Integrated Cutting and Packing Heterogeneous Precast Beams Multiperiod Production Planning Problem

Kennedy Araújo · Tiberius Bonates · Bruno Prata

Received: date / Accepted: date

Abstract We introduce a novel variant of cutting production planning problems named Integrated Cutting and Packing Heterogeneous Precast Beams Multiperiod Production Planning (ICP-HPBMPP). We propose an integer linear programming model for the ICP-HPBMPP, as well as a lower bound for its optimal objective function value, which is empirically shown to be closer to the optimal solution value than the bound obtained from the linear relaxation of the model. We also propose a genetic algorithm approach for the ICP-HPBMPP as an alternative solution method. We discuss computational experiments and propose a parameterization for the genetic algorithm using D-optimal experimental design. We observe good performance of the exact approach when solving small-sized instances, although there are difficulties in finding optimal solutions for medium and large-sized problems, or even in finding feasible solutions for large instances. On the other hand, the genetic algorithm could find good-quality solutions for large-sized instances within short computing times.

Keywords precast beams · modular construction · integer linear programming · metaheuristics · genetic algorithms

This work was developed while the corresponding author was a master degree student at Federal University of Ceará

K. Araújo
Department of Applied Mathematics
University of Sao Paulo
E-mail: kennedyanderson94@gmail.com

B. Prata
Department of Industrial Engineering
Federal University of Ceará
E-mail: baprata@ufc.br

T. Bonates
Department of Statistics and Applied Mathematics
Federal University of Ceará
E-mail: tb@ufc.br
1 Introduction

Nowadays, concrete precast production is increasingly trending in constructions sites. There are great advantages of using such kind of production, such as better and cheaper elements, and a potential to severely shorten construction time as compared to conventional methods. The precast element we consider in this work is a concrete precast beam, which is a kind of beam that is cast in plants away from the construction site, in a controlled environment.

These beams are heterogeneous in the sense that they can vary with respect to curing time, length and the number of traction elements used. We refer to the problem of planning the production of such beams to fulfill the clients demand within a given time horizon as the Heterogeneous Precast Beams Multiperiod Production Planning Problem (HPBMPP).

Araujo et al. (2019) proposed four integer programming models for the HPBMPP, considering prestressed precast beams instead of conventional concrete precast beams. One of the proposed models minimizes the total idle capacity in the molds along the time horizon, two models to minimize the production makespan and one model for total completion time minimization. The authors also proposed several solution methods, in particular a size reduction heuristic that succeeded in finding high-quality solutions in shorter time and using less memory compared to exact methods.

In this work, we propose a variant model of the HPBMPP, which consists in the integration of the production of bars, which are used in the precast beam production, into the problem. We divide the bars in two groups: standard bars and leftovers. Standard bars are new bars of standardized lengths, and leftovers are a type of bar that cannot be readily used in the beam production but can be stored in stock to produce other bars in the future. In this study, we consider that both standard bars and leftovers vary with respect to length. The production of bars to be used in the beam production can be made by the cutting of standard bars or leftovers in stock, or by the process of cutting overlapping leftovers. The overlapping process consists in merging two or more leftovers to create a larger bar that can be cut to produce a bar of appropriate length that will be used in beam production. In this work, we only consider overlapping of two bars. To the best of our knowledge, the consideration of overlapping bars has not been previously studied.

We consider the integration into a single production planning problem of the cutting process of bars, or of overlapping bars, which must be packed in the molds for the production of a given demand of beams. We refer to this problem as the Integrated Cutting and Packing Heterogeneous Precast Beams Multiperiod Production Planning Problem (ICP-HPBMPP). Note that in this work we consider beams that are not prestressed. The mathematical model we propose is based on the model by Arenales et al. (2015), which deals with the cutting stock/leftover problem, and on the model by Araujo et al. (2019) for the HPBMPP. We consider that the bars needed to supply the beam production can be produced by cutting bars or leftovers in stock or by overlapping leftovers in stock. The stock is static, i.e., we are given an initial stock that is not replenished over the entire time horizon.

The ICP-HPBMPP is of practical interest because optimizing the production of prestressed beams has the potential effect of speeding up overall construction
time, while improving the usage of molds and bar stock, while minimizing bars loss. An economical usage of bar stock may result in a reduction of unused bars in the construction site, which can improve the production flow. Furthermore, the reduction of concrete and bar loss may lead to a positive impact in the environment. An optimized process allows factories to accept additional orders due to shorter lead times. Also, the production cost with an optimized process will be lower, which may lead to a reduction of the final product’s price, increasing competitiveness.

It is argued in (Araujo et al., 2019) that the HPBMPP is NP-hard since it includes, as a particular case, the classical one-dimensional cutting stock problem. Thus, the HPBMPP can become too difficult to solve as the dimension of instances increases. The computational results reported in Section 6 show that the ICP-HPBMPP can be difficult to solve to optimality, justifying the use of decomposition techniques and heuristic procedures to deal with the problem. This also suggests that the HPBMPP is interesting to be studied from a theoretical point of view.

The remainder of this paper is organized as follows. In Section 2 we discuss the literature of similar problems to the ICP-HPBMPP. In Section 3 we formally define the problem, propose an integer linear programming model for its solution, argue about its NP-hardness and propose a lower bound for its optimal objective function value. In Section 4 we present three constraint programming models for the generation of packing, cutting and overlapping patterns. In Section 5 we propose a genetic algorithm for the problem under study. In Section 6 we discuss several computational experiments conducted based on instances generated artificially and discuss the results of the proposed solution methods. In Section 7 we discuss the conclusions and contributions of this chapter, as well as point out research gaps and suggest future work.

2 Literature review

To the best of our knowledge ICP-HPBMPP is not defined in the literature, even though the problem has similarities with one-dimensional cutting stock problems (1DCSP) and one-dimensional packing problems (1DPP). On the order hand, 1DCSP, 1DPP, and their variants have been substantially studied in the literature.

As far as one-dimensional cutting and packing problems (C&P) are concerned, the studies of Gilmore and Gomory (1961) and Gilmore and Gomory (1963) proposed a column generation algorithm to solve the linear relaxation of large instances of 1DCSP. Such studies served as basis for a number of subsequent works. Stadtler (1990) studied the 1DCSP proposing a heuristic based on the solution of the linear relaxation supplemented by a one-pass branching up procedure. The authors validated the proposed heuristic approach, testing on benchmark instances and on a case of study. Dyckhoff (1990) introduced a typology of C&P problems, unifying notions in the literature to guide further research on particular types of those problems. Vance (1998) proposed two different branch-and-price approaches to find optimal solutions to the 1DCSP. Wässer et al. (2007) presented a new typology to categorize the types of C&P problems in the literature between years 1995 and 2004, introducing new categorization criteria. Trkman and Gradičar (2007) proposed a model for the multiperiod one-dimensional cutting stock problems (M1DCSP), considering the use of objects/leftovers in stock.
Poldi and Arenales (2010) proposed an integer linear model for the M1DCSP, implemented a column generation to solve the linear relaxation, and developed two rounding heuristics for finding integer solutions to the problem. Melega et al. (2018) proposed a mathematical model for the general integrated lot-sizing and cutting stock problem, and performed a vast classification of the literature of that problem, providing directions for future research.

Regarding the C&P problems and optimization approaches in precast production, De Castilho et al. (2007) described the problem of minimizing production costs for slabs of precast prestressed concrete joists and introduced a genetic algorithm to solve it. Prata et al. (2015) proposed an integer linear programming model for multiperiod production planning of precast concrete beams, which can be seen as a special case of the HPBMPP. Arenales et al. (2015) introduced a mathematical model for the cutting stock/leftover problem and suggested a column generation technique for finding the problem’s linear relaxation solution. Vassoler et al. (2016) proposed a mathematical model based on multiperiod cutting stock problem for the production planning problem of joists in trusses slabs industries. The authors suggested a solution method based on column generation to solve the linear relaxation of the problem. Araujo et al. (2019) proposed several integer linear programming models for the Heterogeneous Prestressed Precast Beams Multiperiod Production Planning Problem, showed its NP-hardness and suggested a constraint programming model for generating cutting patterns for the problem. The authors also carried out computational experiments to validate the performance of the integer linear programming models. Wang et al. (2018) introduced a two-hierarchy simulation-genetic algorithm hybrid model for precast production to ensure the on-time delivery of precast components minimizing the production cost, while simultaneously optimizing the resource waste under uncertainty in the processing time of each operation. The authors validated the model with a case study.

The problem which we study in this work is the integration of the cutting stock/leftover problem proposed by Arenales et al. (2015) and the HPBMPP introduced by Araujo et al. (2019). We explore its solution via exact methods and heuristics methods in the case where instances cannot be solved by the state-of-art solvers.

3 Problem statement

In this section we formally define the ICP-HPBMPP and propose an integer linear programming model for its solution based on the models proposed by Arenales et al. (2015) for the Cutting Stock/Leftover Problem (CSLP) and Araujo et al. (2019) for the Heterogeneous Prestressed Precast Beams Multiperiod Production Planning Problem (HPBMPP).

The ICP-HPBMPP consists in finding a feasible production planning to cast certain quantities of prestressed precast concrete beams, possibly of different types, while minimizing the total length of pieces of bars that cannot be used as leftover. A leftover is understood here as a piece of bar that can be cut or overlapped in the future to meet new demands and is not considered waste. The beam factory has a fixed amount of bars and bar leftovers with standard lengths in stock that can be used within a given time horizon.
Each mold can only be used to cast one type of beam at a time. It is possible, however, to simultaneously cast beams of different lengths in the same mold, as long as they are of the same type. The total length of the beams produced during a given period in a given mold cannot be greater than the mold’s capacity, and the total number of days required to complete the entire production cannot be greater than a given time horizon. After the process of cutting the bars, they are packed in the molds in order to produce the beams, note that different beam types can demand different numbers of bars. For this reason, we refer to this problem as a Cutting and Packing problem. The ICP-HPBMPP process can be seen in Figure 1.

Fig. 1 Cutting and packing production flowchart

As input of the problem we have a deterministic static demand of beams, with their respective types and lengths, stock of bars and stock of bars leftovers, with their respective lengths. The cutting planning of bars is made for the entire time horizon, resulting in more bars leftovers (which can be used in another production planning), and, possibly, incurring in bar loss. The bars cut will be packed in the molds for the beam production along the given time horizon. After the production of all beams demanded is met, there will usually be concrete waste of the beams and additional loss of bars.

3.1 Integer linear programming model

In order to define a model for the ICP-HPBMPP, we make use of the same parameters defined in (Araújo et al., 2019), as follows:
- $M$: number of molds in which the beams are produced;
- $T$: number of available periods to complete the production;
- $C$: number of beam types;
- $q_c$: number of distinct lengths of beams of type $c$, with $c = 1, \ldots, C$;
- $l(c, 1), \ldots, l(c, q_c)$: real numbers corresponding to the actual lengths of beams of type $c$, with $c = 1, \ldots, C$;
- $d(c, k)$: demand for beams of type $c$ and length $l(c, k)$, with $c = 1, \ldots, C$ and $k = 1, \ldots, q_c$;
- $t_c$: integer number corresponding to the curing time (in terms of periods) of beams of type $c$, for $c = 1, \ldots, C$;
– $L_m$: real number corresponding to the capacity of mold $m$, with $m = 1, \ldots, M$;
– $P_i = (c_i, (a_{i1}^1, \ldots, a_{iq_{ci}}^q))$: packing pattern, where $c_i$ stands for the beam type associated with pattern $P_i$ and $a_{i1}^1, \ldots, a_{iq_{ci}}^q$ represent the quantity of each beam of length $l(c_i, 1), \ldots, l(c_i, q_{ci})$ in patterns $P_i$, with $i = 1, \ldots, r, c_i = 1, \ldots, C$.

Note that $r$ represents the number of packing patterns;
– $P_0$: special pattern, which is used to denote that a mold is currently being used for the casting of a pattern that began in a previous period and whose production extends at least up to the current period.

Note that an idle mold (in other words, a mold that is not being used during a specific period) is not assigned the pattern $P_0$. In fact, it has no pattern assigned to it.

In order to refer to specific information on a given pattern $P_i = (c_i, (a_{i1}^1, \ldots, a_{iq_{ci}}^q))$, we define the following notation:
– $N_i(c, k)$: number of beams of type $c$ and length $l(c, k)$ that pattern $P_i$ includes.
  If $c = c_i$, then $N_i(c, k) = a_k$, with $k \in \{1, \ldots, q_{ci}\}$; otherwise, $N_i(c, k) = 0$, for any $k$.
– $u(P_i)$: capacity used by $P_i$, i.e. $u(P_i) = \sum_{k=1}^{q_{ci}} l(c_i, k) \cdot P_i(c_i, k)$, with $i = 1, \ldots, r$.
– $E_i$: number of periods required to produce the beams in $P_i$, with $i = 1, \ldots, r$.

This number equals the quantity of consecutive periods in which $P_i$ remains occupying a mold and is precisely the curing time of beams of type $c_i$, given by $t_{c_i}$.

Given a set of patterns $P = \{P_1, \ldots, P_r\}$, not including $P_0$, we define some important sets as follows:
– $Q(m)$: set containing the indices of the patterns in $P$ whose capacity does not exceed the capacity of the $m$-th mold: $Q(m) = \{i \in \{1, \ldots, r\} : u(P_i) \leq L_m\}$, for $m = 1, \ldots, M$. Note that the same pattern can belong to $Q(m)$ and $Q(m')$, with $m$ and $m'$ being two different molds of potentially distinct lengths.
– $Q^*(m) = Q(m) \cup \{0\}$;
– $S(j)$: set of indexes of the patterns that have curing time $j \in \{1, \ldots, R\}$, with $R = \max\{t_{c_i} : c = 1, \ldots, C\}$ being the largest curing time of all beam types present in the problem instance.

In what follows, we present the parameters that concern bars and bars leftover:
– $W$: number of different bar lengths;
– $V$: number of different bar leftover lengths;
– $H$: number of cutting patterns;
– $O$: number of overlapping patterns;
– $G$: number of different mold lengths;
– $b_1, \ldots, b_W$: bar lengths;
– $b_{W+1}, \ldots, b_{W+V}$: bar leftover lengths allowed. Note that this data narrows the types of cutting, and overlapping patterns;
– $L_1, \ldots, L_G$: mold lengths. Note that this data narrows the types of cutting, and overlapping patterns;
– $G(L_\gamma) = \text{set of molds which are of length } L_\gamma, \gamma = 1, \ldots, G$;
We present the decision variables below:

\[
x_{i,m,t} = \begin{cases} 
1, \text{ if the packing pattern } P_i \text{ starts to be used in} \\
mold m \text{ at period } t \text{ (and its usage, naturally,} \\\nlasts for } E_i \text{ periods);} \\
0, \text{ otherwise.}
\end{cases}
\]

\[
z_t = \begin{cases} 
1, \text{ if at least one mold is used at period } t, \text{ for } t = 1, \ldots, T; \\
0, \text{ otherwise.}
\end{cases}
\]

\[y_{h,w}: \text{ number of bars of length } b_w \text{ cut following a cutting pattern } I_h \in H_w.\]

\[y_{h,w,v}: \text{ number of bars of length } w \text{ cut following a cutting pattern } I_h \in H_w(v) \text{ generating a leftover of length } b_{W+v}.\]

\[o_{\mu}: \text{ number of times the overlapping pattern } O_{\mu} \text{ was used, } \mu \in \mathbb{O}.\]

Note that variables \(y_{h,w}, y_{h,w,v},\) and \(o_{\mu}\) are nonnegative integer decision variables. We present the integer linear programming model proposed for the ICP-HPBMFP as follows:

**(ICP)** min
\begin{align}
\lambda_1 \sum_{t=1}^{T} z_t + & \lambda_2 \sum_{w=1}^{W} \sum_{h \in H_w} f_{h,w} y_{h,w} \\
+ & \lambda_3 \sum_{w=1}^{W} \sum_{v=1}^{V} \sum_{h \in H_w(v)} f_{h,w,v} y_{h,w,v} \\
+ & \lambda_4 \left( \sum_{w=W+1}^{W+V} \sum_{h \in H_w} f_{h,w} y_{h,w} + \sum_{\mu \in \emptyset} f_{\mu} o_{\mu} \right) 
\end{align} 

s. t. \begin{align}
\sum_{i \in Q_2^*(m)} x_{i}^{m,t} & \leq 1, \quad m=1, \ldots, M, \quad t=1, \ldots, T \\
\sum_{m=1}^{M} \sum_{i \in Q_2^*(m)} \sum_{t=1}^{T-E_i+1} P_i(c,k) x_{i}^{m,t} & \geq d(c,k), \quad c=1, \ldots, C, \quad k=1, \ldots, q_c \\
(E_i-1)x_{i}^{m,t} & \leq \sum_{\alpha=1}^{E_i-1} x_{i}^{m,t+\alpha}, \quad m=1, \ldots, M, \quad t=1, \ldots, T-E_i+1, \quad i \in Q_2+1 \\
x_{0}^{m,1} & = 0 \quad m=1, \ldots, M, \\
x_{0}^{m,t} & \leq \sum_{\gamma=2}^{R} \sum_{j=\gamma}^{R} \sum_{i \in (Q(m) \cap S_j)} x_{i}^{m,t-\gamma+1}, \quad m=1, \ldots, M, \quad t=2, \ldots, T \\
M z_t & \geq \sum_{m=1}^{M} \left( \sum_{i \in Q_2^*(m)} x_{i}^{m,t} \right), \quad t=1, \ldots, T \\
\sum_{i \in Q_2^*(m)} x_{i}^{m,t} & \geq \sum_{i \in Q_2^*(m)} x_{i}^{m,t+1}, \quad m=1, \ldots, M, \quad t=1, \ldots, T-1 \\
\sum_{h \in H_w} y_{h,w} + \sum_{\mu \in \emptyset} a_{w,\mu} o_{\mu} & \leq e_w, \quad w=W+1, \ldots, W+V \\
\sum_{h \in H_w} y_{h,w} + \sum_{v=1}^{V} \sum_{h \in H_w(v)} y_{h,w,v} & \leq e_w, \quad w=1, \ldots, W \\
\sum_{w=1}^{W+V} \sum_{h \in H_w} a_{\gamma,h,w} y_{h,w} + \sum_{w=1}^{W+V} \sum_{h \in H_w(v)} a_{\gamma,h,w,v} y_{h,w,v} & + \sum_{\mu \in \emptyset(\gamma)} o_{\mu} = \sum_{m \in G_2(\gamma)} \sum_{t=1}^{T} D_{\gamma} x_{i}^{m,t}, \quad \gamma=1, \ldots, \Gamma \\
x_{i}^{m,t} & \in \{0,1\}, \quad m=1, \ldots, M, \quad t=1, \ldots, T, \quad i \in Q_2^*(m) \\
z_t & \in \{0,1\}, \quad t=1, \ldots, T \\
y_{h,w} & \in \mathbb{Z}_+, \quad w=1, \ldots, W, \quad h \in H_w 
\end{align}
The objective function (1) is divided into 4 terms. The first term is the makespan value. The second term defines the waste related to the use of new bars to produce the demand of bars. The third term describes the waste associated to the use of new bars to produce the bars required by beam production while creating new leftovers. Finally, the fourth term specifies the waste corresponding to the bar leftovers in stock that are used to produce the amount of bars required. Note that each term of (1) could alternatively be regarded as an independent objective function to be minimized. We obtain (1) using the weighted sum method, in which the parameters $\lambda_i \in \mathbb{R}_+$, with $i = 1, \ldots, 4$, indicate the weight of each objective function term. A solution that minimizes (1) is, therefore, a Pareto optimum (Marler and Arora, 2004).

Constraints (2) ensure that at most one pattern must be assigned to mold $m$ at period $t$, with the possibility of this pattern being $P_0$. Constraint set (3) requires that all demands must be satisfied. Constraints (4) force that, if pattern $P_i$ is initiated at period $t$, then the next $E_i - 1$ periods shall have the pattern $P_0$ assigned to them (the right-hand side of the constraint remains unconstrained, in case $x_{m,t} = 0$). Constraint sets (5) and (6) establish that $P_0$ shall only be used in mold $m$ if there is some pattern associated with a previous period in the same mold, whose production has not yet been completed.

Each constraint in set (7) ensures that variable $z_t$ must be 1 if period $t$ is used to produce beams. Constraints (8) force that there is no inactive period during beam production in the molds. This means that the production is continuous, i.e., if a mold is used it will be used with no interruption; in other words, if the production stops at a given mold and period, it will not resume in that mold at a subsequent period.

Constraints (9) establish that the number of bar leftovers cut plus the number of leftover bars used to produce bars via overlapping does not exceed the stock, note that the cutting of a leftover does not generate leftovers. Constraint set (10) ensures that the number of bars cut does not exceed the stock. Constraints (11) force that the amount of bars necessary to produce the beams is achieved, assuming that the required amount of bars is the number of bars used by the forms in the entire time horizon. Constraints (12)-(16) define the domains of the decision variables.

The model (ICP) has $O(MTr + WVH + O)$ variables and $O(q + MTr + V + W + \Gamma)$ constraints, with $q = \sum_{i=1}^{c} q_i$. Thus, depending on the total number of possible packing, cutting, and overlapping patterns, there may be an excessive number of variables and constraints in the model. We choose to limit the number of packing patterns, which are the more numerous type of pattern, in practice, by using only maximal packing patterns, used successfully by Vance (1998) and Araujo et al. (2019). We say that a pattern $P_i$ contains a pattern $P_j$ if $e_i = e_j$ and $a_k \geq a'_k$, with $k = 1, \ldots, q_c$.

Proposition 1 Restricting the model (ICP) to using only maximal packing patterns does not modify its set of optimal solutions.
Proof. Given an optimal solution to model (ICP) that is composed by non-maximal packing patterns we claim that replacing the non-maximal packing patterns with maximal ones that contain such patterns will not have an impact on the makespan. Indeed, the actual number of periods used to fulfill the demand will remain unaffected, given that all packing patterns of a given type have the same associated curing time. In the same way, there will be no changes to the cutting and overlapping patterns used in the optimal solution since the number of bars needed for the beam production will remain unchanged.

3.2 NP-hardness

To argue the ICP-HPBMPP hardness note that for instances where $D_c = 0$, for all $c = 1, \ldots, C$, constraints (9)-(11) are naturally fulfilled and all variables $y_{h,w}, y_{h,w,v}$ and $o_{\mu}$ are set to zero, reducing an instance of ICP-HPBMPP to an HPPMBPP instance involving the minimization of the makespan, up to a constant multiplicative factor. Consequently, the ICP-HPBMPP is a generalization of HPPMBPP, which is already known to be NP-hard (Araujo et al., 2019).

3.3 Objective function lower bound

Since the ICP-HPBMPP is a NP-hard problem, a lower bound for the optimal objective function value may help in evaluating the quality of feasible solutions in heuristic and exact methods. In order to simplify the presentation of our proposed lower bound for objective function (1) optimal value, we present the following notation. For a given $\gamma \in \{1, \ldots, \Gamma\}$ we define the following sets:

- $C_1\gamma = \{ f_{h,w}/a_{\gamma,h,w} : h \in H_w \land a_{\gamma,h,w} > 0 \land 1 \leq w \leq W \}$
- $C_2\gamma = \{ \alpha f_{h,w,v}/a_{\gamma,h,w,v} : h \in H_w(v) \land a_{\gamma,h,w,v} > 0 \land 1 \leq w \leq W \land 1 \leq v \leq V \}$
- $C_3\gamma = \{ \alpha' f_{h,w}/a_{\gamma,h,w} : h \in H_w \land a_{\gamma,h,w} > 0 \land W + 1 \leq w \leq W + V \}$
- $C_4\gamma = \{ \alpha'' f_{\mu} : \mu \in \Omega(\gamma) \}$
- $\hat{C}\gamma = \{ C_1\gamma \cup C_2\gamma \cup C_3\gamma \cup C_4\gamma \}$

An upper bound on the optimal value of model (ICP) is given by Equation (17).

$$\left\lceil \sum_{c=1}^{C} t_c \cdot \left( \sum_{k=1}^{q_c} l(c,k) \cdot d(c,k) \right) / \sum_{m=1}^{M} \hat{L}_m \right\rceil + \sum_{\gamma \in \{1, \ldots, \Gamma\}} \left\lceil \left( \sum_{c=1}^{C} D_c \cdot \left( \sum_{k=1}^{q_c} l(c,k) \cdot d(c,k) \right) \right) / \hat{L}_\gamma \right\rceil \cdot \min\{\hat{C}\gamma\} \right\} \quad (17)$$

The first part of Equation (17) corresponds to a lower bound for the makespan, while the second part stands for the minimum waste resulting from using molds of some fixed length $\hat{L}_\gamma$. 
4 Patterns generation

Instead of carrying out exhaustive enumerations, we generated the desirable packing, cutting, and overlapping patterns for a given instance using constraint programming models, which are described in the remainder of this section.

4.1 Packing patterns generation

Consider the following notation, in addition to the notation presented in Section 3:

- \( K \): the largest number of different lengths among beam types, i.e. \( \max_{c=1,...,C} q_c \) with \( c = 1, \ldots, C \). For example, in an instance with 2 beam types, in which type 1 has 6 distinct beam lengths and type 2 has 4 distinct beam lengths, we have \( K = 6 \).

- \( v_i \in \{1, \ldots, C\} \): a decision variable that corresponds to the type of beam used by the pattern \( P_i \).

- \( \gamma_i \in \{1, \ldots, \Gamma\} \): auxiliary decision variable for generating patterns that will be maximal in at least one mold of the problem. It defines in which mold capacity the generated pattern \( P_i \) is maximal.

- \( A^i \in \mathbb{Z}^K \): a vector of decision variables, with \( A^i_j = 0 \), if \( v_i = c \), \( c = 1, \ldots, C \), \( j = q_c + 1, \ldots, K \).

- \( P_i = (v_i, (A^i_1, \ldots, A^i_K)) \): the generated pattern.

For the generation of a packing pattern \( P_i \) we propose the model, which is adapted from (Araujo et al., 2019).

\[
1 \leq v_i \leq C, \quad (18)
\]

\[
1 \leq \gamma_i \leq \Gamma, \quad (19)
\]

\[
A^i_j = 0, \text{ if } v_i = c, \quad c = 1, \ldots, C, \quad j = q_c + 1, \ldots, K \quad (20)
\]

\[
L_m - \min_{j=1,...,q_c} \left( l(c, j) \right) < \sum_{j=1}^{q_c} l(c, j) \cdot A^i_j \leq L_m, \text{ if } (v_i = c \land \gamma_i = m), \quad c = 1, \ldots, C, \quad m = 1, \ldots, \Gamma, \quad (21)
\]

\[
A^i_k \in \mathbb{Z}_+, \quad k = 1, \ldots, K. \quad (22)
\]

Constraint (18) implies that the pattern type has domain \( \in \{1, \ldots, C\} \). Constraint (19) defines the length of the molds in which the generated pattern should be maximal. Constraint set (20) implies that if the generated pattern is of type \( v \) then it includes no beam of size \( l(v, j) \), such that \( j > q_v \). Constraint set (21) imposes that the capacity used by the generated pattern is simultaneously larger...
than the mold length minus the shortest beam length from its type and no larger than the length of the actual mold. The empty pattern is, therefore, not generated and has to be manually included in the final set of patterns. We utilized the solver CPLEX CP Optimizer to enumerate all the solutions of model (18)-(22).

4.2 Cutting patterns generation

In this section we propose a constraint programming model for cutting patterns generation. The decision variables are given below:

- \( w_h \): index of the bar that will be cut in the generated cutting pattern \( I_h \);
- \( A^h_i \): number of items of length \( L_i \) cut in the pattern, for \( i \in \{ 1, \ldots, \Gamma \} \);
- \( A^h_{\Gamma+1} \): number of items of length \( b_{W+1} \) cut in the pattern, for \( i \in \{ \Gamma+1, \ldots, \Gamma+V \} \);
- \( I_h = (w_h, (A^h_1, \ldots, A^h_\Gamma, A^h_{\Gamma+1}, \ldots, A^h_{\Gamma+V})) \): the generated pattern.

The proposed constraint model for generating a cutting pattern \( H_h \) is given by Equations (23)-(27).

\[
1 \leq w_h \leq W + V, \quad (23)
\]
\[
\sum_{i=1}^\Gamma L_i \cdot A^h_i + \sum_{i=1}^V b_{W+i} \cdot A^h_{\Gamma+i} \leq \text{element}(w_h, b), \quad (24)
\]
\[
\# \{ i \in \{ \Gamma+1, \ldots, \Gamma+V \} | A^h_i > 0 \} = 1, \quad (25)
\]
\[
A^h_i = 0, \text{ if } w_h > W, \quad A^h_i \in \mathbb{Z}_+, \text{ if } i = 1, \ldots, \Gamma+V. \quad (26)
\]

Constraint (23) defines the domain of each decision variables \( w_h \). Each \( w_h \) variable determines defines the bar that will be cut in the current pattern to generate items. If \( 1 \leq w \leq W \), the bar that will be cut is a new bar. If \( W + 1 \leq w \leq W + V \), the bar that will be cut is a bar leftover. Constraint (24) states that the total length of items cut in the pattern must be shorter than the length of the bar used to cut such pattern, with expression \( \text{element}(w_h, b) \) standing for the \( w_h \)-th element of array \( b \) [Beldiceanu and Carlsson, 2018]. Constraint set (25) implies that a cutting pattern only generates one type of leftover. Constraint (26) implies that a leftover does not generate more leftovers. We utilized the CPLEX CP Optimizer to enumerate all the solutions of model (23)-(27).

4.3 Overlapping patterns

In order to enrich the problem by allowing the possibility of using overlapping bars, we recall that an overlapping pattern \( O_\mu \) is a tuple \( O_\mu = (\gamma_\mu, (a^u_1, \ldots, a^u_{\gamma_\mu})) \). Note that \( \gamma \) is associated to the length of the bar that is generated in such pattern. Such length must be equal to the capacity of some mold, since we are only required to produce bars via overlapping that are used for beam production. A bar produced by overlapping is only produced from leftovers in stock.
In order to simplify the model’s notation, consider the following decision variables:

- \( A^\mu_i \): decision variable that represents number of items \( b_{W+i} \) used in the overlapping pattern, for \( i \in \{1, \ldots, V\} \).
- \( \gamma^\mu \in \{1, \ldots, \Gamma\} \): decision variable that defines the length of the bar produced by the overlapping pattern.
- \( f \geq 0 \): decision variable that expresses the waste of bar associated to the overlapping pattern to produce a bar of length \( L_{\gamma^\mu} \).
- \( \Omega^\mu = (\gamma^\mu, (A^\mu_1, \ldots, A^\mu_V)) \): the generated pattern.

The following constraint programming model can be used to produce an overlapping pattern:

\[
\begin{align*}
1 & \leq \gamma^\mu \leq \Gamma, \\
\sum_{i=1}^{V} A^\mu_i b_{W+i} & \geq L_{\gamma^\mu} + \epsilon, \\
\sum_{i=1}^{V} A^\mu_i & = 2, \\
f & = L_{\gamma^\mu} - \sum_{i=1}^{V} A^\mu_i b_{W+i}.
\end{align*}
\]

Constraint (28) ensures that the length of the bar produced is one of the possible mold lengths. Constraint (29) forces that the total length of the chosen leftovers is greater than the length of the bar produced via overlapping plus a constant \( \epsilon \) which is the loss of the bar resulting from the overlapping process. Constraint (30) defines that only 2 leftovers are used in the production of the bar made via overlapping. Constraint (31) defines the bar waste resulting from the overlapping pattern.

The constraint programming model for overlapping pattern generation is sufficiently flexible to accommodate the production planner’s necessities. In a more general setting, we could require that a bar made via overlapping can only be produced by using more than 2 and no more than a predefined number of leftovers and specify the \( \epsilon \) value to be proportional to the number of leftovers used in such pattern.

5 Genetic algorithm for the ICP-HPBMPP

In this section we propose a genetic algorithm to solve the ICP-HPBMPP, formalize the solution representation chosen, the solution fixing procedure, the selection, mutation, and crossover operators, as well as the initial population generation, population restart, and local search.
5.1 Solution representation

The solution representation consists of a 2-row matrix, in which each column \( j \) consists of the genes \( a_j \) and \( x_j \), where \( a_j \) is a pattern index and \( x_j \) is the number of times the pattern represented by \( a_j \) is used. The number of columns of this representation is variable and can be at most \( r + H + O \). The \( a_j \) genes can have values in \( \{1, \ldots, r + H + O\} \), in which the values \( 1, \ldots, r \) represent the packing patterns indices, the values \( r + 1, \ldots, r + H \) correspond to the cutting patterns indices, and the values \( r + H + 1, \ldots, r + H + O \) are associated with the indices of overlapping patterns. In Figure 2, we show a generic scheme of the solution representation, in which the number of columns is exactly \( r + H + O \).

Fig. 2 Solution representation

\[
\begin{array}{cccc}
   a_1 & a_2 & x_1 & \ldots & x_{\text{num columns}} \\
   x_1 & x_2 & \ldots & x_{\text{num columns}} \\
\end{array}
\]

In order to illustrate the solution representation we first present instance cwp000, generated randomly, in Table 1. Its respective packing, cutting, and overlapping patterns are presented in Tables 2, 3, and 4, respectively.

Table 1 Instance cwp000 description

| Instance cwp000 |  |
|----------------|---|
| \( C = 1 \)   |  |
| \( M = 5 \)   |  |
| \( T = 3 \)   |  |
| \( W = 1 \)   |  |
| \( V = 4 \)   |  |
| \( L = (5.95, 5.95, 5.95, 5.95, 11.95) \) |  |
| \( t_1 = 1 \) |  |
| \( q_1 = 2 \) |  |
| \( D_1 = 1 \) |  |
| \( l(1, \cdot) = (1.12, 3.3) \) |  |
| \( d(1, \cdot) = (5, 10) \) |  |
| \( b = (12, 2, 5, 6, 8) \) |  |
| \( e = (30, 16, 28, 25, 29) \) |  |
| \( \epsilon = 0.3 \) |  |

Table 2 Packing patterns for instance cwp000

| ID | Beam type | Capacity | \( a_1^p \) | \( a_2^p \) |
|----|-----------|---------|------------|------------|
| 1  | 1         | 5.6     | 5          | 0          |
| 2  | 1         | 5.54    | 2          | 1          |
| 3  | 1         | 11.2    | 10         | 0          |
| 4  | 1         | 11.14   | 7          | 1          |
| 5  | 1         | 11.08   | 4          | 2          |
| 6  | 1         | 11.02   | 1          | 3          |
Table 3  Cutting patterns for instance cwp000

| ID | Bar cut | Capacity | $a_1^h$ | $a_2^h$ | $a_3^h$ | $a_4^h$ | $a_5^h$ | $a_6^h$ |
|----|---------|----------|--------|--------|--------|--------|--------|--------|
| 7  | 1       | 5.95     | 1      | 0      | 0      | 0      | 0      | 0      |
| 8  | 1       | 7.95     | 1      | 0      | 1      | 0      | 0      | 0      |
| 9  | 1       | 9.95     | 1      | 0      | 0      | 2      | 0      | 0      |
| 10 | 1       | 11.95    | 1      | 0      | 0      | 3      | 0      | 0      |
| 11 | 4       | 5.95     | 1      | 0      | 0      | 0      | 0      | 0      |
| 12 | 5       | 5.95     | 1      | 0      | 0      | 0      | 0      | 0      |
| 13 | 1       | 11.95    | 1      | 0      | 0      | 0      | 1      | 0      |
| 14 | 1       | 10.95    | 1      | 0      | 0      | 1      | 0      | 0      |
| 15 | 1       | 11.9     | 2      | 0      | 0      | 0      | 0      | 0      |
| 16 | 1       | 11.95    | 0      | 1      | 0      | 0      | 0      | 0      |

Table 4  Overlapping patterns for instance cwp000

| ID | Bar generated | Waste of bar | $a_1^c$ | $a_2^c$ | $a_3^c$ | $a_4^c$ |
|----|---------------|--------------|--------|--------|--------|--------|
| 17 | 1             | 1.05         | 1      | 1      | 0      | 0      |
| 18 | 1             | 4.05         | 0      | 2      | 0      | 0      |
| 19 | 1             | 2.05         | 1      | 0      | 1      | 0      |
| 20 | 1             | 6.05         | 0      | 0      | 2      | 0      |
| 21 | 1             | 5.05         | 0      | 1      | 1      | 0      |
| 22 | 1             | 8.05         | 0      | 0      | 1      | 1      |
| 23 | 1             | 4.05         | 1      | 0      | 0      | 1      |
| 24 | 1             | 7.05         | 0      | 1      | 0      | 1      |
| 25 | 1             | 10.05        | 0      | 0      | 0      | 2      |
| 26 | 2             | 4.05         | 0      | 0      | 0      | 2      |
| 27 | 2             | 1.05         | 0      | 1      | 0      | 1      |
| 28 | 2             | 2.05         | 0      | 0      | 1      | 1      |

Note that ID is associated with the pattern indices. An optimal solution for the cwp000 instance is shown as the chromosome in Figure 3.

Fig. 3  Example of a feasible solution of instance cwp000

```
  6  16  2  11  15
  2  2  4  2  1
```

For the solution in Figure 3, we obtain an objective function value of 2.1, with makespan of 2 periods and bar waste of 0.1m. Figure 4 shows that packing patterns with indices 2 and 6, were used 4 and 2 times, respectively. Due to the fact that we are restricted to using only maximal packing patterns in their respective molds and a given packing pattern is maximal with respect to only one distinct length of mold, we infer that packing pattern 2 is associated with molds of length 5.95m, and packing pattern 6 is associated with molds of length 11.95m. Therefore, we need to produce a total number of 2 bars of length 5.95m and 6 bars of length 11.95, since the beam type produced by each solution packing patterns requires
only one bar. The cutting patterns used are those with indices 11, 15 and 16, and their frequencies are 2, 1, and 2, respectively. None of the overlapping patterns was selected in the solution.

The production planning consists of the specification of the exact quantity of bars required for the beam production as long as the available stock of bars is not violated. Thus, the solution represented encoded in the chromosome in Figure 3 is feasible.

![Gantt chart for an optimal solution of instance cwp000](image)

**Fig. 4** Gantt chart for an optimal solution of instance cwp000

5.2 Initial population generation

Since we typically need a large quantity of individuals to generate a population, deterministic methods are not the best choice, despite the high-quality solutions
produced by them. We propose a pseudorandom approach to generate a large quantity of solutions, which is described in Algorithm 1.

**Algorithm 1: Generate pseudo-random solution**

```plaintext
input: Instance, Set of Packing Patterns, Set of Cutting Patterns, Set of Overlapping Patterns
output: Feasible solution
1 Initialize solution with all patterns with their respective frequencies set to zero.
2 while Beam demands is not fulfilled do
   3 pac_p ← random packing pattern that has not yet been selected.
   4 if There is some beam in pac_p whose demand is unfulfilled then
      5 Increment the number of times that pac_p is used in solution until all beams in pac_p have their demands fulfilled.
   6 end
3 end
4 Calculate the number of bars needed according to the packing patterns frequencies
5 for each mold length γ do
   6 while (number of bars of length L_γ needed was not reached) ∨ (there is at least one cutting pattern not selected) do
      7 cut_p ← random cutting pattern that generates bars of length L_γ that has yet not been selected.
      8 bars_needed ← number of bars of length L_γ required.
      9 n ← number of times cut_p can be added to solution without violating bars stock.
      10 Increment cut_p frequency in solution by max(bars_needed, n) times.
   11 end
6 while number of bars of length L_γ needed was not reached do
5 7 ove_p ← random overlapping pattern that generates a bar of length L_γ that has not yet been selected.
   8 bars_needed ← number of bars of length L_γ required.
   9 n ← number of times ove_p can be added to solution without violating bars stock.
   10 Increment ove_p frequency in solution by max(bars_needed, n) times.
7 end
8 Remove from solution the genes associated to patterns that are not used
9 return solution
```

We call this method pseudorandom because we choose the patterns to add to the solution randomly, although each pattern frequency in the solution is computed in such a way as to respect stock and satisfy the demand. The time complexity of the Algorithm 1 is $O(P_{q_e} + \Gamma(H + O))$. Generating the initial population consists of creating of a number of individuals with the use of Algorithm 1 and selecting the best of them based on their fitness value according to the required population size.
5.3 Fitness function and selection operator

We use the objective function from the mathematical model (ICP) as the fitness function to evaluate the solution quality of a given chromosome. The selection operator consists of the process of selecting the best distinct solutions with respect to their respective fitness function value, i.e., the individuals with the lowest fitness values.

5.4 Crossover operators

In this subsection we propose two alternatives to use as crossover operators: crossover type 1, and crossover type 2. Given two parents, both crossover types generate one offspring, which consists of a new solution (chromosome).

In crossover type 1, we preserve all pattern indices from both parents, but the number of times each pattern is used in the offspring corresponds to the mean of the number of times they are used by the parents rounded to the largest integer. For each gene there is a probability of mutation. When the mutation occurs the number of times that the current pattern is used in such gene is set to zero. After this crossover process, if the generated offspring results in an infeasible solution, an iterative procedure, shown in Algorithm 6, is applied for its correction. If some pattern from the current offspring is used zero times, the gene associated to it is removed from the chromosome.

In crossover type 2, we first initiate the offspring using all patterns that used in both parents with their respective frequencies set to zero. For the genes that have patterns that are part of both parents simultaneously, their respective frequencies are set as the mean of their frequencies in the parents rounded to the largest integer. For each remaining gene we have a probability of 50% of setting its respective frequency to be equal to the originating parent frequency or keeping it equal to zero. If the resulting offspring is not feasible, the fixing procedure, shown in Algorithm 6, is applied to it and all patterns with final frequencies equal to zero have their respective genes removed from the chromosome.

5.5 Mutation operator

The mutation of an individual consists of choosing one pattern \( p_1 \) that is in the solution, and in the addition of one pattern \( p_2 \), chosen randomly, that is not part of the solution. The number of times that \( p_2 \) is used becomes the number of times that \( p_1 \) is used, and the number of times that \( p_1 \) is used is set to zero. If the solution is infeasible after this procedure we apply the fixing phase to it. This process is frequently required in practice and is described in this next subsection.

5.6 Infeasible solution fixing

Since that the proposed genetic operators of crossover and mutation can affect the feasibility of solutions, we must define a procedure to fix infeasible solutions to turn them into feasible ones before.

A chromosome may be an infeasible solution due to different reasons, as follows:
1. Infeasibility type 1, due to beam demand: the frequencies of packing patterns in the solution are not enough to fulfill the beam demands;
2. Infeasibility type 2, due to bar stock: the number of bars which are used in cutting and overlapping patterns exceed the bar stock;
3. Infeasibility type 3, due to inconsistent number of bars produced and required: the number of necessary bars generated by cutting and overlapping patterns is different from the number of bars that beam production requires.

If we detect any of those kinds of infeasibility, we must apply the infeasible solution fixing phase, which consists of Algorithm 6. Each infeasibility type is
treated in a particular procedure: Algorithms 3, 4, and 5 are used to fix infeasibility type 1, 2, and 3, respectively.

Algorithm 2: Remove unnecessary packing patterns

input: Infeasible chromosome
output: Potentially modified chromosome
1. Initialize produced beams with zeros;
2. demand_fulfilled ← false;
3. for each packing pattern $P_i$ in Chromosome do
   4. if demand_fulfilled = false then
      5. for cont = 1, ..., frequency($P_i$) do
         6. Update produced beams;
         7. if produced beams fulfill the beam demands then
            8. demand_fulfilled ← true;
            9. frequency($P_i$) ← cont;
            10. break;
      11. end
   12. else
      13. frequency($P_i$) ← 0
   14. end
   15. end
   16. return Chromosome

Algorithm 3: Fix chromosome with respect to infeasibility 1

input: Infeasible chromosome
output: Potentially feasible chromosome
1. while Infeasibility type 1 = true do
2. for each beam type $c$ do
3.   for each beam length $l_c$ whose demand is not fulfilled do
4.     for each packing pattern $P_i$ with type $c$ in Chromosome do
5.       if frequency of $l_c$ in $P_i$ ≠ 0 then
6.         Increment frequency($P_i$) until the demand of $l_c$ is achieved;
7.         break;
8.       end
9.   end
10. end
11. end
12. return Chromosome

Algorithm 4: Fix chromosome with respect to infeasibility 2

input: Infeasible chromosome
output: Potentially feasible chromosome
1. Calculate the #bars used;
2. for each standard bar or bar leftover $w$ do
3.   if #bars $w$ used ≠ stock of $w$ bars then
4.     for each cutting pattern $I_h$ that uses $w$ in Chromosome do
5.       $rt ← \#bars w$ used - stock of $w$ bars;
6.       frequency($I_h$) ← frequency($I_h$) - $rt$;
7.       Update the #bars $w$ used;
8.      if #bars $w$ used ≠ stock of $w$ bars then
9.        break;
10.     end
11. end
12. for each overlapping pattern $O_{\mu}$ that uses $w$ in Chromosome do
13.   $rt ← \#bars w$ used - stock of $w$ bars;
14.   $rt ← \frac{rt}{\#bars w$ in $O_{\mu}}$
15.   frequency($O_{\mu}$) ← frequency($O_{\mu}$) - $rt$;
16.   Update the #bars $w$ used;
17. if #bars $w$ used ≠ stock of $w$ bars then
18.      break;
19. end
The unnecessary packing patterns procedure, shown in Algorithm 2 in Appendix A, works like a solution treatment phase, which is not a necessary part of the solution fixing process, although applying such procedure we may improve solution quality and simplify the fixing process, i.e., it would be less likely that the modified solutions could not be fixed. The procedure consists of decreasing the frequency of packing patterns after the beam demands are already fulfilled if there are beam surplus.

In Figure 5 we show an example of the crossover operators, with offspring 1 as the solution generated by crossover operator type 1, and offspring 2 as the solution created by crossover operator type 2. Note that the fixing procedure was applied for offspring 2 and not for offspring 1. In Figure 6 we show an example of the proposed mutation operator. The resulting chromosome is infeasible, therefore, the solution fixing procedure must be applied. If the application of the solution fixing procedure to a given chromosome could not turn it into a feasible solution, the chromosome is discarded.

**Fig. 5** Crossover operators

```
Parent 1
1 2 3 4 5
6 7 8 9

Offspring 1
1 2 5
6 7

Parent 2
5 6
3

Offspring 2
2
3
```

**Fig. 6** Mutation operator and solution correction

```
2
10
10

Mutator

2
10
6
1

Fixing

2
5
```

### 5.7 Population restart

The population restart consists of the creation of a new population to compose the next generation after a predefined number of epochs. We apply a population restart after a given number of generations with no improvement of the best-fitness value. We divide such procedure into three parts, as follows: 1. selecting a certain number of the best-fitness individuals from the current population; 2. generating a number new pseudo-random individuals; 3. creating a new population with individuals from steps 1 and 2 and applying the selection operator to form the next population.
5.8 Local search

In order to improve the quality of final solutions, we apply a local search to every individual of the final population. For the local search we use the *insert* movement, which consists of, given two genes indices \( i \) and \( k \), with \( i < k \), inserting the gene \( i \) one position in front of \( k \)-th gene, i.e., all the genes between positions \( i \) and \( k + 1 \) are moved one position to the right after the insertion of the \( k \)-th gene. In Figure 7 an insert movement neighbor is shown for a given solution after inserting 2nd gene in front of 5th gene.

Fig. 7 Insert movement

\[
\begin{array}{cccccc}
6 & 1 & 12 & 16 & 5 \\
2 & 1 & 1 & 5 & 3
\end{array}
\]

Solution chromosome

\[
\begin{array}{cccccc}
6 & 12 & 16 & 5 & 1 \\
2 & 1 & 3 & 3 & 1
\end{array}
\]

Insert movement neighbor

Considering the function \( \text{INSERT}(\text{solution}, i, k) \) as the movement of insertion given indices \( i \) and \( k \), we describe the local search procedure in the Algorithm 7.

**Algorithm 7**: Insert neighborhood  

```plaintext
input: InitialSolution 
output: BestSolution 
1 BestSolution ← InitialSolution; 
2 for \( i = 1, \ldots, nℓ - 1 \) do 
3     for \( k = k + 1, \ldots, nℓ \) do 
4         neighbor ← \text{INSERT}(\text{InitialSolution}, i, k); 
5         if \( \text{makespan}(\text{neighbor}) < \text{makespan}(\text{BestSolution}) \) then 
6             BestSolution ← neighbor; 
7         end 
8     end 
9 end 
10 return BestSolution;
```

5.9 Algorithm description

In order to describe the proposed genetic algorithm we define the following parameters: population size (TP), number of generations (NG), crossover type (CRS), number of pseudo-random solutions generated for the initial population and restart selections (AS), mutation probability (MUT), number of generations with no fitness improvement to apply population restart (RST), and the number of individuals from the current population selected to be used in restart operator procedure (TER).
The proposed genetic algorithm can be seen as a steady-state model since only one new individual is generated per generation, even though we generate several individuals in the formation of the initial population and in a population restart process. A simplified scheme of the proposed genetic algorithm is shown in the flowchart in Figure 8.

Fig. 8  Simplified flowchart of proposed genetic algorithm

6 Computational experiments

In this section we present computational experiments on a set of benchmark instances that were produced with the intent to mimic real-world scenarios, to evaluate the solution methods proposed in this study.

The patterns corresponding to each test instance were generated using the constraint programming solver IBM ILOG CPLEX 12.8 CP Optimizer. For the integer programming model implementation we adopted the solver IBM ILOG CPLEX 12.8. Both solvers were used with Concert technology using the C++ programming language. The genetic algorithms were also developed with the C++ programming language.

We carried out every test in this paper on a Linux Ubuntu 18.04 64bits machine with 8GB of memory and Intel Core i5-3470 CPU 3.20 GHz ×4 processor. We compiled the created codes with the GNU GCC 7.3.0 compiler using Code::Blocks 17.12 IDE. Note that, for different values of \( \lambda \) we can form the Pareto front and may have different behaviors of the proposed model and algorithms. However, for the purpose of the study, we did not approach the multi-objective nature of the problem and considered, for each test described in this section, \( \lambda_i = 1 \), with \( i = 1, \ldots, 4 \).

6.1 Test instances generation

In this subsection, we describe how we generate the set of benchmark instances used in this section. We introduce a set of instances that are based on data arising
from a possible real-world scenario. The different instances represent a sample of the variability of the problem’s parameters, such as number of beam types, number of molds, and mold lengths.

In Table 5 we present details about each test instance parameter. We can see that the number of packing patterns increases as the number of beam types increases. However, the number of cutting and overlapping patterns remains constant because of the fact that we expect that the possible distinct bar lengths are standardized in real-world scenarios and therefore do not lead to variability.

| Instance | C | M | T | r | H | O |
|----------|---|---|---|---|---|---|
| cwp001   | 1 | 15| 6 | 145| 10| 12|
| cwp002   | 1 | 15| 6 | 199| 10| 12|
| cwp003   | 1 | 15| 6 | 236| 10| 12|
| cwp004   | 1 | 15| 6 | 210| 10| 12|
| cwp005   | 1 | 15| 6 | 236| 10| 12|
| cwp006   | 1 | 30| 3 | 257| 10| 12|
| cwp007   | 1 | 30| 3 | 257| 10| 12|
| cwp008   | 1 | 30| 3 | 199| 10| 12|
| cwp009   | 1 | 30| 3 | 218| 10| 12|
| cwp010   | 1 | 30| 3 | 199| 10| 12|
| cwp011   | 2 | 15| 15| 414| 10| 12|
| cwp012   | 2 | 15| 21| 395| 10| 12|
| cwp013   | 2 | 15| 21| 361| 10| 12|
| cwp014   | 2 | 15| 14| 387| 10| 12|
| cwp015   | 2 | 15| 17| 451| 10| 12|
| cwp016   | 2 | 30| 8 | 466| 10| 12|
| cwp017   | 2 | 30| 8 | 352| 10| 12|
| cwp018   | 2 | 30| 9 | 459| 10| 12|
| cwp019   | 2 | 30| 8 | 500| 10| 12|
| cwp020   | 2 | 30| 9 | 466| 10| 12|
| cwp021   | 3 | 15| 29| 662| 10| 12|
| cwp022   | 3 | 15| 36| 643| 10| 12|
| cwp023   | 3 | 15| 30| 614| 10| 12|
| cwp024   | 3 | 15| 29| 671| 10| 12|
| cwp025   | 3 | 15| 35| 684| 10| 12|
| cwp026   | 3 | 15| 30| 589| 10| 12|
| cwp027   | 3 | 15| 18| 560| 10| 12|
| cwp028   | 3 | 15| 18| 433| 10| 12|
| cwp029   | 3 | 30| 17| 620| 10| 12|
| cwp030   | 3 | 30| 20| 557| 10| 12|
| cwp031   | 4 | 15| 45| 952| 10| 12|
| cwp032   | 4 | 15| 50| 650| 10| 12|
| cwp033   | 4 | 15| 45| 896| 10| 12|
| cwp034   | 4 | 15| 41| 839| 10| 12|
| cwp035   | 4 | 15| 41| 783| 10| 12|

We consider mold capacities of 5.95m and 11.95m, while we take 1.12m, 1.45m, 2.35m, 2.5m, 2.65m, 2.95m, and 3.3m as possible beam lengths. For each instance, the possible curing times may be 1, 2, or 3 periods, chosen randomly when instances have more than 3 types. In addition, if the instance has up to 3 beam types, we associate the curing time to the beam type index, for example the beam type 2 needs a curing time of 2 periods. With respect to the number of bars that some beam type demands, we choose randomly a value between 1 and 3 for each beam
type. We choose the beam demands uniformly between 17 and 50. For total time horizon $T$, we calculate it as the ceiling of 150% of the optimal makespan lower bound, defined by Equation (32) as follows:

$$T = \left\lceil 1.5 \cdot \sum_{i=1}^{C} t_i \cdot \left( \sum_{k=1}^{q} l(c, k) \cdot d(c, k) \right) / \sum_{m=1}^{M} L_m \right\rceil. \quad (32)$$

For all instances, we consider an unique length of new bars as 12m and the possible lengths of bar leftovers as 2m, 5m, 6m, and 8m. We do not vary such lengths along the test instances since, in practice, it is expected that they are standardized. To generate realistic bar stocks we introduce an upper bound for the number of bars needed to fulfill the beam demand as $UB$, defined in Equation (33):

$$UB = 2 \cdot T \cdot M \cdot \max_{D_c=1,\ldots,C} \{D_c\}. \quad (33)$$

We set the stock of new bars of length 12m equal to $UB$, whilst we choose the stock of each leftover randomly between $\lceil UB/5 \rceil$ and $UB$ following an uniform distribution. We implemented the instance generator using MATLAB programming language.

6.2 Computational experiments with the mathematical model

In this subsection we discuss the results of the computational tests with the benchmark instance set that we generated following the scheme described in Subsection 6.1. In Table 6 we show the results of the computational experiments for the model (ICP) and its linear relaxation. The solution time was limited to 3,600 seconds. As regards to notation in Table 6, we consider LB, IP, and LP standing for the optimal objective function value lower bound, best solution value by CPLEX for model (ICP), and its linear relaxation value, respectively. When we say gap we mean the relative percentage deviation between the best integer objective and the objective of the best node remaining in the CPLEX B&C tree, calculated like this:

$$gap = 100 \cdot |bestbound - bestinteger| / (1e-10 + |bestinteger|) \quad (0\% \text{ means a proven optimal solution).}$$

We denote by “B&C nodes” the number of nodes generated in the branch-and-cut tree in the solution process, and $t \text{ (s)}$ as the solution time in seconds.

We can see in Table 6 that the linear relaxation of all instances could be solved, with the average time of 53.21 seconds, and with 624.61 seconds being the longest time to get to the optimal solution. On the other hand, only 11 instances could be solved to optimality by the integer programming model (4 of them solved in the root node of the B&C tree). For 23 instances we could not even find a feasible solution, a situation that we denote by “-”. Moreover, we could not solve 36 instances to optimality within the time limit, although feasible solutions for them were found. We can infer from the computational test results that the larger the instance parameter values are, the larger the problem is and the most difficult it is to find solutions for it. With high values of the instance parameters, when solutions are found, the optimality gap tends to be worse, i.e. the solutions achieved within the time limit are even further from the optimal solution.
We compare the results of the integer linear model (ICP), its linear relaxation, and our lower bound, in Equation 17, for the optimal value of objective function in the chart in Figure 9.

In Figure 9, the lower bound proposed in this work for the optimal objective function value was greater than the linear relaxation for all test instances and highly close to the objective function values obtained by CPLEX.

6.3 Experimental design and computational experiments with the proposed genetic algorithm

In order to achieve a better parameterization for the robustness of the proposed genetic algorithm, we apply fractional factorial parameter design. [Gholami et al.]
2009] used Taguchi experimental design [PIGNATIELLO JR, 1988] to achieve improved robustness of the genetic algorithm which they proposed. In this method the optimal parameter choice is found with the analysis of different level combinations of the control factors in an orthogonal array, with no necessity of testing all of the possible level combinations. [7] displays the proposed levels for the genetic algorithm parameters (control factors) introduced in Section 5.

### Table 7  Factor levels

| Factors | Index of levels | Levels     |
|---------|-----------------|------------|
| TP      | 1               | 25         |
|         | 2               | 50         |
| NG      | 1               | 500·r      |
|         | 2               | 1000·r     |
| MUT     | 1               | 0.01       |
|         | 2               | 0.025      |
|         | 3               | 0.05       |
| RST     | 1               | [0.1 · NG] |
|         | 2               | [0.2 · NG] |
| AS      | 1               | 100·r      |
|         | 2               | 500·r      |
| CRS     | 1               | Type 1     |
|         | 2               | Type 2     |
| TER     | 1               | [0.1 · Tr] |
|         | 2               | [0.2 · Tr] |

We must have one degree of freedom for total mean, one degree of freedom for each factor with two levels, and two degrees of freedom for the factor with 3 levels amounting to a total of nine degrees of freedom (1 + 1 × 6 + 2 × 1 = 9). However, with the control factors and respective levels that we defined, there is no orthogonal array aside from the full factorial array. Thus, we are not able to use a classical Taguchi orthogonal array design. In such circumstances one alternative is to use the D-optimal design [de Aguiar et al., 1995], which are constructed to minimize the generalized variance of the estimated regression coefficients. Note that D-optimality is only one possible criterion to choose a particular design. We obtain the D-optimal design, by Fedorov algorithm [TRIFENBACH, 2008] using R programming language for 9 trials for the chosen factors and their respective levels, illustrated in Table 8.

### Table 8  D-optimal design with 9 trials

| Trial | TP | NG | MUT | RST | AS  | CRS | TER |
|-------|----|----|-----|-----|-----|-----|-----|
| 1     | 1  | 1  | 1   | 1   | 2   | 2   | 1   |
| 2     | 1  | 2  | 3   | 1   | 1   | 2   | 1   |
| 3     | 1  | 2  | 1   | 1   | 2   | 1   | 2   |
| 4     | 1  | 1  | 2   | 2   | 1   | 2   | 2   |
| 5     | 2  | 2  | 2   | 1   | 1   | 1   | 1   |
| 6     | 2  | 1  | 2   | 1   | 2   | 2   | 1   |
| 7     | 2  | 1  | 3   | 1   | 1   | 1   | 2   |
| 8     | 2  | 2  | 1   | 1   | 1   | 2   | 2   |
| 9     | 2  | 2  | 3   | 2   | 2   | 2   | 2   |
Furthermore, the effectiveness characteristic of the genetic algorithms proposed is the expected fitness value, which we seek to minimize, i.e., “the lower is better principle”. Thus, for increased robustness of the algorithm we use the S/N (signal-to-noise) ratio, defined as follows. Note that the larger the value of S/N ratio the better.

\[ \eta_i = -10 \ln \left( \frac{1}{N} \sum_{j=1}^{N} \text{FIT}_{ij}^2 \right) \]  \hspace{1cm} (34)

with \(i\) and \(j\) denoting index of trial and index of replication, while \(\text{FIT}_{ij}\) stands for the best objective function value obtained by running the GA. We denote by “trial” a certain combination of the control factor levels.

We define a replication as one GA run of some trial for a given instance, and \(N\) as the number of test instances multiplied by the number of replications. Since we have an instance set of size 70 and we run each instance 10 times, we perform 700 replications for each trial.

Since CPLEX could not find optimal or even a feasible solution for most instances, we are unable to use the relative percentage deviations from the optimal solution as a performance indicator for the GA. Thus, we use the lower bound relative percentage deviations (LBD) of the fitness values for such purpose. Given a trial \(i\) and a replication \(j\) the LBD value is defined as follows:

\[ LBD_{ij} = \frac{\text{FIT}_{ij} - \text{LB}_j}{\text{LB}_j} \]  \hspace{1cm} (35)

where \(\text{LB}_j\) stands for the lower bound of the optimal objective function value for the test instance used in replication \(j\). The \(LBD\) for a given trial \(i\), denoted by \(LBD_i\), is the average \(LBD\) for all replications of instance set, as we can see in the following equation:

\[ LBD_i = \frac{1}{N} \cdot \sum_{j=1}^{N} LBD_{ij} \]  \hspace{1cm} (36)

The remainder of the experimental design procedure consists of three phases:

1. Evaluate the impacts of the control factors on the S/N ratios and on the LBD values.
2. For each factor, which has significant impact on the S/N ratios values, we choose the level which increases the S/N ratios.
3. For the factors, which do not have significant impact on the S/N ratios and have significant impact the LBD values, we choose the level which better approximate the lower bound values.
4. For the factors, which have significant impact neither on the S/N ratios nor on the LBD values, we select the factor levels regarding the more economic manner, that is, we choose the levels which have less impact on the algorithm running time.

We can see in Table 6.3 the results after carrying out the computational tests for each trial with the test instance set.
Table 9  LBD, S/N ratio, and average execution time results for each trial

| Trial | Control factors | LBD values | S/N ratios | Average time (s) |
|-------|-----------------|------------|------------|-----------------|
| 1     | TP 1 NG 1 MUT 1 RST 2 AS 1 CRS 1 TER 1 | 0.23419 | -80.01779 | 274.7 |
| 2     | TP 1 NG 2 MUT 3 RST 1 AS 2 CRS 1 TER 1 | 0.28382 | -80.88757 | 149.5 |
| 3     | TP 1 NG 2 MUT 1 RST 2 AS 1 CRS 2 TER 1 | 0.21597 | -79.52707 | 589.7 |
| 4     | TP 1 NG 1 MUT 2 RST 1 AS 2 CRS 2 TER 1 | 0.33001 | -81.74368 | 69.0 |
| 5     | TP 2 NG 2 MUT 1 RST 1 AS 1 CRS 2 TER 1 | 0.20842 | -79.28486 | 245.7 |
| 6     | TP 2 NG 1 MUT 2 RST 1 AS 1 CRS 2 TER 1 | 0.31188 | -81.62437 | 482.4 |
| 7     | TP 2 NG 3 MUT 1 RST 1 AS 1 CRS 1 TER 1 | 0.20360 | -79.18351 | 180.1 |
| 8     | TP 2 NG 1 MUT 1 RST 1 AS 1 CRS 2 TER 1 | 0.32368 | -81.78923 | 255.1 |
| 9     | TP 2 NG 3 MUT 2 RST 1 AS 2 CRS 1 TER 1 | 0.28475 | -80.88283 | 344.0 |

In Figure 10, we can see the main effects plot for the control factors using S/N ratios as the response variable. In Figure 11, we show the boxplots for each factor also using S/N ratios as the response variable. The mean response is clearly influenced by the type of crossover, while it is not so clear to affirm whether or not the other factors influence the response variable.

Fig. 10  Main effects plot for S/N ratio for lowerbound deviation values
Adjusting the linear regression model for all seven factors and performing an ANOVA test, we observe that only CRS is statistically significant with $P$-value 0.0259. Then we remove, one by one, the factors whose $P$-value is the greatest and readjust the regression model until all factors are statistically significant obtaining the ANOVA results in Table 11.

**Table 10** ANOVA table for S/N ratios for linear regression model fit considering all 7 factors

| Factor | df | Sum Sq | Mean Sq | $F$ value | $P$-value |
|--------|----|--------|---------|-----------|-----------|
| TP     | 1  | 0.0002 | 0.0002  | 0.0125    | 0.9290    |
| NG     | 1  | 0.0640 | 0.0640  | 4.5126    | 0.2801    |
| MUT    | 1  | 0.3786 | 0.3786  | 26.6784   | 0.1218    |
| RST    | 1  | 0.0749 | 0.0749  | 5.2817    | 0.2613    |
| AS     | 1  | 0.0292 | 0.0292  | 1.3807    | 0.4489    |
| CRS    | 1  | 8.5564 | 8.5564  | 602.9946  | *         |
| TER    | 1  | 0.0196 | 0.0196  | 1.3807    | 0.4489    |
| Residuals | 1   | 0.0142 | 0.0142  |           |           |
| Total  | 8  | 9.1371 |         |           |           |

Signif. codes: * 0.05
Table 11 ANOVA table for S/N ratio for linear regression model fit considering most significant factors

| Factor | df | Sum Sq | Mean Sq | F value | P-value |
|--------|----|--------|---------|---------|---------|
| NG     | 1  | 0.0627 | 0.0627  | 5.2218  | 0.08431 |
| MUT    | 1  | 0.3671 | 0.3671  | 30.5578 | 0.00523 ** |
| RST    | 1  | 0.0794 | 0.0794  | 6.6076  | 0.06195 |
| CRS    | 1  | 8.5799 | 8.5799  | 714.2988 | 0.00001 *** |
| Residuals | 4 | 0.0480 | 0.0120  |         |         |
| Total  | 8  | 9.1371 |         |         |         |

Signif. codes: 0 *** 0.01 ** 0.05 * 0.1

The number of generations, mutation rate, restart, and type of crossover showed to be statistically significant, meaning that we chose the levels whose average S/N ratios are the greater. The parameter levels chosen as a result of the ANOVA test are 1000 generations, 0.05 of mutation rate, $\lceil 0.2r \rceil$ generations with no improvement to apply restart, and crossover type 1.

As regards to the LBD as response variable to the linear regression model. We observe in the main effects plot in Figure 12 and in boxplots in Figure 13 that LBD have a similar behavior on the control factors. However, we note that, in this case, the lower the LBD value the better.

Fig. 12 Main effects plot for lowerbound deviation

Fig. 13 Boxplots for LBD values with each factor
Adjusting the linear regression model for all the seven factors and performing an ANOVA test using the LDB as response variable, we conclude that only CRS is statistically significant with $P$-value 0.03219. Therefore, we remove from the regression model the variables, one by one, whose $P$-value is the greatest and readjust the model until all factors are statistically significant achieving the ANOVA results illustrated in Table 11.

Table 12  ANOVA table for LBD values for linear regression model fit considering all 7 factors

| Factors | df | Sum Sq | Mean Sq | $F$ value | $P$-value |
|---------|----|--------|---------|-----------|-----------|
| TP      | 1  | 0.00000| 0.00000 | 0.00340   | 0.96270   |
| NG      | 1  | 0.00012| 0.00012 | 2.39350   | 0.36531   |
| MUT     | 1  | 0.00058| 0.00058 | 11.48800  | 0.18265   |
| RST     | 1  | 0.00006| 0.00006 | 1.23330   | 0.46669   |
| AS      | 1  | 0.00021| 0.00021 | 4.22270   | 0.28833   |
| CRS     | 1  | 0.01960| 0.01960 | 390.49990 | 0.03219** |
| TER     | 1  | 0.00011| 0.00011 | 2.27660   | 0.37261   |
| Residuals | 1 | 0.00005| 0.00005 |            |           |
| Total   | 8  | 0.02074|         |           |           |

Signif. codes: * 0.05

Table 13  ANOVA table for LBD values for linear regression model fit considering most significant factors

| Factors | df | Sum Sq | Mean Sq | $F$ value | $P$-value |
|---------|----|--------|---------|-----------|-----------|
| MUT     | 1  | 0.00063| 0.00063 | 6.02770   | 0.03944** |
| CRS     | 1  | 0.01949| 0.01949 | 187.86960 | 0.00001***|
| Residuals | 6 | 0.00062| 0.00010 |            |           |
| Total   | 8  | 0.02074|         |           |           |

Signif. codes: *** 0.001 ** 0.01 * 0.05

Taking into consideration the LBD as response variable to the regression model, only the mutation rate, and type of crossover are statistically significant, meaning that we would choose the mutation rate 0.05, and crossover type 1. However, these variables were already fixed at the S/N ratios analysis, and no factors that were not statistically significant for the S/N ratios showed to be statistically significant with LBD values. This leads us to choose the levels that would spend less computational time, for the factors whose level was not selected yet. Therefore, the most robust parameterization of the levels for the proposed control factors is: population size 25, 1000 generations, 200 generations with no improvement to apply restart, 100 pseudo-random solutions generated in the initial population and restarts, crossover type 1, 5 preserved individuals upon restart, and mutation rate of 0.05.

6.4 Analysis of the final genetic algorithm parameterization

In order to observe the genetic algorithm behaviour, we run the GA with instance cwp021. Figure 14 illustrates the best fitness and mean fitness of the populations along all generations. The x-axis of the Figure 14 is on logarithmic scale. The
largest improvement takes place during the first generations of the GA, while in the last ones the best fitness is stagnant with some improvement upon the first restart.

**Fig. 14** Average and best objective function value curves for instance cwp021 along generations of the selected genetic algorithm parameterization

![Graph](image)

In Figure 14, the best fitness values obtained by running the GA were better than CPLEX in five instances, while solutions were obtained for all instances which CPLEX could not solve.

**Fig. 15** Lower bound relative deviations for CPLEX and GA with the selected parameterization

![Graph](image)

In Figures 16 and 17, the time spent by the GA on solving each instance was significantly better than the CPLEX solution time on the large and medium-sized instances. Thus, CPLEX was faster than the GA in the small-sized instances. The y-axis is in logarithmic scale.
Fig. 16 Mean time for each instance solved by CPLEX and GA with the selected parameterization.

![Graph showing mean time for each instance solved by CPLEX and GA with the selected parameterization.](image)

Fig. 17 Mean time for each instance solved by CPLEX and GA with the selected parameterization, with y-axis in logarithmic scale.

![Graph showing mean time for each instance solved by CPLEX and GA with the selected parameterization, with y-axis in logarithmic scale.](image)

7 Final remarks

In this work, we proposed a novel variant of cutting sequencing problems called the integrated cutting and packing heterogeneous precast beams multiperiod production planning (ICP-HPBMPP), which, to the best of the authors’ knowledge, has not yet been studied and may have a large impact on both real-world and theoretical studies. The ICP-HPBMPP consists in integrating the problem of production planning of precast beam with the problem of cutting the traction elements used in such production, while taking into consideration the generation of leftovers and bar generated via overlapping.
We argued that the problem is NP-hard and proposed an integer linear programming model for its solution, in addition to a lower bound on its optimal objective function value. We also showed that restricting the formulation to using exclusively maximal packing patterns does not change the optimal solution set of the problem.

We also proposed three constraint programming models for generating distinct types of beam production patterns. Additionally, we introduced a set of benchmark instances and carried out computational experiments in order to evaluate the relative performance of the different solution methods studied.

The experiments showed that the integer programming model can be used to solve small size instances, while it typically does not reach optimality while solving medium size instances. In addition, the model usually does not find feasible solutions for large size instances. We introduced a genetic algorithm for solving the problem and fine tuned its parameters by means of a D-optimal experimental design to achieve improved robustness of the algorithm. The final genetic algorithm is an attractive alternative to the integer programming model, resulting in high-quality solutions in shorter solution times as compared with the exact model.

There are numerous opportunities for future work regarding the ICP-HPBMP. In the domain of modeling, the problem can be modified to take into consideration distinct types of bars varying in matter of diameter or material, instead of only in matter of length. Also, dynamic demand could be considered, i.e., in each period a new demand of beams could be included, while not exceeding a prescribed stock of bars. Regarding solution approaches, multi-objective optimization algorithms can naturally be applied to the problem, since it involves preferences between makespan and bar waste. Decomposition approaches, such as column generation, or MIP heuristics, e.g., size-reduction heuristics, can also be interesting methods to be explored in conjunction with the proposed integer programming model.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Conflict of interest

The authors declare that they have no conflict of interest.

References

de Aguiar PF, Bourguignon B, Khots M, Massart D, Phan-Than-Luu R (1995) D-optimal designs. Chemometrics and intelligent laboratory systems 30(2):199–210

Araujo K, Bonates T, Prata B, Pitombeira-Neto A (2019) Heterogeneous prestressed precast beams multiperiod production planning problem: Modeling and solution methods. Preprint, https://arxiv.org/abs/1903.08609v1 arXiv:1903.08609

Arenales MN, Cherri AC, Nascimento DNd, Vianna A (2015) A new mathematical model for the cutting stock/leftover problem. Pesquisa Operacional 35(3):509–522
Beldiceanu N, Carlsson M (2018) Global constraint catalog. URL http://sofdem.github.io/gccat/gccat/Celement.html

De Castilho VC, El MK Debs, do M Carmo Nicoletti (2007) Using a modified genetic algorithm to minimize the production costs for slabs of precast prestressed concrete joists. Engineering applications of artificial intelligence 20(4):519–530

Dyckhoff H (1990) A typology of cutting and packing problems. European Journal of Operational Research 44(2):145–159

Gholami M, Zandieh M, Alem-Tabriz A (2009) Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns. The International Journal of Advanced Manufacturing Technology 42(1-2):189–201

Gilmore PC, Gomory RE (1961) A linear programming approach to the cutting-stock problem. Operations research 9(6):849–859

Gilmore PC, Gomory RE (1963) A linear programming approach to the cutting stock problem part ii. Operations research 11(6):863–888

Marler RT, Arora JS (2004) Survey of multi-objective optimization methods for engineering. Structural and multidisciplinary optimization 26(6):369–395

Melega GM, de Araujo SA, Jans R (2018) Classification and literature review of integrated lot-sizing and cutting stock problems. European Journal of Operational Research 271(1):1–19

PIGNATIELLO JR JJ (1988) An overview of the strategy and tactics of taguchi. IIE transactions 20(3):247–254

Poldi KC, Arenales MN (2010) O problema de corte de estoque unidimensional multiperíodo. Pesquisa Operacional 30(1):153–174

Prata BA, Pitombeira AR Neto, Sales CJM (2015) An integer linear programming model for the multiperiod production planning of precast concrete beams. Journal of Construction Engineering and Management 141(10):1–4

Stadtler H (1990) A one-dimensional cutting stock problem in the aluminium industry and its solution. European Journal of Operational Research 44(2):209–223

Triefenbach F (2008) Design of experiments: the d-optimal approach and its implementation as a computer algorithm. Bachelor’s Thesis in Information and Communication Technology

Trkman P, Gradisar M (2007) One-dimensional cutting stock optimization in consecutive time periods. European Journal of Operational Research 179(2):291–301

Vance PH (1998) Branch-and-price algorithms for the one-dimensional cutting stock problem. Computational Optimization and Applications 9(3):211–228

Vassoler AHD, Poltroniere SC, Araujo SA (2016) Modelagem matemática para o problema de produção de vigotas na indústria de lajes treliçadas. Revista Eletrônica Paulista de Matemática 7:68–77

Wang Z, Hu H, Gong J (2018) Framework for modeling operational uncertainty to optimize offsite production scheduling of precast components. Automation in Construction 86:69–80

Wässcher G, Haßner H, Schumann H (2007) An improved typology of cutting and packing problems. European Journal of Operational Research 183(3):1109–1130