 2.

Each edge of the tree is associated with casting a single box of product A, B, or C, and the downward nodes are the states of the ladle as if the product had been cast. The dashed edges at the bottom show which products are possible next. The path in bold is the feasible sequence A, C, C, which has an embedded delay caused by a pause to let the metal cool until the first box of product C can be cast after product A. The ladle’s temperature of \( 1387^\circ \) after pouring the first box of product A exceeds the range of appropriate temperatures for product C, so the ladle pauses to cool to an appropriate \( 1375^\circ \). After pouring a box of product C the temperature further cools to the stated \( 1373^\circ \). There are several instances of similar pauses in the tree.

The desired production rate is 21 boxes per hour, and we consider an illustrative hour with a total demand of 19 boxes to promote the sequencing problem. Assume the demands for five products are: 4 boxes of product A, 5 boxes of product B, 2 boxes of product C, 5 boxes of product D, and 3 boxes of product E. We further assume that swapping a mold pattern can be accomplished while 4 molds are created with the pattern in use, i.e. \( \beta = 4 \) in Stage I, which is the actual value used in the application. Consider the following two collections of feasible ladle pours that could satisfy demand,
In both options we are asked to arrange the pouring sequence to reduce delays in the creation of molds. Importantly, all molds must proceed to one of the two pouring lines once created. We arrange the ladle pours to denote the number of boxes that are cast for each product.

\[
\begin{array}{c|c|c}
\text{Option 1 Product} & \text{Option 2 Product} \\
\hline
\text{Slot} & 1 & 1 \\
A & 2 & 2 \\
B & 3 & 3 \\
C & 2 & 2 \\
D & 1 & 1 \\
E & 2 & 2 \\
\hline
\text{total} & 4 & 4 \\
\end{array}
\]

In both options ladle 2 is slotted first, ladle 1 is slotted second, and ladle 3 last. The first option is not as favorable as the second because the single box of product B in slot 3 interferes with the creation of molds. Creating 4 molds of product A and then 5 of product B is not possible in the first option since we are unable to push the last box of product B onto one of the two pouring lines. Hence, the first option necessitates that molds be created, for example, in an order like

A, A, A, A, B, B, B, B, C, C, D, D, D, D, D, E, E, E.

The sole box of product B in the third slot, which is underlined, is problematic, for without an inventory of molds it can’t be created until one of the first two ladle pours in the first two slots is complete. Unfortunately, the ladle pours of the first option force a delay independent of how they are slotted because we don’t have time to swap the unused pattern as the other pattern creates at least $\beta = 4$ molds.

The second option does not suffer from the same problem, and in this case molds can be created exactly in the order of demands, i.e. in the order

A, A, A, A, B, B, B, B, B, C, C, D, D, D, D, D, D, E, E, E.

The problematic box of product B from option 1 now adjoins the other boxes of product B and causes no delay. The two boxes of product C do not cause a delay because they are followed by 4 boxes of product D, and hence, the pattern for product C can be swapped for
product E’s pattern as the 4 molds of product D are created. The last 3 boxes of product E do not cause a delay within the hour but might create a delay should this hour be followed by another.

The goal of Stage II is to solve a MIP whose solution permits the molds of each product to be made without interruption. In the best of all situations, the demands from Stage I would all be greater than $\beta$. In this case a Stage II solution that sequenced each product’s molds contiguously would automatically have no production delays, even as the schedule continued into the next hour. This best case situation is not generally possible, and indeed, as product C exhibits in the example above, the demands do not need to be in units of at least $\beta$ to remove delays. The point is that as long as the demands from Stage I are reasonable, then we can limit further delays by seeking schedules that sequence products contiguously.

The decision variables of Stage II’s MIP are,

\[ w_{is} \]  
binary variable indicating if ladle pour \( i \) is assigned to pouring slot \( s \)

\[ a_{sp} \]  
integer amount of product \( p \) cast in pouring slot \( s \)

\[ u_p \]  
binary variable to denote the product spread of the ladle pour assigned to the first slot

\[ l_p \]  
binary variable to denote the product spread of the ladle pour assigned to the third slot

\[ e_p \]  
a (real) variable bound used to encourage an even distribution of castings over ladle pours.

The \( a_{sp} \) variables are the values assigned, for example, to the tables in (1). The binary variables \( u_p \) and \( l_p \) indicate the spread of products in the first and third slots, and they are logically constrained to be non-increasing and non-decreasing. For example, the \( u_p \) and \( l_p \) variables of the tables in (1) would be, respectively,

<table>
<thead>
<tr>
<th>Option 1</th>
<th>Option 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>Product</td>
</tr>
<tr>
<td>A B C D E</td>
<td>A B C D E</td>
</tr>
<tr>
<td>Slot 1</td>
<td>Slot 1</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>1 1 1 0 0</td>
</tr>
<tr>
<td>← u</td>
<td>← u</td>
</tr>
<tr>
<td>Slot 3</td>
<td>Slot 3</td>
</tr>
<tr>
<td>0 1 1 1 1</td>
<td>0 0 1 1 1</td>
</tr>
<tr>
<td>← l</td>
<td>← l</td>
</tr>
</tbody>
</table>

The first option is problematic because the ladle pour of the first slot overlaps the ladle pour of the third slot in more than the single product at the end of the first slot and at the start of the third, i.e. the schedule overlaps at products B and C instead of just B or C. The second option has no such hindrance. The structure of option 2 is what we impose by enforcing the monotonicity of \( u_p \) and \( l_p \) and by requiring that their sum be no more than \(|P_h| + 1\). The ladle pour of the second slot can overlap the other two arbitrarily without forcing a delay in mold creation.

Ladle pours are selected on three criteria, which like Stage I are aggregated into a single objective. The criteria are to best meet the hour’s demands, to reduce the number of ladle pours, and to evenly distribute a product’s demands over the selected ladle pours. Achieving the hourly demands from Stage I can be impossible with the restriction that we can only use three ladle pours that satisfy temperature and weight requirements. Hence, we limit hourly production to be no more than the assigned demand and include in the objective a term to encourage the satisfaction of demand. This modeling construct typically leaves a small amount of unsatisfied demand per week, but the benefit is that the majority of hours can be scheduled with two ladle pours and without costly delays in mold creation. The third goal to evenly distribute each product’s castings over the ladle pours is an attempt to reduce the number of embedded pauses in a pour sequence. Recall that the data tree in Figure 2 includes delays caused by waiting for the metal to cool, and the third objective term encourages each
ladle pour to include a collection of products with a contiguous, non-increasing temperature sequence so that castings can be made as the metal cools.

The MIP of Stage II is,

$$\min \left\{ \lambda \sum_p \left( d^h_p - \sum_s a^s_p \right) + \rho \sum_{is} w_{is} + \sum_p e_p \right\}$$

s.t.

$$\sum_i t^h_ip w_{is} = a^s_p, \quad \forall p \in P^h, \forall s \in S$$

$$\sum_s a^s_p \leq d^h_p, \quad \forall p \in P^h$$

$$\sum_i w_{is} \leq 1, \quad \forall s \in S$$

$$\sum_p (u_p + l_p) \leq |P^h| + 1$$

$$u_{p-1} \geq u_p, \quad \forall p \in P^h \setminus \{p_0\}$$

$$l_{p-1} \leq l_p, \quad \forall p \in P^h \setminus \{p_0\}$$

$$a^1p \leq D_{\text{max}} u_p, \quad \forall p \in P^h$$

$$a^3p \leq D_{\text{max}} l_p, \quad \forall p \in P^h$$

$$a^{s'}p - a^{s''}p \leq e_p, \quad \forall p \in P^h, \forall s', s'' \in S \ni s' \neq s''$$

$$w_{is}, u_p, l_p \in \{0,1\} \quad \forall i \in I^h, \forall s \in S, \forall p \in P^h$$

$$a^s_p \in \mathbb{Z}_+ \quad \forall s \in S, \forall p \in P^h$$

$$e_p \in \mathbb{R}_+ \quad \forall p \in P^h.$$ 

Adaptations are again possible, and we comment on some in the next section.

3. Implementation

The mathematical programs of Section 2 provide clean, abstract models of the foundry’s manufacturing process, but extrapolating symbolic models to a trustworthy product is often tedious and important innovation. ProSO’s development has required care, and in this section we comment on some of the more notable aspects of its development. We consider both the computer science and software engineering elements as well as ProSO’s model development.

The ProSO software was entirely written in python, which was selected because it allowed all elements to be written in a single, cross-platform language. Through numpy we further had intuitive matrix and vector operations to facilitate natural model development. We used pyomo\(^2\)\(^3\) for model generation, which worked seamlessly with the other python constructs. Many MIP solvers link with pyomo, and as noted earlier, we found Gurobi to be a dependable stalwart after experimenting with several options. Data is read and solutions are written with python’s xls module, and the data tree is a custom class designed for ProSO. The ProSO package is run through a GUI that is designed with python’s Tk widget module Tkinter.

Code development was nontrivial, and the final product is the result of several re-writes. Some of the implementation concerns were difficult to overcome. The data tree has gone through several revisions, and its recursive implementation required care to debug. We have experimented with time frames greater than one hour, but these can cause computational delays and/or memory concerns as the tree grows due to the increased number of options. A one hour period with about 21 boxes over 5 or 6 product types seems to be the “sweet
spot,” which thankfully works well with the application. The python module pandas was used to read, store, and search data on all but the last version of ProSO. However, the pandas structures were found to be substantially slower than common python constructs, and major speed-ups were achieved by authoring our own classes that made use of standard data structures such as dictionaries and lists.

ProSO’s MIPs have been repeatedly revised, and many adaptations have been tested. The authors at The Rose-Hulman Institute of Technology originally missed the lack of a mold inventory, and the initial attempt used the data tree’s ladle pours to form an integer version of the set-cover problem. This tactic proved numerically possible, but the results were unrealistic because the schedules couldn’t be realized without undue delays in mold creation. The models of Section 2 were developed after this initial failure.

The MIP of Stage I has proven to be substantially more difficult to solve and, in the authors’ experience, is the more important of two with regard to modeling sensitivity. Numerous adaptations have been tested in search of a model that better distributes weekly demands and that solves efficiently with many solvers. For instance, a reasonable adaptation might include constraints to ensure that each product in $P \setminus P'$ be assigned to a single hour. However, such a constraint does not routinely lead to better schedules. Another alternative would be to add an objective term to reduce the number of hours in which production occurs. Such an adaptation can reduce the number of production hours, which can lead to improved efficiency metrics like the box per hour rate. The downside is that the total hourly demands are routinely higher in a way that degrades the hourly schedules. The problem is that ladle pours should constitute pouring sequences that approach the ladle’s weight capacity, and higher hourly demands often necessitate a small-weight pour, which can jeopardize product quality.

The last construct added to Stage I’s MIP was the collection of elastic constraints,

$$- z_{th} \leq \sum_{p:k_p=u_t} x_{ph} - r_t \sum_{p} x_{ph} \leq z_{th},$$

(2)

together with their corresponding objective term $\sum_{th} z_{th}$. The goal was to minimize the 1-norm temperature discrepancies between the weekly and hourly demands, and once this modeling element was included, the hourly demands regularly had contiguous temperature spreads that helped reduce delays caused by waiting for the ladle to cool. Various other modeling attempts to achieve the same goal, for example an inf-norm counterpart, were less successful.

We halt Stage I’s solve once the duality gap is less than 5% or once a time limit of 300 seconds is reached. All but one of the examples from the foundry terminated with a gap less than 5%, and the sole example that terminated due to the 300 second limitation halted with a gap of 5.16%. Gurobi’s performance on Stage I’s MIP was superb, and the demand distributions from these near optimal solutions were excellent. However, Gurobi’s success seems to be predicated on the explicit statement of two tacit modeling elements. The constraints $y_{ph} \leq x_{ph}$ and $f_{p'\ell} \leq y_{p'\ell}$ are both implied by domain restrictions and other constraints; however, removing one or both degrades Gurobi’s ability to nicely distribute the weekly demands with the aforementioned termination criteria.

Fewer adaptations of the second stage MIP have been considered since its focus on interleaving ladle pours to best satisfy demands is less opaque than is the role of the first stage. The second stage MIP permits underproduction, although the first objective term attempts to minimize the number of boxes that go unscheduled. This design strategy is important since if the inequality $\sum a_{sp} \leq d_p^h$ had been replaced with an equality, then some problems would have been infeasible. The infeasibility is caused by the inability to satisfy the hourly demands with three or fewer feasible ladle pours. Potential underproduction is a limitation of the solution methodology, but the results of the next section show that the underproduction is typically small (less than 3.01% in our tests). The manufacturing process is not
foolproof, and the number of boxes left unscheduled is well within the tolerance of boxes that would be typically managed under special circumstances anyway. The objective term \( \sum p e_p \) in the second stage MIP importantly coincides with the intent of (2) in the first stage to reduce delays due to waiting for the ladle to cool.

The second stage MIPs are solved to optimality within a few seconds by numerous solvers, although solutions tend to differ due to degeneracy. As with the first stage, we have found Gurobi to be an efficient solver of the second stage MIPs in a way that nicely represents the manufacturing process. The sequential scheduling process as expressed by the first and second stage MIPs works symbiotically to design quality schedules in a few moments on real problems. Results are presented in the next section.

4. Results

ProSO was tested on 5 weekly data sets as described in Table 3. Parameters were selected to agree with the manufacturing process and were assigned the following default parameters in all cases,

\[
\begin{array}{cccccc}
B_{\min} & B_{\max} & D_{\max} & C_{\max} & W_{\max} & \beta \\
21 & 22 & 8 & 5 & 2700 & 4
\end{array}
\]

The ladle’s capacity was 1000 kg, and the time to move the ladle between consecutive boxes was assumed to be 3 seconds. The ladle’s cooling rate was approximated linearly as \(-10^\circ\) per minute. There were at least four temperature classes in all cases, and \(T\) was a subset of \{1390\(^\circ\), 1380\(^\circ\), 1370\(^\circ\), 1360\(^\circ\)\} in every instance. In some cases temperature requirements went lower, e.g. down to 1340\(^\circ\). We assumed that a product could be cast within \(\pm 5^\circ\) of its required temperature.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Number of Boxes</th>
<th>Number of Production Products</th>
<th>Number of Hours</th>
<th>Number of Boxes Scheduled</th>
<th>Number of Boxes Unscheduled</th>
<th>Number of Boxes per Hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1456</td>
<td>77</td>
<td>69</td>
<td>1445</td>
<td>11 (0.76%)</td>
<td>20.942</td>
</tr>
<tr>
<td>2</td>
<td>1346</td>
<td>82</td>
<td>64</td>
<td>1341</td>
<td>5 (0.37%)</td>
<td>20.953</td>
</tr>
<tr>
<td>3</td>
<td>1395</td>
<td>78</td>
<td>66</td>
<td>1353</td>
<td>42 (3.01%)</td>
<td>20.500</td>
</tr>
<tr>
<td>4</td>
<td>1286</td>
<td>145</td>
<td>61</td>
<td>1271</td>
<td>15 (1.17%)</td>
<td>20.836</td>
</tr>
<tr>
<td>5</td>
<td>1222</td>
<td>129</td>
<td>58</td>
<td>1214</td>
<td>8 (0.65%)</td>
<td>20.931</td>
</tr>
</tbody>
</table>

The objective weights \(\eta\) (Stage I) and \(\lambda\) and \(\rho\) (Stage II) were tuned by iteratively solving each data set over a wide range of settings and assessing the computational and practical outcomes listed in Table 3. In the first stage the value of \(\eta\) needs to be ‘small,’ otherwise Gurobi was unable to solve the problem. Indeed, duality gaps of over 100% after 300 seconds were observed if \(\eta\) wasn’t sufficiently small, and the resulting schedules left many boxes unscheduled. A value of \(\eta = 0.025\) worked well across all tests including queries into model adaptation, and this is the value of \(\eta\) in all our results. The parameters of the second stage generally gave reasonable schedules as longs as \(\rho > \lambda > 1\), and we set the default values to \(\rho = 10\) and \(\lambda = 5\), which lead to good results across all tests.

The dominant role of \(\rho\) in the second stage illustrates the importance of using 2 ladle pours per hour instead of 3. As long as weekly demands are distributed well, most of the hourly demands can be satisfied with 2 ladle pours, see Table 4. An hour with 2 ladle pours is advantageous because,

- each ladle pour (typically) better uses the weight capacity of 1000 kg of molten metal per ladle,
- probable rejection due to imperfections is lessened,
• the linear model of cooling remains accurate,
• an 'hour' might be collapsed into a shorter period of time.

The last point suggests that heightened efficiencies might be possible. The furnace capacity is 2700 kg per hour, and if an hour's demands can be satisfied with 2 ladle pours, then the furnace can supply the molten metal quicker than the hour time frame. The results in Table 4 show that an average hour from ProSO requires about 1,704 kg of metal, which the furnace can provide in about 38 minutes. Running the foundry for the products that we are scheduling costs approximately $1,400 per hour, and if the rest of the manufacturing process could realize the efficiency of reducing each hour to 38 minutes, then the average savings on the five data sets is about $56,161 per week. The other manufacturing processes are of course in play, e.g. mold creation, pattern retrieval and storage, sand reclamation, etc..., and making all of these simultaneously more efficient is difficult.

Table 4. 269 of 318 production hours efficiently use 2 ladle pours instead of 3.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Production Hours</th>
<th>Two Ladles</th>
<th>Three Ladles</th>
<th>Average Weight (kg) per Hour</th>
<th>Average Weight (kg) per Ladle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>69</td>
<td>48</td>
<td>21</td>
<td>1691</td>
<td>734</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
<td>62</td>
<td>2</td>
<td>1549</td>
<td>763</td>
</tr>
<tr>
<td>3</td>
<td>66</td>
<td>57</td>
<td>9</td>
<td>1790</td>
<td>838</td>
</tr>
<tr>
<td>4</td>
<td>61</td>
<td>51</td>
<td>10</td>
<td>1756</td>
<td>811</td>
</tr>
<tr>
<td>5</td>
<td>58</td>
<td>51</td>
<td>7</td>
<td>1733</td>
<td>817</td>
</tr>
<tr>
<td>Totals</td>
<td>318</td>
<td>269</td>
<td>49</td>
<td>1704</td>
<td>793</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td></td>
<td>Average</td>
<td>Averages</td>
</tr>
</tbody>
</table>

Schedules are output as xls files that provide overviews as well as detailed hourly sequences. Although formatted slightly differently, a typical hour is depicted in Table 5. The P indicates the end of a ladle pour, and if there had been three ladle pours, then one of the two lines would have had two Ps. This hourly schedule has the favorable quality that there are no delays caused by skipping a temperature class, which is representative of the majority of hourly schedules.

Table 5. An example hour's schedule.

<table>
<thead>
<tr>
<th>Product (cast sequence −→)</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>B</th>
<th>B</th>
<th>B</th>
<th>B</th>
<th>B</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line 2</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Weight</td>
<td>84</td>
<td>84</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>Temp.</td>
<td>1390</td>
<td>1390</td>
<td>1380</td>
<td>1380</td>
<td>1380</td>
<td>1380</td>
<td>1380</td>
<td>1380</td>
<td>1380</td>
</tr>
<tr>
<td>Line 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Line 2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>130</td>
<td>130</td>
<td>130</td>
<td>130</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>Temp.</td>
<td>1370</td>
<td>1370</td>
<td>1370</td>
<td>1370</td>
<td>1360</td>
<td>1360</td>
<td>1360</td>
<td>1340</td>
<td></td>
</tr>
</tbody>
</table>

Scheduling the hours once the solution of the first stage is complete is embarrassingly parallel, and while we have not needed to parallelize the computation for speed, the individuality of the hours is practically favorable. One reason is that the hourly schedules have
to be manually interleaved with molds for larger products, and while we don’t address scheduling the larger molds, the flexibility of inserting them within the schedule at will is handy. Another reason is that the beginning of every shift requires special consideration as sand is reclaimed. Once products are removed from their molds, sand is processed through a tumbler, and if the tumbler isn’t sufficiently warm, then the process can clog and halt production. The tumbler is warmed prior to the start of production, but the initial boxes should be selected so that their warm sand aids the warming process. We assign each hour a ‘hot sand index,’ with 1 being the coolest and 5 being the hottest. We rank each hour according to \[ \sum_{p \in P} k_p \cdot w_p / s_p, \] where \( s_p \) is the amount of sand in a mold of product \( p \). The top 20\% of the hours receive a value of 5, the second 20\% a value of 4, and so on. Since the hours are interchangeable, hours with hot sand indices of 5 can be moved to the front of shifts.

5. Conclusion

The ProSO software divides the problem of scheduling products at a small, captive foundry into two tasks, both of which are completed by solving MIPs. The first task distributes the weekly demands into individual hours, and the second sequences the products to ensure temperature and weight requirements. The Currumbin foundry operates without a mold inventory, and hence, molds must be used in the order in which they are made. The ProSO software sequences products so that mold sequences are split among two pouring lines without the need to store molds. Moreover, the pouring sequences are designed to ensure that the ladle does not have to move between the two pouring lines and so that delays are minimized. ProSO’s use is estimated to save $500,000 per year in direct and ancillary improvements, i.e., policy changes that will help realize the benefits of the schedules.

There are several places for improvement. Maybe most important would be to post analyze the hourly schedules to see if adjustments might decrease unmet demand. Another might be to sequence the hourly schedules themselves to aid weekly planning. The possibilities are many and important as foundry manufacturing becomes an increasingly efficient and competitive market.

References

