An LP-based Tabu Search 
for Batch Scheduling in a Cutting Process 
with Finite Buffers

Claudio Arbib\textsuperscript{a}, Fabrizio Marinelli\textsuperscript{b}, Ferdinando Pezzella\textsuperscript{b}

\textsuperscript{a}Dipartimento di Informatica, Universit\`a dell’Aquila, Via Vetoio, Coppito, I-67010 L’Aquila, Italia; e-mail: claudio.arbib@univaq.it
\textsuperscript{b}Dipartimento di Ingegneria Informatica, Gestionale e Automazione, Universit\`a Politecnica delle Marche, Via Brecce Bianche, I-60131 Ancona, Italia; e-mail: fabrizio.marinelli@univpm.it, pezzella@diiga.univpm.it

\textbf{Abstract}

This paper addresses a cutting stock problem under typical resource constraints that arise when working centres with nesting capabilities are associated with automatic feeders/stackers. The critical resource is the number of buffers available to host the batches built up by the centre. To cope with it, pattern and batch sequencing problems must be addressed simultaneously. A tabu-search algorithm exploring batch output sequences is proposed. The algorithm never opens more stacks than buffers, respects batch compatibility/precedence constraints, and keeps the maximum order spread under control. To demonstrate its effectiveness and efficiency, a computational study was set up, solving 920 test problems derived from literature. The study enabled a proper tuning of the method and offered encouraging results: in 228 cases an optimum was found; in nearly all, the gap from the optimum was below 1%. Computation times range from fractions of seconds to a couple of minutes in the worst cases. Compared to existing methods, the algorithm provides on average the same solution quality, with the advantage of solving a problem which is more general and hence closer to application. The article includes a discussion on the method extensions required to deal with asynchronous stacking and heterogeneous batches.

\textit{Keywords:} cutting stock, open stacks, batch scheduling, heuristics

\textit{Mathematics Subject Classification:} 90B30 (Production models), 90C59 (Approximation methods and heuristics), 90C10 (Integer programming)
1. Introduction

In its very general form, the cutting stock problem (CSP) consists in obtaining given amounts of small objects \((parts)\) by cutting large objects \((stock-items)\) in a way that allows specific goals to be reached – for a survey see Cheng et al., 1994 and for a reasoned typology see Dyckhoff et al., 1997 and Wäschner et al., 2007. The way a single stock item is cut, is called a cutting pattern, implying that the same type of cut is generally repeated a number of times: this number is often called the pattern activation level or run length, and a solution of the CSP is identified by the run lengths of all the feasible patterns.

Unlike most literature, in this work we are not only concerned with the usual objective of trim-loss minimization, but also on how trim-loss quasi-equivalent CSP solutions differently impact on the production process. In fact, according to recent trends (SCOOP, 2008), cutting systems integrate workcentres — provided with nesting capabilities — and handling systems — able to manipulate pallets and panels and to stack the produced parts for further operation. In most cases (like the robot-assisted series or the integrated angular workcentres produced by world-renowned machine tool manufacturers, see e.g. (BISSSE Group, 2011)) regularity of downstream material flow can be ensured only through a careful operation planning of the associated workcentre. To do this, choosing patterns and run lengths is no longer sufficient, but it is also necessary to determine a cutting plan, that is, to choose and activate patterns in a sequence compliant with machine tool resources and capacity. In order to address the problem, let us briefly discuss some important issues that generally arise in a practical context, referring to Section 6 for a detailed discussion.

1.1. A general problem

In many practical contexts, e.g. the furniture industry, the organization of a cutting process obeys two basic rules:

\(R1\): each cutting pattern is applied once, up to its run length;

\(R2\): finished parts are stacked together into piles, and a pile is released when completed. We say that a new stack is open when a new pile starts building, and that an open stack is closed when a pile is consolidated.
R1 aims at reducing both time consuming machine set-ups, required for pattern changeover, and work-in-process: in fact, pattern pre-emption does not reduce either set-ups or maximum/mean order spread. It is here interesting to point out that R1 permits to describe the process as a sequence of patterns instead of a sequence of cuts.

R2 is adopted to reduce handling (and, when the case, the consequent risk of part breakage), as well as to facilitate part sorting and/or batch identification. Industrial production is very often organized in batches: a batch can be either homogenous — that is, formed by parts of the same size/type — or heterogeneous — for instance, when different part types are grouped to form a particular client order. According to work organisation, each particular stack either forms a specific batch required, and thus may contain parts of different types (batch stacking), or simply contains identical parts (part-type stacking).

Batch scheduling is often important to meet specific constraints and process efficiency.

Specific constraints. Depending on technical and organisational requirements, not all the batch schedules are feasible in general. An important technical issue is that automated cutting machines usually have a limited number of downstream buffers: hence, the number of stacks maintained open at any time during operation can never exceed that of downstream buffers (Belov and Scheithauer, 2007). Production organization may also require precedence constraints due, e.g., to downstream assembly phases. Compatibility constraints among open stacks can also be imposed, e.g., to manage batch similarity (Matsumoto et al., 2010).

Process efficiency. Relevant performance indicators for a batch schedule are, among others, batch due-dates violation (Reinertsen and Vossen, 2010), sum of completion times (Arbib et al., 2003), and maximum/mean order spread (Foerster and Wäscher, 1998).

However, the output sequence of batches depends on cut scheduling, that is — under R1 — on pattern sequencing. Thus, the problem of finding a cutting plan that takes into account the technical and production issues so far analyzed can, in very general terms, be phrased as follows:

Problem 1.1. Find a set of patterns, their run lengths and a pattern sequence such that:

- the demand of all part types is fulfilled;
• the output schedule of batches is feasible and meets the desired level of performance;
• the total trim-loss is minimized.

For this problem we here suggest an approach that partially enumerates the region of feasible sequences and computes, for each solution, an efficient pattern set that can produce that sequence. This is an alternative approach to that implicitly suggested by the literature on pattern sequencing (see §1.2), according to which the global problem is solved by choosing the best pattern sequence among those obtained from different solutions of the CSP. A major advantage of our approach is that the search can be guided by an evaluation function whose computation relies on a pattern-based CSP formulation, i.e., requires to solve a problem much easier than pattern sequencing.

In order to trade-off between inventory, holding and production costs, the cutting process can be integrated with mid-term planning level decisions concerning the inventory of semi-finite parts, the procurement and the assortment of stocks, see Holthaus, 2003, Arbib and Marinelli, 2005 and Gramani et al., 2009. An integration between mid-term and operational decision levels is however beyond the scope of this work.

1.2. Literature review and related problems

Literature on batch scheduling in the context of a cutting process offers various examples of studies focussed on open stacks. The problem calls, in general, for finding a pattern sequence that fulfills demand and reduces, as far as possible, both the trim-loss and the number of stacks maintained open throughout the process. Most recent research deal with problems with homogenous batches: one exception is the Ordered CSP, see Ragsdale and Zobel, 2004, that describes a case with heterogeneous batches. Solution approaches concentrate, with few exceptions, on the Open Stack Minimization Problem (Chu and Stuckey, 2009; Smith and Gent, 2005), that is, sequencing a given set of patterns so as to minimize the maximum or the total number of stacks maintained open during production. This requires a set of cutting patterns to be sequenced (normally, one reducing trim-loss to a minimum), and the two issues of cutting and sequencing are tackled as stand-alone problems to be sequentially solved. Such a sequential approach has however some drawbacks:
• first, the approach is not the same as minimizing open stacks under a prescribed trim-loss: identical (or similar) trim-losses may be obtained by many different pattern sets, and the quality of a pattern sequence (and the time needed to compute it) can strongly depend on the number and type of patterns chosen;

• moreover, even the best optimal solution among all the sequencing problems associated with minimum trim-loss cutting plans may require more open stacks than allowed by the cutting machine, and then turn out to be practically infeasible. This is not just a theoretical eventuality: Yanasse et al., 1999, showed problem instances with 150 batches having solutions with over 100 open stacks.

The literature records very few papers attempting at an integrated solution, and only a couple of papers propose an exact approach for small instances. In Yanasse and Lamosa, 2007, the problem is decomposed, via Lagrangian relaxation of a formulation with exponentially many variables and constraints, into a cutting stock and a pattern sequencing problem. These are repeatedly solved in a feedback scheme, the CSP by column generation, and pattern sequencing by an exact procedure. In both problems, pattern selection is guided by dual multipliers associated to the relevant constraints. A different formulation devised in Arbib et al., 2010, has a number of constraints that grows quadratically with the number of batches.

A heuristic algorithm can be found in Belov and Scheithauer, 2007: the CSP is solved by an effective sequential heuristic, i.e., a procedure that computes patterns one after the other, each time solving a knapsack problem where profits are so-called pseudo-prices obtained by looking to previously generated patterns. The number of open stacks is controlled in the solution of the knapsack problem by limiting the number of batches whose production is not yet started.

In Armbruster, 2002, a real application with additional technological constraints is described: an assortment of stock items of various lengths is available in a limited number of pieces, and each downstream buffer is lengthwise subdivided in a set of compartments. In order to be stacked, each part-type requires a set of consecutive compartments on the same buffer. The cutting stock problem is heuristically solved and then patterns are sequenced according to an assignment of part-types to compartments. If no sequence is found, the cutting stock solution is altered by imposing that two part-types, chosen by estimating trim-loss worsening, are separately produced.
In the case addressed by Matsumoto et al., 2010, homogenous batches are split into lots, and a lot is completed as soon as a stack is consolidated. Lengthwise similar part-types are forbidden in simultaneously open stacks, due to recognition reasons. Matsumoto et al. introduce the notion of cutting group, i.e., a subset of lots that can be cut without violating any scheduling constraint. Since all the stacks are closed at the end of each cutting group, the problem of defining a feasible cutting plan is recast to that of determining a sequence of cutting groups. A solution is computed by a local search that swaps lots between cutting groups.

2. Problem definition and basic solution approach

In the following we focus for simplicity on 1-dimensional cutting, although the proposed algorithm, which relies on a pattern-based CSP formulation, can be extended to other cases as soon as a suitable pricing algorithm is available (in fact the geometric dimension of parts, as well as other technical aspects such as guillotine cuts etc., affects the pricing algorithm only). We here assume that batches are homogenous, i.e., corresponds to part types, and there are no precedence/compatibility constraints. In particular, we do not address the problem of deciding the order in which parts are cut within a given pattern, also because this order is very much dependent on cut technology. However we will introduce, and later discuss (§2.1), a condition that allows to easily cope with some of these constraints. More general cases will be then discussed in Section 6.

Consider a problem where a finite set $M$ of $m$ part types, the $i$-th of width $w_i$ and required in $d_i$ parts, must be cut from a sufficiently large set of identical stock items of width $w$. A cutting plan $P$ consists of the repeated application of cutting patterns from a feasible set $K$ until all the required parts are obtained. Plan $P$, be it CSP-optimal or not, is implemented by applying its patterns in some order $\pi$, which will in turn output the part types in some (partial) order $\beta$. The latter is called the output sequence: $\beta(t) \in M$ denotes the $t$-th part type in the sequence, and $\beta^{-1}(i) = \alpha(i) \in \{1, \ldots, m\}$ denotes the position of the $i$-th part type in the sequence. We assume that the cutting machine has an $s$-slot outbuffer able to maintain up to $s$ distinct part types at a time, and that according to $R2$ a slot can be released only after the part type it hosts has been completed.

**Definition 2.1.** We say that $P$ is schedulable if it can be sequenced in an
order \( \pi \) so that, at any time, the number of distinct part types which are not completed (open stacks) never exceeds \( s \).

**Remark 2.1.** Clearly, any pattern of a schedulable cutting plan produces parts of \( \leq s \) distinct types, except (possibly) the last one, which can produce \( r > s \) types, provided that the pattern multiplicity is 1 and the exceeding part types form \( r - s \) full batches. This case can easily be handled separately, so we will w.l.o.g. assume that no batch can entirely be cut from a single stock item. \( \square \)

We can then reformulate Problem 1.1 as the following *Cutting Stock with Bounded Open Stack Problem* (in short, *BSP*).

**Problem 2.1.** Find a schedulable cutting plan \( P \) that produces all the required batches with a minimum trim-loss.

In the 1-dimensional case, an optimal solution of Problem 2.1 can be computed in polynomial time if every two individual parts fit in a stock item but no three do, see Aloisio et al., 2010. In this case the solution turns also out to be CSP-optimal. In general, however, Problem 2.1 is hard to solve since it admits the CSP as a special case.

Due to the difficulty of finding an exact solution with a reasonable computational effort, we propose a heuristic algorithm based on tabu search. As outlined at the end of Section 1.1, to simplify handling the constraints on both the part types schedule \( \beta \) and the pattern schedule \( \pi \) it is convenient to search solutions in the space of output sequences. For this reason we introduce the following problem \( \text{BSP}(\beta) \):

**Problem 2.2.** Find a schedulable cutting plan \( P \) that produces all the required part types in a prescribed order \( \beta \), with a minimum trim-loss.

Problem 2.2 can be tackled by a sequential heuristic, see Belov and Scheithauer, 2007, modified so as to complete part types in the prescribed order. As explained in Section 1.2, this algorithm does not generate patterns according to a proper pricing algorithm, and therefore does not provide in general an optimum.

A feasible solution of \( \text{BSP}(\beta) \) can also be found by choosing a CSP-feasible pattern set \( H \subseteq K \) fulfilling the following condition: for any \( i, j \in M \), let \( |\alpha(i) - \alpha(j)| \) be the *distance* between part types \( i \) and \( j \) in the sequence \( \beta \). Then
**Condition 2.2.** Any pattern of $H$ produces part types at reciprocal distance $\leq s - 1$ in the sequence $\beta$.

**Proposition 2.3.** Let $H$ respect Condition 2.2. Then processing its patterns in an order $\pi$ that first completes $\beta(1)$, then $\beta(2)$ and so on, never opens more than $s$ stacks.

The pattern set $H$ can be computed by using a pattern-based CSP formulation. Let $a_{ik}$ be the number of parts of type $i$ produced by the $k$-th pattern. Let then $K^t \subseteq K$ contain the patterns producing no parts out of $\{\beta(t), \ldots, \beta(t+s-1)\}$. The best feasible solution of $BSP(\beta)$ one can obtain from the patterns in $K^1, K^2, \ldots$ is an optimal solution of the following integer program $HBSP(\beta)$:

$$\min \sum_{t=1}^{m-s+1} \sum_{k \in K^t} x_k$$

$$\sum_{r=r(t)}^{t} \sum_{k \in K^r} a_{\beta(t)k} x_k \geq d_{\beta(t)} \quad t = 1, \ldots, m$$

$$x_k \geq 0 \text{ and integer} \quad k \in K^t, t = 1, \ldots, m - s + 1$$

where $r(t) = \max \{1, t-s+1\}$. $HBSP(\beta)$ is a classical Gilmore-Gomory CSP formulation where not all patterns are feasible, and its continuous relaxation can be computed via column generation.

Remark that optimal solutions of $HBSP(\beta)$ are feasible but may not be optimal for $BSP(\beta)$. Indeed, optimal solutions of $BSP(\beta)$ may not fulfill Condition 2.2: let us show it by an example.

**Example 2.4.** Let $m = 4$, $s = 2$, $w = 100$, $w = (6, 7, 8, 90)$, $d = (1, 1, 1, 3)$ and $\beta = (1, 2, 3, 4)$. Condition 2.2 requires $|\alpha(i) - \alpha(j)| \leq 1$ for any part types $i, j$ produced by the same pattern. However, an optimal solution of $BSP(\beta)$ runs patterns $[1, 0, 0, 1], [0, 1, 0, 1], [0, 0, 1, 1]$ at length 1, see Figure 1.a, and hence cuts three stock items: this solution is clearly schedulable, but the first two patterns do not respect the condition because produce parts of non-consecutive part types. The best solution respecting Condition 2.2, instead, cuts four stock items, see Figure 1.b.
2.1. Why Condition 2.2?

Condition 2.2 permits the use of a pattern-based formulation (problem (1)), whose continuous relaxation is widely recognized to provide very tight lower bounds. Its main advantage, however, is that it allows to easily control both order spread and part type incompatibility.

For the latter, cutting plans respecting compatibility constraints (such as in Matsumoto et al., 2010) can be obtained by searching output sequences in which the distance between incompatible part types is \( \geq s \).

The former is defined as follows: let \( H_i \subseteq H \) denote the subset of patterns producing part type \( i \), and \( \pi \) be a pattern sequence. The order spread of part type \( i \) in \( \pi \) is

\[
OS_i(\pi) = \max_{h,k \in H_i} \{\pi(h) - \pi(k)\}
\]

So, the mean and the maximum order spread are, respectively,

\[
\mathcal{OS}(\pi) = \frac{1}{m} \sum_{i \in M} OS_i(\pi) \quad OS(\pi) = \max_{i \in M}\{OS_i(\pi)\}
\]

On the basis of numerical experience, various Authors observed a high correlation between \( \mathcal{OS}(\pi) \) and \( s \) (Belov and Scheithauer, 2007; Foerster and Wäscher, 1998). Indeed, it is easy to see that \( \mathcal{OS}(\pi) \leq \frac{s|H|}{m} \) (recall that in practice \( |H| \simeq m \), thus \( \mathcal{OS}(\pi) \simeq s \)). Example 2.4 demonstrates instead that the maximum order spread is not correlated in general to \( s \). However

**Proposition 2.5.** Let \( H^R \) be the set of patterns obtained by solving the continuous relaxation of (1) (notice that \( H^R \) respects Condition 2.2). Then every feasible order \( \pi \) of \( H^R \) is such that \( OS(\pi) < 2s \).
Proof. In fact, Condition 2.2 ensures that part type \( i \) is produced together part types positioned from \( \alpha(i) - s + 1 \) to \( \alpha(i) + s - 1 \). Hence, the columns of problem (1) that produce \( i \) have non-zero elements in \( \leq 2s - 1 \) consecutive rows. As a solution of the continuous relaxation of (1) is formed by linearly independent columns, no more than \( 2s - 1 \) of these correspond to patterns producing part type \( i \).

Note that good integer solutions obtained by rounding the continuous relaxation of (1) add in practice very few patterns to \( H^R \). Consequently, these solutions admit an order \( \pi \) where \( OS(\pi) \) does not outnumber the bound \( 2s - 1 \) too much. Moreover,

**Proposition 2.6.** Let \( s = 2 \) and \( H \subseteq K \) be a solution of \( BSP(\beta) \) compliant with Condition 2.2. Then there exists an order \( \pi \) of \( H \) such that all part types are produced with no preemption.

**Proof.** Due to Condition 2.2, every pattern of \( H = H^1 \cup H^2 \cup \ldots \) produces \( \leq 2 \) consecutive part types. Patterns producing 2 part types, say those in positions \( t \) and \( t + 1 \), belong to \( K^t \), whereas single-part type patterns, i.e. those producing just the part type in position \( t \), belong to \( K^t \cap K^{t-1} \). Assign those patterns to \( H^t \) and schedule the single-part type patterns in this set before all the others. The sequence \( \pi \) so obtained is non-preemptive.

Equivalently, Proposition 2.6 states the existence of an order \( \pi \) that minimizes the number of discontinuities, see Yanasse, 1997.

3. The algorithm

In the following, let \( CSP(s) \) denote a cutting stock problem where each pattern is constrained to produce parts of \( \leq s \) distinct part types, and, for any problem \( P \) admitting an integer programming formulation, let \( P^R \) denote its continuous relaxation. Clearly \( CSP(s) \) is a relaxation of \( BSP \), since every BSP-feasible pattern has \( \leq s \) non-zeroes.

Basically, our algorithm repeatedly finds an output sequence \( \beta \) and a pattern set \( H \) that is feasible for \( BSP(\beta) \). In order to ensure BSP-feasibility, \( H \) is computed by solving \( HBSP^R(\beta) \) and rounding the fractional solution; then we heuristically move from \( \beta \) to some \( \beta' \) guided by a tabu-search scheme. We can resume the procedure as follows:

1. Find a pattern set \( H_0 \) and a sequence \( \beta_0 \);
2. if $H_0$ fulfills Condition 2.2 under $\beta_0$, then output $H_0$ and stop; otherwise,

3. construct an initial feasible solution that outputs parts according to $\beta_0$ and fulfills Condition 2.2;
4. define a neighborhood of $\beta_0$;
5. for each neighbor $\beta$ of $\beta_0$, find a pattern set $H$ that fulfills Condition 2.2 under $\beta$, keeping track of the less expensive;
6. penalize the patterns of $H_0$ that do not fulfill Condition 2.2, and go back to Step 1.

**Step 1**

Pattern set $H_0$ is obtained by solving $CSP^R(s)$. To compute $\beta_0$ we proceed as follows.

- Introduce an undirected multigraph $G = (M, E)$ with $m$ vertices and an edge $ij$ for any pair $i, j$ of part types whose parts are produced by the same pattern of $H_0$ (every pattern thus defines a clique of $G$).

- Given a linear arrangement of the vertices of $G$, let as usual $\beta(i)$ denote the position of the $i$-th vertex (i.e., part type) of $G$: the bandwidth of $G$ under $\beta$ is

$$\Delta(\beta) = \max_{ij \in E} |\beta(i) - \beta(j)|$$

The BANDWIDTH problem calls for finding $\beta^*$ such that $\Delta^* = \Delta(\beta^*) \leq \Delta(\beta)$ for any arrangement $\beta : M \to \{1, \ldots, m\}$.

- Since BANDWIDTH is NP-hard, a heuristic is used to find a linear arrangement $\beta_0$ that tries to approximate $\Delta^*$.

**Step 2**

$H_0$ fulfills Condition 2.2 if and only if $\Delta^* < s$. Thus, if $\Delta(\beta_0) < s$, then $H_0$ is a solution of $HBSP^R(\beta_0)$ and hence its rounding is BSP-feasible.

**Step 3**

Otherwise, the rounding of $H_0$ may or not be BSP-feasible, but instead of trying to sequence it and meet BSP-feasibility (which would mean solving a difficult pattern sequencing problem with no feasibility guarantee), a new
pattern set is computed by solving the continuous relaxation of (1), i.e. a CSP where patterns are generated according to Condition 2.2 written for $\beta_0$.

**Step 4**
In order to improve the current solution, we then search among the sequences neighbouring $\beta_0$. Search proceeds by swapping elements of the sequence (vertices of $G$). Not all the neighbours are explored, but only a critical set within which we select a $\beta$ trying to minimize the violation of $\Delta(\beta) < s$.

**Step 5**
For such $\beta$ we calculate the optimal value of $H_{BSP}^R(\beta)$ applying column generation to (1). If the obtained fractional solution is promising, i.e., if its value is less than the current best integer solution minus one, it is rounded in order to get a new feasible solution of $BSP$ which, if the case, replaces the best solution found so far. Search is halted as soon as either optimality is proved or the current sequence has not been improved (in terms of reduction of violation) for a prescribed number of iterations.

**Step 6**
Diversification is then tried by restarting the procedure from a different solution of $CSP(s)$. To find this solution, we penalize the patterns of $H_0$ that do not fulfill Condition 2.2 under the best sequence found so far.

The algorithm outlined above returns a part type sequence $\beta$ and a schedulable (but unscheduled) pattern set $H$. The procedure is then completed by scheduling the patterns of $H$ according to $\beta$ (see Propositions 2.3, 2.6). Figure 2 shows a schematic flow diagram of the method proposed.

Figure 2: Flow diagram of the algorithm.

4. Algorithm details

A detailed description of the method is provided by the pseudo-code Algorithm 1. In the following we go through each of the functions used.

4.1. Solving $CSP^R(s, c)$

The pricing problem of $CSP^R(s)$ is an integer knapsack with the restriction that no more than $s$ variables can be non-zero. In our context, however,
Algorithm 1

\[ \begin{align*}
    z_{\text{best}} & \leftarrow \infty; \quad z_L \leftarrow \lceil \text{CSP}_R(s) \rceil \\
    i & \leftarrow 0; \quad c \leftarrow 1 \\
    \textbf{while} \ i < \text{restarts} \ \textbf{do} \\
    & \quad H \leftarrow \text{CSP}_R(s, c) \\
    & \quad G \leftarrow \text{setGraph}(H) \\
    & \quad \beta \leftarrow \text{bandwidth}(G) \\
    & \quad \{\text{tabu search}\} \\
    & \quad d_{\text{best}} \leftarrow \text{getViolation}(G, \beta) \\
    & \quad j \leftarrow 0 \\
    & \quad \textbf{while} \ j < \text{fails} \ \textbf{do} \\
    & \quad \quad \beta \leftarrow \text{getCriticalNeighbour}(G, \beta) \\
    & \quad \quad d \leftarrow \text{getViolation}(G, \beta) \\
    & \quad \quad \textbf{if} \ d \leq d_{\text{best}} \ \textbf{then} \\
    & \quad \quad \quad \beta \leftarrow \beta; \quad j \leftarrow 0 \\
    & \quad \quad \quad z^R \leftarrow \text{HBSP}_R(\beta) \\
    & \quad \quad \quad \textbf{if} \ z^R < z_{\text{best}} - 1 \ \textbf{then} \\
    & \quad \quad \quad \quad z_{\text{best}} \leftarrow \min(z_{\text{best}}, \text{ROUNDING}(z^R)) \\
    & \quad \quad \quad \textbf{if} \ z_{\text{best}} = z_L \ \textbf{then} \ \textbf{stop} \\
    & \quad \quad \textbf{end if} \\
    & \quad \quad \textbf{else} \\
    & \quad \quad \quad j \leftarrow j + 1 \\
    & \quad \quad \textbf{end if} \\
    & \quad \textbf{end while} \\
    & \quad \{\text{Diversification}\} \\
    & \quad c \leftarrow \text{update}(G, \beta) \\
    & \quad i \leftarrow i + 1 \\
    & \quad \textbf{end while} \\
\end{align*} \]

A little more sophistication is required: diversification in fact entails the use of more than one multigraph \( G \), each derived from a distinct solution of \( \text{CSP}_R(s) \). Such solutions can be computed by altering the cost vector \( c \) of \( \text{CSP}_R(s) \): let \( \text{CSP}_R(s, c) \) denote the problem so perturbed.

A perturbation is made in such a way that patterns causing the current \( G \) a bandwidth \( > s \) are penalized (see §4.8). Denoting with \( k \) the number of perturbed patterns in the current \( \text{CSP}_R(s, c) \), one has to compute the \((k+1)\)-th optimal knapsack to prevent the pricing algorithm from generating
again one of the penalized patterns (which can happen because in the pricing problem reduced costs are evaluated with respect to non-perturbed costs). In practice, it is sufficient to maintain a pool with \( k + 1 \) columns having negative reduced cost. Column generation halts when all the columns in the (possibly empty) pool belong to the current master problem.

The pricing algorithm implemented modifies as follows the branch-and-bound algorithm proposed by Horowitz and Sahni, 1974, for 0-1 Knapsack:

- Integer variables are binary-encoded and \textsc{Integer Knapsack} is so transformed into 0-1 (Vanderbeck, 1999).

- The forward move of Horowitz-Sahni is implemented by allowing no more than \( s \) insertions of distinct part-types in the current solution. This check just requires a counter, because in the non-increasing rank of profit-to-size ratios binary variables are grouped per part types.

- Since we are interested in any set of \( k + 1 \) solutions with negative reduced cost, the lower bound is set to the constant value 1.0 and the algorithm terminates as soon as \( k + 1 \) solutions have been found or the maximum number of nodes (set in our tests to \( 5 \cdot 10^6 \)) has been explored.

4.2. Generating the multigraph by \textsc{setGraph}(H)

\textsc{setGraph}(H) generates a multigraph \( G \) with \( m \) nodes (associates to part types) and an edge \( ij \) whenever parts of batches \( i \) and \( j \) are cut through a pattern of \( H \). Notice that the solution of the \textsc{Bandwidth} problem on graphs is also proposed in other works on pattern sequencing, e.g., Madsen, 1979. In those cases, however, the graph has a node per pattern and an edge between pairs of patterns that share a part type.

4.3. Finding an initial output sequence by \textsc{Bandwidth}(G)

An initial sequence \( \beta_0 \) of the nodes of \( G \) is computed via the heuristic algorithm proposed by Gibbs et al., 1976, for the \textsc{Bandwidth} problem. This algorithm first elaborates a \textit{level structure} of the graph, that is a partition of the vertices into sets \( L_1, \ldots L_b \) (called \textit{levels}), where all the nodes adjacent to nodes in level \( L_i \) are in either the same level, the level \( L_{i-1} \) (if \( i > 1 \)) or the level \( L_{i+1} \) (if \( i < b \)). Consecutive positions in the sequence are then assigned to nodes belonging to the same level, starting from \( L_1 \) and proceeding for increasing levels.
The bandwidth found clearly depends on the number of nodes in each level, which in turn depends on the number of levels, that is the structure depth. The goal then is to determine a structure with as many levels as possible. In Gibbs et al., 1976, it is observed that many-level structures are often generated starting from the extremes of the diameter of $G$. Since diameter computation is hard, Gibbs et al. use the extremes $u, v$ of a pseudo-diameter, generate two structures from $u$ and from $v$, and eventually merge them into a new one which typically turns out to be deeper than the parent structures.

4.4. Measuring violation of Condition 2.2: \textsc{getViolation($G, \beta$)}

We call \textit{critical} with respect to a given $\beta$ any $ij \in E$ such that $|\alpha(i) - \alpha(j)| \geq s$. The presence of a critical edge makes $\beta$ violate Condition 2.2. An evaluation of “how much” does $\beta$ violate Condition 2.2 is given by

$$d(\beta) = |\{ij \in E : |\alpha(i) - \alpha(j)| \geq s\}|.$$

4.5. Searching a neighbour sequence: \textsc{getCriticalNeighbour($G, \beta$)}

As in many tabu search algorithms for sequencing problems, we define a neighbour of a given $\beta$ as a sequence obtained by swapping a pair of vertices of $\beta$. A sequence $\beta'$ improves the current best solution $z_{\text{best}}$ if $\text{HBSP}(\beta') < z_{\text{best}}$. Since the number of swaps grows quadratically with $m$ and computing $\text{HBSP}(\beta')$ is rather expensive, as it requires column generation, not all the swaps are evaluated and the filter is given by $d(\beta')$. In particular, \textsc{getCriticalNeighbour($G, \beta$)} returns the best neighbour $\beta'$, filtered with respect to the violation $d$, among those obtained by performing a non-tabu swap $(u, v)$ between a critical vertex $u$, i.e., an extreme of a critical edge, and all other vertices of $G$. A swap is tabu if it has been performed in one of the last \textit{tenure} iterations. The \textit{tenure} parameter is fixed for each execution and ranging between 20 and 40 in our experiments. For a survey on tabu search algorithms see Glover and Laguna, 1997.

4.6. Solving $\text{HBSP}^R(\beta)$

$\text{HBSP}^R(\beta)$ is the continuous relaxation of program (1), a Gilmore-Gomory CSP formulation restricted to the patterns fulfilling Condition 2.2 for sequence $\beta$. Full pricing requires the solution of $m - s + 1$ unbounded integer
knapsacks, each returning a pattern of $K^t$ with (possibly) negative reduced cost. The $t$-th knapsack is

$$P_t : \max \left\{ \sum_{i \in M^t} \lambda_i^* y_i \mid \sum_{i \in M^t} w_i y_i \leq w, y_i \geq 0 \text{ integer} \right\}$$

with $M^t$ containing the $s$ part types produced by $K^t$, i.e., those between $\beta(t)$ and $\beta(t + s - 1)$, and $\lambda_i^*$ optimal dual value of the $i$-th part type of $M^t$ in the current master problem. However, pricing has been implemented applying a partial pricing strategy, because after part types in positions $\alpha(i)$ and $\alpha(j)$ have been swapped, useful columns are most likely generated in sets $K_{\max}\{1, \alpha(i) - s + 1\}, \ldots, K_{\min}\{\alpha(i), m - s + 1\}$ and $K_{\max}\{1, \alpha(j) - s + 1\}, \ldots, K_{\min}\{\alpha(j), m - s + 1\}$.

### 4.7. Getting a feasible solution: ROUNDING($z^R$)

A BSP-feasible solution is obtained by rounding the fractional one computed via (1). Classically, fractional components are rounded down to the closest integer; residual demand is then fulfilled using patterns respecting Condition 2.2. To do so, residual part types are selected as ranked in $\beta$ and are assigned to patterns in a First Fit manner in a way that also satisfies Condition 2.2. Rounding is performed only when the fractional solution $z^R$ is promising, i.e., for $z^R < z_{best} - 1$. The execution of the algorithm ends if the current best integer value $z_{best}$ reaches the lower bound $z_L$ obtained by rounding up the optimal value of $CSP(s)$.

### 4.8. Diversification: UPDATE($G$, $\beta$)

At the end of tabu search, the exploration is diversified: an alternative solution of $CSP^R(s)$ is found, a new graph $G$ constructed and tabu search restarted. Let $Q_h$ be the clique of $G$ associated to the pattern $h$ of the current $CSP^R(s)$ solution, and $\beta$ the best sequence found by the tabu search. When sequence $\beta$ is applied, a measure of how the pattern $h$ violates the Condition 2.2 is the following:

$$D_h = |\{(i, j) \in Q_h : |\bar{\alpha}(i) - \bar{\alpha}(j)| \geq s\}| - |\{(i, j) \in Q_h : |\bar{\alpha}(i) - \bar{\alpha}(j)| < s\}|$$

To find the new solution of $CSP^R(s)$, the cost coefficients of the $CSP^R(s)$ formulation are updated as follows:

$$c_h = \max\{1.0, D_h \cdot \text{perturb\_coeff}\}$$

for a given value of $\text{perturb\_coeff}$. 

16
5. Experimental results

A set of computational experiments has been carried out to evaluate the efficiency and effectiveness of the algorithm described above. In order to compare the outcome with literature, the experience refers to the basic setting, hence no precedence and compatibility constraints were considered.

5.1. Computational resources

The algorithm was coded in C++ and compiled with Microsoft cl compiler (version 12.00.8804) with option /O2. Numerical precision was set to $10^{-8}$. Test problems were solved on a Intel Core 2 Duo E8500 with 3Gb RAM. Linear programs were solved by LPSolve 5.5 with default settings. Time and size limits were set to 3600 seconds and 25000 columns, respectively.

5.2. Test bed organization

Test instances from the literature were used, organized in classes with similar features.

To construct the test bed we started from 184 test problems organized into eight classes $BS_1, \ldots, BS_8$ of twenty instances each (proposed by Belov and Scheithauer, 2007), and three classes $M_1, \ldots, M_3$ of eight instances each (proposed by Matsumoto et al., 2010). For each instance, we solved Problem 2.1 with $s \in \{2, 3, 4, 6, 10\}$: thus we solved 920 problems overall.

According to Authors, instances in each class $BS_i$ have been generated at random so as to span different cases (in terms of demand, and part type and stock-item widths) with a minimum number of parameter variations. Table 1 shows the distribution parameters: columns $w_i$ and $d_i$ define the intervals from which width and demand of parts have been uniformly picked up.

Classes $M_1$, $M_2$ and $M_3$ are respectively formed by instances with 10, 20 and 50 part types generated by CUTGEN (Gau and Wäscher, 1995) with various parameter settings (see Matsumoto et al., 2010, Table 1).

5.3. Code tuning

Experiments were done to set parameters restarts, fails and tenure appropriately. The best trade-off setting for $s > 2$ and $m > 20$ turned out to be restarts = 40, fails = 200 and tenure = 40. For $s = 2$, tabu search generates relatively few columns, hence we incremented diversification setting restarts = 200. Also, for small instances ($m \leq 20$) the size of the neighborhood search space reduces, so a value of 20 resulted adequate for tenure.
5.4. Computational results

Computational results are listed in Table 2 (Belov and Scheithauer instances) and Table 3 (Matsumoto et al. instances). Each row indicates mean values in the relevant class. Column 2 gives the number of stacks available. Columns from 3 to 8 respectively give an over-approximation of the optimality gap \( \frac{z_{\text{best}} - \lceil CSP^R(s) \rceil}{\lceil CSP^R(s) \rceil} \), the number of columns generated and the computation time of tabu search without diversification (first execution) and with diversification (end of algorithm). Column 9 reports in how many cases, out of the class, the algorithm was able to certify optimality (gap = 0). Column 10 indicates the diversification step at which the best integer solution was found. Finally, columns 11 and 12 give the mean and the maximum order spread (average and max number of patterns required to produce a part type), respectively.

5.5. Discussion

Here we discuss our major observations on the experiments carried out.

Algorithm effectiveness

Like the optimal value of \( BSP \), also the gap between \( z_{\text{best}} \) and \( \lceil CSP^R(s) \rceil \) decreases as \( s \) increases (with two exceptions in \( BS_6 \) and \( BS_7 \), observed when passing from \( s = 2 \) to \( s = 3 \): this most likely depends on parameter setting, see Section 5.3). For classes \( BS \) with \( s \geq 6 \) and classes \( M \) with \( s \geq 3 \), the gap is always below 1%. The worst mean performance is definitely on \( BS_4 \) — the sole class with very low average demand — independently on \( s \). This outcome mainly derives from the high variance of the results (\( \sigma^2 = 3.87 \) vs. values below 1.2 for other classes), as the optimal solutions found in \( BS_4 \) are not the fewest in the experiment (the worst case is in fact \( BS_7 \)).

One can argue that, as in the CSP, the gap increases as demand decreases (although too much little demand can make the BSP not sensible, because it is more likely that an CSP optimal solution fulfills the demand of each part type with a single cut: in the extreme situation of unit demand, the BSP becomes in fact meaningless). Further experiments were done in order to verify a possible correlation between demand and gap, and new classes of instances were then constructed by progressively reducing the demand of existing ones. The reduction was operated on the instances of class \( BS_6 \), the one with the most homogeneous demand values (see Table 1), from 80% down to 20% the original value. Tests were run for \( s = 2, 3, 4, 6, 10 \), hence
solving 400 new problems on the whole. Average gaps and CPU times are summarized in the charts of Figures 3 and 4. The gaps actually increase significantly (from 46% for $s = 3$ to 225% for $s = 10$); in absolute terms, however, the deterioration is always less than 1%. CPU time is roughly stable for $s = 2$ and $s = 10$; for intermediate values of $s$ an increase of CPU time, still not dramatic, is observed as soon as demand is reduced to 80%; from then on, the CPU time remains stable.

Data show that the algorithm is most effective in class $BS_8$: excluding $M_1$, whose instances have 10 part types only, the mean gap in $BS_8$ is the best for all values of $s$ but $s = 4$ (here it is slightly worse than $BS_1$), and is constantly below 1%. If instead one measures effectiveness by optimality certification (column 9), then the best performance is obtained independently on $s$ for classes $M_1$ (33 optima in 40 cases) and $BS_1$ (51 optima in 100 cases).

For benchmarking we adopted the sequential heuristic proposed by Belov and Scheithauer, 2007. Although more efficient than our algorithm, this heuristic indeed solves a problem much simpler than ours: a comparison of computation times (see below) is therefore inappropriate and we limited our attention to the quality of solutions. For $BS$ instances, in twelve (in sixteen) cases out of forty, the mean (the maximum) gap improves that obtained by the benchmark heuristic, see optimality gaps in bold in Table 2. Conversely, the mean (the maximum) gap left to the benchmark more than 1% in just six (in five) cases out of forty, see optimality gaps in italic in Table 2.

**Algorithm efficiency**

CPU time directly relates to the number of columns generated: a correlation figure of 0.81 for classes $BS$ and of 0.73 for classes $M$ can be observed. The average number of columns tends to increase with $s$ up to some point ($s = 6$ in most classes $BS$ and $s = 3$ in classes $M$), then rapidly drops. This behaviour can be explained noting that, as $s$ increases, the pattern search space indeed increases, but the tabu search is more effective and this often causes a quick convergence towards the lower bound provided by $CSP(s)$. The latter aspect is confirmed by the value of restarts at which the best solution is found: such a value roughly decreases for increasing $s$, independently on instance class. Note in particular that in $BS_1$ and $BS_2$, and for $s = 10$, the algorithm finds the best integer solution at the very first iteration (in those cases, therefore, diversification turned out to be useless). Same in all classes $M$ for $s \geq 6$, and in some cases for $s < 6$.

As far as large instances are concerned (50 part types), the best algorithm
performance was on average observed for $BS_8$, the worst (both in terms of CPU time and mean number of columns generated) for $BS_3$. The hardest case in $BS_3$ occurs for $s = 6$, with about 250 CPU seconds per instance.

**Effectiveness vs. efficiency**

This relation can be evaluated via the ratio \( \frac{\text{gap improvement}}{\text{CPU time increase}} \). Parameter \texttt{restarts} plays here a relevant role, and can be tuned by comparing the results obtained at the first iteration (columns without \texttt{restarts}) to the final ones (columns with \texttt{restarts} in the tables): independently on class, the optimality gap improvement after the first iteration is less and less marked as \( s \) increases. With the exception of \( s = 2 \) for which \texttt{restarts} was set to 200, also the ratio \( \frac{\text{gap improvement}}{\text{CPU time increase}} \) decreases as \( s \) increases, again independently on class. We can therefore deduce that the optimal value of \texttt{restarts} goes inversely with \( s \).

**Cutting Stock and Order Spread**

It is interesting to assess the impact that limited buffers have on trim-loss minimization. In our experiments, the gap between the solutions of \( CSP \) and \( [CSP^R(s)] \) — not reported in the tables — is always 0 for all classes \( M \) and for classes \( BS \) with \( s \geq 4 \). For classes \( BS \) with \( s = 3 \) a positive gap occurs in just 7 cases and is always below 1%. A positive gap ranking between 0.1\% (\( BS_2 \)) and 1.55\% (\( BS_4 \)) was detected in 129 cases of classes \( BS \) with \( s = 2 \). Therefore, for \( s \geq 4 \) the gap between optimal \( CSP \) and \( BSP \) solutions is negligible and does not derive from the restriction to the number of allowed part-types per pattern. For \( s < 4 \) instead, the way on how Condition 2.2 and the constraint on the number of part-types per pattern affect the gap is not clear and deserves further investigation.

Due to patterns added by rounding, the maximum order spread is greater than the theoretical indication \( 2s - 1 \) (Proposition 2.5) in 278 cases out of 920, more than half of which for \( s \leq 3 \). In 2 cases the max order spread is \( 2s + 2 \) and in 43 cases it is \( 2s + 1 \).

The order spread per part type was on average observed to be about 40\% the theoretical indication \( 2s - 1 \) in classes \( BS \), and 27\% in classes \( M \). In the former case, it ranked from 19\% to 58\%, in the latter from 15\% to 39\%. 

20
Table 1: Classes of instances by Belov and Scheithauer, 2007.

<table>
<thead>
<tr>
<th>class</th>
<th>m</th>
<th>w</th>
<th>$w_i$</th>
<th>$d_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS₁</td>
<td>20</td>
<td>10000</td>
<td>[100, 7000]</td>
<td>[1, 100]</td>
</tr>
<tr>
<td>BS₂</td>
<td>50</td>
<td>10000</td>
<td>[100, 2000]</td>
<td>[1, 100]</td>
</tr>
<tr>
<td>BS₃</td>
<td>50</td>
<td>10000</td>
<td>[100, 4000]</td>
<td>[1, 100]</td>
</tr>
<tr>
<td>BS₄</td>
<td>50</td>
<td>10000</td>
<td>[100, 7000]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>BS₅</td>
<td>50</td>
<td>10000</td>
<td>[100, 7000]</td>
<td>[1, 100]</td>
</tr>
<tr>
<td>BS₆</td>
<td>50</td>
<td>10000</td>
<td>[100, 7000]</td>
<td>[50, 100]</td>
</tr>
<tr>
<td>BS₇</td>
<td>50</td>
<td>10000</td>
<td>[2000, 4000]</td>
<td>[1, 100]</td>
</tr>
<tr>
<td>BS₈</td>
<td>50</td>
<td>10000</td>
<td>[2000, 7000]</td>
<td>[1, 100]</td>
</tr>
</tbody>
</table>

Figure 3: optimality gap (%) vs. demand downsizing (% of original demand) in BS₆ instances.

Figure 4: CPU time (seconds) vs. demand downsizing (% of original demand) in BS₆ instances.

6. Extensions

6.1. Heterogeneous batches

In this paper we basically focus on homogenous batches but in several context client orders correspond to heterogeneous batches. When batches of this sort are present, buffers can be either operated with batch stacking, or with part-type stacking.

In case of part-type stacking, instead, controlling open stacks does not correspond to controlling open orders, and the algorithm cannot be directly used as is: it is in fact required to decide (i) how should a part-type belonging
Table 2: Results for Belov and Scheithauer instances.

<table>
<thead>
<tr>
<th>instance class</th>
<th>s</th>
<th>gap without restarts (%)</th>
<th>cols without restarts (#)</th>
<th>time without restarts (sec)</th>
<th>gap with restarts (%)</th>
<th>cols with restarts (#)</th>
<th>time with restarts (sec)</th>
<th>opt. restart</th>
<th>best OS (# pat.)</th>
<th>OS (# pat.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BS₁</td>
<td>2</td>
<td>2.77 73.70 0.05</td>
<td>1.11 151.30 7.55</td>
<td>3 79.80 1.35 3.3</td>
<td>1.85 2.01 6</td>
<td>0.07 88.05 0.20</td>
<td>15 3.15 3.94 9.6</td>
<td>0.05 16.00 4.97 13.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₂</td>
<td>2</td>
<td>2.40 327.45 0.78</td>
<td>1.71 1695.55 158.88</td>
<td>3 79.80 0.73 3.9</td>
<td>2.85 0.59 37.15</td>
<td>0.13 7.95 2</td>
<td>13 1.85 4.76 9.9</td>
<td>1.00 17.6 14.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₃</td>
<td>2</td>
<td>2.65 319.45 0.80</td>
<td>1.57 1262.90 157.79</td>
<td>0 76.55 1.65 3.9</td>
<td>8.88 2.57 112.01</td>
<td>0.20 3.37 1</td>
<td>4 6.5 9.24 10.9</td>
<td>0.09 111.25 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₄</td>
<td>2</td>
<td>5.21 252.50 0.65</td>
<td>4.74 889.45 116.75</td>
<td>0 65.50 0.74 2.9</td>
<td>3.83 159.80 132.5</td>
<td>0.15 103.62 123.47</td>
<td>12 4.05 3.69 11.5</td>
<td>0.09 424.90 0.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₅</td>
<td>2</td>
<td>3.88 288.70 2.85</td>
<td>3.51 1570.90 103.62</td>
<td>1 7.80 1.17 3.8</td>
<td>4.27 139.55 3</td>
<td>0.61 705.45 284.69</td>
<td>10 2.40 3.2 7.3</td>
<td>4.97 227.35 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₆</td>
<td>2</td>
<td>1.59 364.55 2.93</td>
<td>0.72 2272.35 86.19</td>
<td>7 4.95 2.32 7.1</td>
<td>6.40 777.90 8.90</td>
<td>0.23 1039.40 395.17</td>
<td>1 4.65 3.24 10.9</td>
<td>11 1.08 8.07 16.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₇</td>
<td>2</td>
<td>4.57 332.55 3.18</td>
<td>0.24 3041.45 129.14</td>
<td>2 16.00 4.81 10.8</td>
<td>10 0.33 344.80 1.99</td>
<td>0.09 2248.60 81.26</td>
<td>4 7.95 6.85 16.7</td>
<td>0.09 253.45 1.34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₈</td>
<td>2</td>
<td>2.61 246.70 2.37</td>
<td>1.77 927.25 132.25</td>
<td>0 99.15 1.59 3.7</td>
<td>6.20 294.55 3.18</td>
<td>0.35 2747.85 123.45</td>
<td>10 2.00 8.25 17.6</td>
<td>0.09 305.15 0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₉</td>
<td>2</td>
<td>2.77 203.15 0.75</td>
<td>1.45 675.45 127.16</td>
<td>0 82.70 1.51 3.9</td>
<td>2.01 315.85 4.35</td>
<td>0.16 1943.75 172.69</td>
<td>10 3.91 8.09 17.2</td>
<td>0.07 203.15 0.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₁₀</td>
<td>2</td>
<td>3.37 281.55 3.40</td>
<td>1.83 1487.85 136.14</td>
<td>0 15.00 2.32 5.8</td>
<td>4.09 160.65 0.42</td>
<td>0.11 360.35 8.41</td>
<td>2 4.85 2.07 6.5</td>
<td>0.09 305.15 0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS₁₁</td>
<td>2</td>
<td>6.20 135.25 0.21</td>
<td>0.05 212.05 3.09</td>
<td>10 3.65 2.68 9.7</td>
<td>6.67 305.15 0.02</td>
<td>0.04 126.95 0.18</td>
<td>11 1.50 3.68 13.2</td>
<td>0.09 305.15 0.02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

22
Table 3: Results for Matsumoto et al. instances.

<table>
<thead>
<tr>
<th>instance class</th>
<th>without restarts</th>
<th>with restarts</th>
<th>opt. restart</th>
<th>OS (# pat.)</th>
<th>OS (# pat.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s</td>
<td>gap (%)</td>
<td>cols (#)</td>
<td>time (sec)</td>
<td>gap (%)</td>
</tr>
<tr>
<td>M1</td>
<td>2</td>
<td>3.28</td>
<td>18.00</td>
<td>0.02</td>
<td>1.93</td>
</tr>
<tr>
<td>M1</td>
<td>3</td>
<td>0.05</td>
<td>10.00</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>M1</td>
<td>4</td>
<td>0.00</td>
<td>10.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>M1</td>
<td>6</td>
<td>0.00</td>
<td>15.00</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>M1</td>
<td>10</td>
<td>0.00</td>
<td>20.00</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>M2</td>
<td>2</td>
<td>3.07</td>
<td>93.63</td>
<td>0.08</td>
<td>1.51</td>
</tr>
<tr>
<td>M2</td>
<td>3</td>
<td>0.52</td>
<td>104.38</td>
<td>0.13</td>
<td>0.45</td>
</tr>
<tr>
<td>M2</td>
<td>4</td>
<td>0.22</td>
<td>51.13</td>
<td>0.05</td>
<td>0.22</td>
</tr>
<tr>
<td>M2</td>
<td>6</td>
<td>0.12</td>
<td>29.25</td>
<td>0.00</td>
<td>0.12</td>
</tr>
<tr>
<td>M2</td>
<td>10</td>
<td>0.12</td>
<td>29.25</td>
<td>0.00</td>
<td>0.12</td>
</tr>
<tr>
<td>M3</td>
<td>2</td>
<td>3.94</td>
<td>195.13</td>
<td>0.39</td>
<td>1.82</td>
</tr>
<tr>
<td>M3</td>
<td>3</td>
<td>1.10</td>
<td>257.13</td>
<td>0.88</td>
<td>0.50</td>
</tr>
<tr>
<td>M3</td>
<td>4</td>
<td>0.40</td>
<td>243.25</td>
<td>0.78</td>
<td>0.12</td>
</tr>
<tr>
<td>M3</td>
<td>6</td>
<td>0.06</td>
<td>49.13</td>
<td>0.01</td>
<td>0.06</td>
</tr>
<tr>
<td>M3</td>
<td>10</td>
<td>0.06</td>
<td>49.13</td>
<td>0.01</td>
<td>0.06</td>
</tr>
</tbody>
</table>

to distinct orders be stacked (it can in fact be either placed onto a single stack or distributed among more than one) and (ii) when should a stack be closed (as soon as a part-type or an order is completed).

Clearly, batch and part-type stacking are equivalent when batches are homogenous.

6.2. Asynchronous stacking

The downstream buffer capacity differently affects operations, depending on whether stacking and cutting are synchronous or not. With synchronous stacking, stacks are opened or closed as soon as a stock item has completely been cut, and before processing the next one: in this case the number of open stacks can change only after the completion of a cutting pattern and before closing a consolidated stack. With asynchronous stacking, stacks can be opened at any time: therefore, the number of open stacks evolves over time also depending on the order in which parts are cut within a given pattern. As a consequence, the condition requiring that every cutting pattern produces $\leq s$ part-types is no longer necessary. For instance, consider a problem where $m = 2$, $s = 1$, $w = 100$, $w = (30, 20)$, $d = (10, 13)$. An optimal solution with asynchronous stacking is given by the pattern sequence $\langle [3, 0], [1, 3], [0, 5] \rangle$. 

23
which requires 5 stock items: the second pattern of this sequence has \( > s \) part types; conversely, an optimal solution with synchronous stacking is trivially given by the sequence \( ([3, 0], [0, 5]) \), which cuts 7 stock items.

The algorithm can be extended with no difficulty to asynchronous stacking. Let \( \bar{K} \) be the set of patterns that produce \( > s \) part types. It is immediate to see that \( \bar{K} \) can be partitioned into \( \bar{K}^t \), \( 1 \leq t \leq m - s \), so that every pattern in \( \bar{K}^t \) can appear in the sequence only after all patterns in \( \bar{K}^t \) and before all those in \( \bar{K}^{t+1} \). Such a pattern cannot therefore produce more than the \( s + 1 \) part-types produced by any pattern in \( \bar{K}^t \cup \bar{K}^{t+1} \), and its run length never exceeds 1. Let \( \bar{K}^0 = \bar{K}^{m-s+1} = \emptyset \). Model (1) becomes:

\[
\min \sum_{t=1}^{m-s+1} \sum_{k \in K^t \cup K^{t+1}} x_k \\
\sum_{r=r(t)} x_k \leq 1 \quad t = 1, \ldots, m - s \\
\sum_{k \in K^t} x_k \geq 0 \text{ and integer} \quad k \in K^t \cup K^{t+1}, t = 1, \ldots, m - s + 1
\]

and column generation must also include the following \( m-s \) pricing problems:

\[
P_t: \max \left\{ -\sigma_t + \sum_{i \in M^t \cup M^{t+1}} \lambda_i^* y_i \mid \sum_{i \in M^t} w_i y_i \leq w, y_i \geq 0 \text{ integer} \right\}
\]

where \( 1 \leq t \leq m - s \), and \( \sigma_t \) is the dual variable associated with the \( t \)-th constraint (2).

Finally note that, in solutions fulfilling Condition 2.2, asynchronous stacking never saves more than \( m - s \) cuts, and this advantage is paid by the addition of \( m - s \) new patterns. With this type of stacking, then, it is sensible to use the proposed algorithm for very low production volumes, and when raw material is relatively expensive.

### 7. Conclusions and future research

The problem of finding a cutting plan — patterns, run lengths and pattern sequence — that produces required batches of parts in a way compliant
with resource availability — downstream buffers — and specific constraints — batch precedence and/or compatibility — has been considered. This problem has practical relevance in industries adopting recent technology for material cutting: here, cutting workcentres are closely associated with automated handling systems whose operation, due to limited buffer capacity, must be properly scheduled; moreover, enhanced production planning capabilities of the manufacturing system may involve precedence relations and/or incompatibilities among batches/orders. Both these issues impact on cutting operations, which have to be planned with a goal of minimizing trim loss in accordance with the described scenario.

For this problem, we devised an algorithm operating a tabu search on batch output sequences, guided by the solution of (the continuous relaxation of) a suitable cutting stock problem. The algorithm has nice properties, not only in terms of effectiveness and efficiency, but also because it keeps under control such indicators as the maximum order spread. Traditionally, the spread of an order is counted as the maximum distance, in the pattern sequence, of those patterns producing parts of the relevant batch. Although this figure is useful for instances with negligible machine set-up times and/or low demands, it does not actually measure the permanence of the order in the system (time span from the first to the last part of the batch), because it does not consider the run lengths of the patterns involved. In order to take this issue into account, further research is needed.

References


