5-2016

A horizon decomposition approach for the capacitated lot-sizing problem with setup times

Ioannis FRAGKOS

Zeger DEGRAEVE

Bert DE REYCK
Singapore Management University, bdereyck@smu.edu.sg

INFORMS Journal on Computing
Vol. 28, No. 3, Summer 2016, pp. 465–482
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.2016.0691

A Horizon Decomposition Approach for the Capacitated Lot-Sizing Problem with Setup Times

Ioannis Fragkos
Rotterdam School of Management, Erasmus University Rotterdam, 3062 PA Rotterdam, Netherlands, fragkos@rsm.nl

Zeger Degraeve
Melbourne Business School, University of Melbourne, Carlton VIC 3053, Melbourne, Australia, z.degraeve@mbs.edu

Bert De Reyck
UCL School of Management, University College London, London WC1E 6BT, United Kingdom, b.dereyck@ucl.ac.uk

We introduce *horizon decomposition* in the context of Dantzig-Wolfe decomposition, and apply it to the capacitated lot-sizing problem with setup times. We partition the problem horizon in contiguous overlapping intervals and create subproblems identical to the original problem, but of smaller size. The user has the flexibility to regulate the size of the master problem and the subproblem via two scalar parameters. We investigate empirically which parameter configurations are efficient, and assess their robustness at different problem classes. Our branch-and-price algorithm outperforms state-of-the-art branch-and-cut solvers when tested to a new data set of challenging instances that we generated. Our methodology can be generalized to mathematical programs with a generic constraint structure.

Keywords: algorithms; lot sizing; branch and price
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received August 2014; revised August 2015; accepted December 2015. Published online May 24, 2016.

1. Introduction

Since the seminal work of Dantzig and Wolfe (1960), Dantzig-Wolfe decomposition has been applied successfully to solving linear, integer, and mixed-integer linear programming problems and a variety of applications of increasing complexity (Lübbecke and Desrosiers 2005). The implementation of the Dantzig-Wolfe decomposition principle involves the recognition of a part of the constraint matrix that has block diagonal structure, where each block is associated with a subset of variables. The variables that appear in each block should not appear in other blocks, and if so, the corresponding constraints are treated as "complicating." This explains why most research and practical applications are usually problem specific. In addition, although for certain large-scale problems, branch-and-price algorithms may have superior performance against branch-and-cut software, the range of applications is limited by the block diagonal structure that is in place, and by how exploitable this structure is. The competitive advantage of Dantzig-Wolfe reformulations stems from exploiting these substructures to obtain an improved dual bound. This occurs in cases where the subproblem does not have the integrality property (Geoffrion 1974), which means that its linear relaxation does not have all integral extreme points. The backbone of the most successful applications is usually a specialized algorithm that solves the subproblem efficiently.

In this paper, we introduce a novel Dantzig-Wolfe decomposition scheme that, contrary to the existing ones, does not rely on any exploitable subproblem structure. The methodology and subsequent computational study are in the context of capacitated lot sizing, but the developed approach is applicable to any mixed-integer linear program (MIP). A distinct characteristic of our method is that it regulates the size of the master problem and the subproblem independently, by introducing two scalar parameters. This flexibility suggests that one can experiment with alternative decompositions and address the trade-off between subproblem difficulty and dual bound strength directly. Extensive computational experiments that analyze the efficiency of the horizon decomposition approach indicate that certain decomposition configurations can tackle some particularly hard instances far more efficiently than modern branch-and-cut solvers.

We introduce the main idea in the context of the capacitated lot-sizing problem with setup times (CLST) because it constitutes one of the simplest but yet most computationally challenging problem structures. Trigeiro et al. (1989) introduced the problem and constructed a data set of 540 instances, the hardest of which remained unsolvable until the last decade. Although

today all instances can be solved within a few seconds, several researchers (Süral et al. 2009, Müller et al. 2012, de Araujo et al. 2015) have constructed instances with long horizons, tight capacity constraints, or without setup costs that remain intractable. Further, the multiperiod nature of lot-sizing problems and the complicating structure of the capacity constraints provide an excellent ground to demonstrate the horizon decomposition principle. Based upon this setting, the generalization of our approach comes naturally. Finally, CLST is well studied in the literature and therefore we can benchmark the efficiency of our approach against other techniques, such as valid inequalities, extended formulations, and alternative decomposition schemes. In addition, in some special cases it is possible to establish which approach gives the best bound or to draw correspondences across methodologies.

The principal aim of this work is to illustrate that the application of horizon decomposition to the CLST has at least two important benefits. First, one can exploit the technology of modern solvers in solving subproblems of manipulable size and strength. Since the subproblem size is controlled independently from the size of the master problem, it is possible to find a balance between dual bound quality and subproblem tractability. Second, our computational experiments show that in practice the method shows excellent behavior in perhaps the most challenging class of problems, namely, instances with low ratio of items over periods and tight capacity constraints.

The remainder of this paper is organized as follows. Section 2 gives a brief literature review on column generation methodologies and on CLST-specific research. Section 3 introduces the problem formulation. Section 4 applies horizon decomposition. A comparison and correspondences with other lower bounds are demonstrated. Section 5 describes a branch-and-price algorithm that uses horizon decomposition. Section 6 presents computational experiments and Section 7 presents two ways the horizon decomposition principle can be generalized to generic MIPs. Finally, Section 8 concludes the paper with suggestions for future work.

2. Literature Review

Since the early days of column generation, many authors have used it either as a stand-alone technique to solve large linear programs (Elhallaoui et al. 2005), or as a bounding technique within branch-and-bound algorithms (Degraeve and Jans 2007), a scheme also known as *branch and price*. On the theoretical side, there are works that examine the efficient convergence of column generation and the branching rules used in branch and price. Ben Amor et al. (2006) show that reducing the feasible dual space of the master program leads to faster convergence. Degraeve and Jans (2007) demonstrate how the Dantzig-Wolfe decomposition principle is applied to MIPs with an application to the CLST and Vanderbeck and Savelsbergh (2006) develop a theoretical framework. Vanderbeck (2011) explores the issue of branching in branch and price when the subproblems are identical, and Villeneuve et al. (2005) construct a compact formulation and use the corresponding variables for branching. The reviews of Lübbecke and Desrosiers (2005) and Barnhart et al. (1998) describe plenty of applications and discuss in detail technical issues of column generation and branch and price, respectively.

Lagrange relaxation is a related reformulation that, in theory, gives the same dual bound as column generation. Fisher (2004) gives an overview of Lagrange relaxation and describes early applications. The strong dual bound and the relative speed of Lagrange relaxation have led to the development of efficient exact and heuristic methods. Lagrangian decomposition is a generalization of Lagrange relaxation that yields stronger lower bounds. Guignard and Kim (1987) were the first to introduce it in the context of MIPs that have two sets of constraints. The main idea is to introduce "copy" constraints for the original variables and dualize them in the objective function. Our implementation can be seen as a case of Lagrange decomposition since we also introduce copies of variables and explore the convex hull of the corresponding subproblems. It is more versatile however in that it can be tailored to each instance, it avoids unnecessary variable copying, and performs a systematic reformulation that creates a decomposable structure.

The literature in capacitated lot-sizing problems is vast. In their seminal paper, Wagner and Whitin (1958) introduced the single-item uncapacitated version of the problem and solved it using a dynamic programming recursion. Trigeiro et al. (1989) were the first to examine a multiitem problem with capacity constraints and setup times. They showed experimentally that setup times make the problem harder and developed a Lagrange-based smoothing heuristic whose performance remains competitive up to date. An earlier result by Kleindorfer and Newson (1975) proves that the problem is strongly NP-hard. To obtain an improved lower bound, Eppen and Martin (1987) reformulated the problem with shortest-path variables that describe the convex hull of the single-item uncapacitated polyhedron. Similarly, Barany et al. (1984) describe the same polyhedron using valid inequalities. In more recent advancements, Degraeve and Jans (2007) develop an exact branch-and-price algorithm using a per-item decomposition and Jans and Degraeve (2004) describe a decomposition of the shortest path formulation that leads to an improved lower bound. The most recent work that applies Dantzig-Wolfe decomposition to the CLST is Pimentel et al. (2010). They develop three alternative decompositions and

branch-and-price algorithms and compare their performance. Finally, another stream of research focuses on finding good feasible solutions with heuristics. Süral et al. (2009) develop a Lagrange-based heuristic for a variant of the CLST without setup costs. They used the subproblem solutions to construct incumbents during the subgradient optimization process and obtained small integrality gaps over a set of hard instances. Similarly, Müller et al. (2012) use large-scale neighborhood search combined with strong formulations and report results on new hard instances. Finally, Akartunal and Miller (2012) give insights on which lot-sizing substructures are computationally challenging, and Akartunali et al. (2016) use column generation to generate cuts from a two-period relaxation of CLST. In the latter work, the authors use several distance functions, by which they are able to generate valid inequalities that cut off the linear programming relaxation solution, when this solution has a positive distance from a predefined two-period lot-sizing set.

Our work has contributions in both the Dantzig-Wolfe decomposition and lot-sizing research streams. First, we show how Dantzig-Wolfe decomposition can be applied in a novel way, such that MIPs can be decomposed in subproblems that preserve the structure of the original problem, but are of smaller size. Second, we demonstrate the applicability of this idea in lot sizing and investigate under which conditions it is advantageous against competitive methodologies. Third, we show experimentally that a class of CLST instances, namely, those with tight capacity constraints and small ratios of items over periods, are time consuming to solve with modern branch-and-cut software. We develop a branch-and-price approach based on horizon decomposition and demonstrate its efficiency against competitive approaches. Finally, we demonstrate the extension of our idea to generic MIPs.

3. Problem Description and Formulation

3.1. Original Formulation

The capacitated lot-sizing problem with setup times generalizes the basic single-item uncapacitated lot-sizing problem studied by Wagner and Whitin (1958). Specifically, it models a multi-item setting with one capacity constraint per period and item-specific setup times and production times. It can be used in production planning for determining the production and setup decisions of an MRP system by taking into consideration one bottleneck resource (Pochet and Wolsey 2006). We formulate the problem using the following notation:

Sets

$I = \{1, \dots, n\}$: set of items, indexed by i.
$T = \{1, \dots, m\}$: set of periods, indexed by t.

Parameters

d_{it}: demand of item i in period t, $i \in I$, $t \in T$.
sd_{itk}: sum of demand of item i from period t to period k, $i \in I$, t, $k \in T$: $t \le k$.
hc_{it}: cost of holding inventory for item i from period $t-1$ to period t, $i \in I$, $t \in T$.
sc_{it}: setup cost of item i in period t, $i \in I$, $t \in T$.
vc_{it}: production cost of item i in period t, $i \in I$, $t \in T$.
st_{it}: setup time of item i in period t, $i \in I$, $t \in T$.
vt_{it}: variable production time of item i in period t, $i \in I$, $t \in T$.
cap_t: time capacity in period t, $t \in T$.
M_{it}: big-M quantity, defined as $M_{it} = \min\{sd_{itm}, (cap_t - st_{it})/(vt_{it})\}$, $i \in I$, $t \in T$.

Decision Variables

x_{it}: production quantity of item i in period t, $i \in I$, $t \in T$.
s_{it}: inventory quantity of item i at the beginning of period t, $i \in I$, $t \in T \cup \{m+1\}$.
y_{it}: equals 1 if a setup occurs for item i in period t, 0 otherwise, $i \in I$, $t \in T$.

The mathematical formulation of CLST is then as follows:

$$\min \left\{ \sum_{i \in I} \sum_{t \in T} sc_{it} y_{it} + \sum_{i \in I} \sum_{t \in T} vc_{it} x_{it} + \sum_{i \in I} \sum_{t \in T} hc_{it} s_{it} \right\} \quad (1)$$

$$\text{s.t.} \quad s_{it} + x_{it} = d_{it} + s_{i,t+1} \quad \forall i \in I, \forall t \in T \quad (2)$$

$$x_{it} \le M_{it} y_{it} \quad \forall i \in I, \forall t \in T \quad (3)$$

$$\sum_{i \in I} st_{it} y_{it} + \sum_{i \in I} vt_{it} x_{it} \le cap_t \quad \forall t \in T \quad (4)$$

$$x_{it}, s_{it} \ge 0, s_{i,m+1} = 0, y_{it} \in \{0,1\} \quad \forall i \in I, \forall t \in T. \quad (5)$$

The objective function (1) minimizes the total cost, which consists of the setup cost, the production cost, and the inventory holding cost. To model problems that are infeasible without initial inventory, we allow for initial inventory at a high cost (Vanderbeck 1998). Constraints (2) indicate that demand in each period is covered by initial inventory and by production, and that the remaining quantity is transferred to the next period. Constraints (3) link the setup and production decisions and (4) describe the per-period capacity constraints. Finally, constraints (5) pose nonnegativity and integrality restrictions to the problem variables. We use v_{CLST} to denote the optimal objective value of (1) over constraints (2)–(5) and $\bar{v}_{CLST}$ to denote optimal objective value of its LP relaxation. The next paragraph describes a family of reformulations that allow a generic decomposition scheme for the CLST.

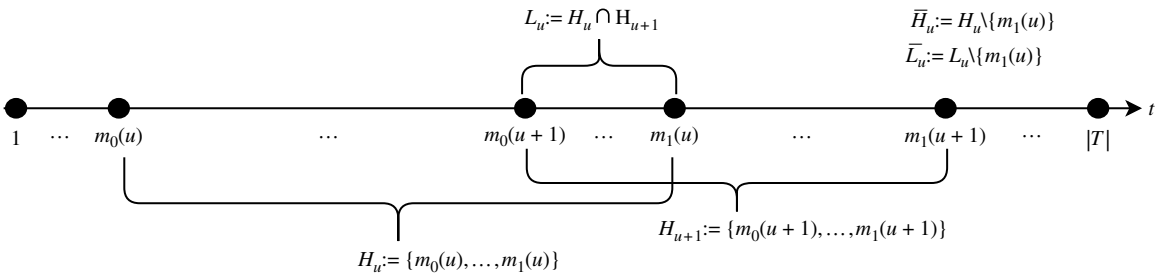

Figure 1 Notation Used in Horizon Covering

Note. Each black bullet indicates a discrete time period $t \in T$.

3.2. Horizon Reformulation

The fundamental idea of horizon decomposition is to reformulate the problem so that it decomposes in subproblems of identical structure but of shorter horizons. Modern MIP solvers can solve small subproblems efficiently, and the fact that small problems do not have the integrality property (Geoffrion 1974) implies that column generation can lead to an improved lower bound. A question that arises naturally in this context is whether defining subproblems over overlapping or nonoverlapping horizons has an impact on the lower bound quality. In our formulation, two consecutive subproblems with nonoverlapping horizons share common inventory variables. Specifically, the ending inventory of the earlier subproblem and the starting inventory of the later subproblem have costs that are adjusted jointly by the dual prices of some master problem constraints. As a result, from a qualitative perspective, subproblems defined over nonoverlapping horizons share information only by the cost of their initial and ending inventory variables; the production and setup variables of each subproblem are disjoint, and this can have a negative impact on the lower bound quality. One way of increasing communication among subproblems is by introducing horizon overlaps, which lead to some setup and production costs to be adjusted jointly; this, in turn, can give an improved lower bound. However, horizon overlaps also introduce additional linking constraints in the master problem, which can cause degeneracy and poor convergence of column generation. Therefore, depending on the size of each instance, zero overlaps might be beneficial because the column generation inhibits better convergence, but might also lead to poor lower bounds, because limited information is shared across subproblems. An important contribution of our computational study is to gain insights on when overlapping horizons can lead to improved performance. We next provide some technical definitions that facilitate the exposition of the horizon reformulation.

We define a *horizon cover* $\mathcal{P}$ as a set whose elements are horizons H of the form $H = \{m_0, m_0+1, \ldots, m_1\} \subseteq T$, with $m_1 > m_0$. Therefore, each horizon consists of a certain number of consecutive periods, starting at m_0 and ending at m_1, and the horizon cover is the union of possibly overlapping horizons. To characterize the horizon cover set, we introduce the following notation (see also Figure 1):

Index set of the horizon cover $\mathcal{P}$: $U := \{1, \ldots, |\mathcal{P}|\}$.
uth horizon: $H_u := \{m_0(u), \ldots, m_1(u)\}$, where $m_0(u), m_1(u) \in T, \forall u \in U$.
uth core horizon: $\bar{H}_u := H_u \backslash \{m_1(u)\}, \forall u \in U$.
Horizon intersection: $L_u := H_u \cap H_{u+1}, \forall u \in U$.
Core horizon intersection: $\bar{L}_u := L_u \backslash \{m_1(u)\}, \forall u \in U$.
Boundary conditions: $m_1(|\mathcal{P}|) = m+1$, and $H_0 = H_{|\mathcal{P}|+1} = \varnothing$.

For convenience, we assume throughout the paper that whenever a set that defines a constraint is empty, then the constraint is not defined. Note that some periods can be common in two or more horizons. The case where a period is common in more than two horizons is not of practical interest in our context. Therefore, we impose the condition $H_{u-1} \cap H_{u+1} = \varnothing$ for each $u \in U$. Finally, we assume that two contiguous horizons H_u and H_{u+1} have at least one common period, which implies that $m_1(u) \in H_{u+1}$, for all $u \in U$.

Next, we define production, setup, and inventory variables for each u-horizon. Note that the last period of each horizon is used to define the inventory variable only:

x^u_{it}: production quantity of item i in period t in horizon H_u, $\forall i \in I$, $\forall t \in \bar{H}_u$, $\forall u \in U$.
s^u_{it}: starting inventory quantity of item i in period t in horizon H_u, $\forall i \in I$, $\forall t \in H_u$, $\forall u \in U$.
y^u_{it}: equals 1 if a setup occurs for item i in period t in horizon H_u, 0 otherwise, $\forall i \in I$, $\forall t \in \bar{H}_u$, $\forall u \in U$.

In addition, let $\alpha_{tu} = 1$, if $t \in H_u \backslash L_{u-1}$, $\alpha_{tu} = 0$ otherwise, for all $t \in H_u$, $u \in U$. Using the above notation, problem (1)–(5) can be reformulated as follows:

$$\min \left\{ \sum_{i \in I} \sum_{u \in U} \sum_{t \in \bar{H}_u} \alpha_{tu} (sc_{it} y^u_{it} + vc_{it} x^u_{it}) + \sum_{i \in I} \sum_{u \in U} \sum_{t \in H_u} \alpha_{tu} hc_{it} s^u_{it} \right\} \tag{6}$$

$$\text{s.t.}\quad s^u_{it}+x^u_{it}=d_{it}+s^u_{i,t+1}\quad \forall i\in I,\forall t\in\bar{H}_u,\forall u\in U \tag{7}$$

$$x^u_{it}\le M_{it}y^u_{it}\quad \forall i\in I,\forall t\in\bar{H}_u,\forall u\in U \tag{8}$$

$$\sum_{i\in I}st_{it}y^u_{it}+\sum_{i\in I}vt_{it}x^u_{it}\le cap_t\quad \forall t\in\bar{H}_u,\forall u\in U \tag{9}$$

$$s^u_{it}=s^{u+1}_{it}\quad \forall i\in I,\forall t\in L_u,\forall u\in U \tag{10}$$

$$x^u_{it}=x^{u+1}_{it}\quad \forall i\in I,\forall t\in\bar{L}_u,\forall u\in U \tag{11}$$

$$y^u_{it}=y^{u+1}_{it}\quad \forall i\in I,\forall t\in\bar{L}_u,\forall u\in U \tag{12}$$

$$x^u_{it}\ge 0,y^u_{it}\in\{0,1\}\quad \forall i\in I,\forall t\in\bar{H}_u,\forall u\in U \tag{13}$$

$$s^u_{it}\ge 0\quad \forall i\in I,\forall t\in H_u,\forall u\in U. \tag{14}$$

Constraints (7)–(9) and (13)–(14) define a CLST over the uth core horizon $\bar{H}_u$. This implies that the corresponding inventory variables s^u_{it} are defined over the uth horizon $H_u=\bar{H}_u\cup\{m_1(u)\}$. Therefore, period $m_1(u)$ is used to associate the ending inventory variables of each CLST defined over a uth core horizon $\bar{H}_u$, exactly as period $m+1$ is used to set the ending inventories to zero in formulation (1)–(5). Constraints (10)–(12) impose that variables indexing the same period in two horizons should attain the same values. Finally, objective function (6) considers the setup, inventory, and production costs of all horizons. Parameter α_{tu} is an indicator used for the appropriate allocation of costs: if a variable is defined in two horizons, then its cost is allocated to the earliest horizon. Like in Lagrange decomposition (Guignard and Kim 1987), it is straightforward to see that the variables indexed within horizon overlaps can be allocated any fraction of the original cost, without loss of generality.

Note that a benefit of the above reformulation is its flexibility. By selecting the parameters $m_0(u)$ and $m_1(u)$ for each $u\in U$, one can regulate the number of subproblems, subproblem length, and periods of overlap. Moreover, the formulation remains valid when no overlap between horizons exists, i.e., when $\bar{H}_u\cap\bar{H}_{u+1}=\varnothing$. In this case $\bar{L}_u=\varnothing$, and there are no linking constraints for the production and setup variables. Finally, the original formulation (1)–(5) can be considered as a special case of (6)–(14), where the horizon cover is a singleton with $m_0=1$ and $m_1=m+1$. The next section describes how the above structure can be used in Dantzig-Wolfe reformulations.

4. Horizon Decomposition

4.1. Initial Formulation

Formulation (6)–(14) decomposes per horizon, with the exclusion of constraints (10)–(12). Let us note with $(\bar{f}_u)$ the subset of a block of constraints (f) that refer to a specific horizon H_u, $u\in U$. Also let $(x,y,s)_u=((x^u_{it},y^u_{it})\colon i\in I,t\in\bar{H}_u;s^u_{it}\colon i\in I,t\in H_u)$. We then define the *single horizon polyhedron* as $\mathscr{W}_u:=\{(x,y,s)_u\mid(\bar{7})_u$–$(\bar{9})_u,(13)_u$–$(\bar{14})_u,s^u_{im_1(u)}\le sd_{im_1(u),m},\forall i\in I\}$ and let $\mathscr{E}_u$ be the set of extreme points of $\mathrm{conv}(\mathscr{W}_u)$, for each $u\in U$. Note that we bound the ending inventory variables with the remaining item demand in order to avoid the use of extreme rays and to tighten the subproblem formulation. Each extreme point $e=(\bar{x},\bar{y},\bar{s})_{ue}\in\mathscr{E}_u$ is associated with the following elements:

c_{ue}: total cost of production, setup, and inventory of horizon H_u according to production plan e, i.e.,

$$\sum_{i\in I}\sum_{t\in\bar{H}_u}\alpha_{tu}(sc_{it}\bar{y}^u_{ite}+vc_{it}\bar{x}^u_{ite})+\sum_{i\in I}\sum_{t\in H_s}\alpha_{tu}hc_{it}\bar{s}^u_{ite};$$

z_{ue}: fraction of production plan e that is used for actual production.

The Dantzig-Wolfe reformulation is then as follows:

$$[\widetilde{\mathbf{DW}}]$$

$$\min\sum_{u\in U}\sum_{e\in\mathscr{E}_u}c_{ue}z_{ue} \tag{15}$$

$$\text{s.t.}\sum_{e\in\mathscr{E}_u}\bar{s}^u_{ite}z_{ue}=\sum_{e\in\mathscr{E}_{u+1}}\bar{s}^{u+1}_{ite}z_{u+1,e}\quad \forall i\in I,\forall t\in L_u,\forall u\in U\backslash\{|\mathscr{P}|\} \tag{16}$$

$$\sum_{e\in\mathscr{E}_u}\bar{x}^u_{ite}z_{ue}=\sum_{e\in\mathscr{E}_{u+1}}\bar{x}^{u+1}_{ite}z_{u+1,e}\quad \forall i\in I,\forall t\in\bar{L}_u,\forall u\in U\backslash\{|\mathscr{P}|\} \tag{17}$$

$$\sum_{e\in\mathscr{E}_u}\bar{y}^u_{ite}z_{ue}=\sum_{e\in\mathscr{E}_{u+1}}\bar{y}^{u+1}_{ite}z_{u+1,e}\quad \forall i\in I,\forall t\in\bar{L}_u,\forall u\in U\backslash\{|\mathscr{P}|\} \tag{18}$$

$$\sum_{e\in\mathscr{E}_u}z_{ue}=1\quad \forall u\in U \tag{19}$$

$$s_{it}=\sum_{e\in\mathscr{E}_u}\bar{s}^u_{ite}z_{ue}\quad \forall i\in I,\forall(t,u)\in H_u\times U\colon\alpha_{tu}=1 \tag{20}$$

$$x_{it}=\sum_{e\in\mathscr{E}_u}\bar{x}^u_{ite}z_{ue}\quad \forall i\in I,\forall(t,u)\in\bar{H}_u\times U\colon\alpha_{tu}=1 \tag{21}$$

$$y_{it}=\sum_{e\in\mathscr{E}_u}\bar{y}^u_{ite}z_{ue}\quad \forall i\in I,\forall(t,u)\in\bar{H}_u\times U\colon\alpha_{tu}=1 \tag{22}$$

$$z_{ue}\ge 0\quad \forall e\in\mathscr{E}_u,\forall u\in U \tag{23}$$

$$s_{it}\ge 0\quad \forall i\in I,\ t\in T \tag{24}$$

$$y_{it}\in\{0,1\},x_{it}\ge 0\quad \forall i\in I,\ t\in T. \tag{25}$$

Formulation $[\widetilde{\mathbf{DW}}]$ is equivalent to the original formulation, in the sense that they both attain the same optimal solution. However, the optimal linear programming relaxation objective of $[\widetilde{\mathbf{DW}}]$ is always at least as large as that of the original formulation, because the subproblems do not have the integrality property

(Geoffrion 1974). Constraints (16)–(18) correspond to (10)–(12) and denote that in any period common to two horizons, the production, setup, and inventory quantities should attain the same value in both horizons. Constraints (19) together with the nonnegativity constraints (23) impose that each decision variable is a fraction of an extreme production plan. Equations (20)–(22) define the variables of the original formulation as convex combinations of extreme production plans. Although the number of variables and constraints is large, there are certain reductions that can be performed, which are described in Section 4.2.

4.2. Model Reductions

[$\widetilde{\mathbf{DW}}$] is a valid reformulation of the CLST. Without loss of generality, constraints (17) and (20)–(21) can be eliminated. The elimination of the latter is straightforward because they only map the solution to the original variable space. To see that (17) is redundant, note that $\bar{x}^u_{ite} = d_{it} + \bar{s}^u_{i,t+1,e} - \bar{s}^u_{ite}$ for each $e \in \mathscr{E}_u$. This implies that $\sum_{e\in\mathscr{E}_u} \bar{x}^u_{ite} z_{ue} = d_{it} + \sum_{e\in\mathscr{E}_u} \bar{s}^u_{i,t+1,e} z_{ue} - \sum_{e\in\mathscr{E}_s} \bar{s}^u_{ite} z_{ue} = d_{it} + \sum_{e\in\mathscr{E}_{u+1}} (\bar{s}^{u+1}_{i,t+1,e} - \bar{s}^{u+1}_{ite}) z_{u+1,e} = \sum_{e\in\mathscr{E}_{u+1}} \bar{x}^{u+1}_{ite} z_{u+1,e}$. We have shown the following result.

Corollary 1. *Constraints*

$$\sum_{e\in\mathscr{E}_{u+1}} \bar{x}^{u+1}_{ite} z_{u+1,e} = \sum_{e\in\mathscr{E}_u} \bar{x}^u_{ite} z_{ue},\ \forall\, i \in I,\ \forall\, t \in \bar{L}_u,\ \forall\, u \in U\backslash\{|\mathscr{P}|\}$$

are redundant.

We denote [**DW**] the model resulting from (15)–(25) with the exclusion of redundant constraints.

Note that one cannot eliminate the setup definition constraints and impose the integrality restrictions on the extreme production plan variables z_{se} (Degraeve and Jans 2007, Vanderbeck and Savelsbergh 2006). A correct reformulation would define, for each extreme point, a binary variable that describes the setup configurations and a continuous variable with the associated production decisions. However, the usability of this reformulation is restricted, because the resulting branch-and-bound tree is unbalanced (Vanderbeck 2011). In our implementation, we branch on the original setup variables by fixing them at the subproblems and by removing the generated columns that do not adhere to the node branching decisions, therefore using (22) only implicitly.

4.3. Strength of the Lower Bound

In this section, we investigate the strength of the lower bound obtained by horizon decomposition. Since an explicit description of the convex hull of CLST is not known, we can compare the lower bound strength with lower bounds obtained by other approaches. The fact that the subproblems do not have the integrality property implies that the lower bound obtained by the LP relaxation of (1)–(5), $\bar{v}_{CLST}$, cannot be better than that obtained by [**DW**], $\bar{v}_{DW}$ (Geoffrion 1974). More interesting is the comparison with the bound obtained when the (l, S) inequalities (Barany et al. 1984, Miller et al. 2000) are appended to the original formulation (1)–(5). If we denote this bound by $\bar{v}_{lS}$, we can state the following proposition.

Proposition 1. *The lower bound $\bar{v}_{DW}$ does not dominate $\bar{v}_{lS}$, or vice versa.*

Proof. Consider an instance with $cap_t \geq \sum_{i\in I}(sd_{itm} + st_{it})$ for each $t \in T$. This condition makes the capacity constraints redundant and the problem decomposes in a series of single-item uncapacitated problems. Since the (l, S) inequalities describe the convex hull of the single-item uncapacitated problems, $\bar{v}_{lS} \geq \bar{v}_{DW}$. Moreover, this inequality can be strict. To see this, consider without loss of generality an instance for which the inequality $s_{ik} + \sum_{t\in\{k,\ldots,l\}\backslash S} x_{it} + \sum_{t\in S} sd_{itl} y_{it} \geq sd_{ikl}$ is binding for some fixed i, k and l such that $k < l$, and the associated part of the optimal solution is $x_{ik} = sd_{ikl}$; $y_{ik} = 1$, with all other variables in $\{k, \ldots, l\}$ being zero. We can then construct a horizon cover with two subproblems, i.e., $S = \{1, 2\}$ and let L be the index set of overlapping periods such that $k \in H_1\backslash L$, and $l \in H_2\backslash L$. Then it follows that $\bar{v}_{lS} > \bar{v}_{DW}$ because the production quantity x_{ik} in subproblem S_1 will never be sd_{ikl}, because $x_{ik} = sd_{ikl}$ does not have the Wagner-Whitin property of optimality (Wagner and Whitin 1958), and therefore is not an extreme point of S_1. Next, consider a single-item instance with binding capacity constraints, and $s_k = 0$ for some period k at an optimal solution. A horizon decomposition with $H_1 = \{1, \ldots, k-1\}$ and $H_2 = \{k, \ldots, m\}$ will deliver an optimal solution of the original problem, so $\bar{v}_{DW} = v_{CLST}$. However, $\bar{v}_{lS} \leq v_{CLST}$ because the (l, S) inequalities do not suffice to describe the convex hull of the capacitated problem. □

We can use similar arguments to show that there is no strict dominance between horizon decomposition and the decomposition considered by Jans and Degraeve (2004) and de Araujo et al. (2015). Note that the lower bound of the latter is at least as strong at $\bar{v}_{lS}$, since they apply decomposition to the network reformulation of Eppen and Martin (1987), which describes the same convex polyhedron as the (l, S) inequalities. Finally, $\bar{v}_{DW}$ is at least as strong as the lower bound obtained by the per period decomposition of Pimentel et al. (2010), since their per period decomposition formulation is a special case of a horizon decomposition, where each horizon defines a single-period subproblem for each period.

5. A Branch-and-Price Algorithm

Although the relaxation of [**DW**] can give a strong lower bound in most problems, the setup variables,

defined by (22), can be fractional, and therefore a branch-and-price approach is necessary. We first employ a simple heuristic that constructs good quality feasible solutions. Then, we do column generation to find a lower bound for the MIP optimal solution. Finally, we embed column generation in a branch-and-bound scheme, thereby developing a branch-and-price algorithm. This section describes the most important components of our algorithm and outlines the most crucial implementation decisions.

5.1. Initialization

The column generation procedure has finite convergence and gives a lower bound only if the restricted master problem is initialized so that it has a feasible solution (Lübbecke and Desrosiers 2005). The most common approach to initialize the master problem is to introduce columns with high cost that render it feasible. However, this might result in a large number of iterations, thereby reducing computational efficiency (Vanderbeck 2005). To tackle this issue, we employ the lot elimination heuristic (LEH) utilized by Degraeve and Jans (2007) on top of introducing high cost columns. LEH starts by fixing all setup variables to 1 and progressively eliminates them using some priority rules. LEH terminates when all setup variables are considered for elimination. Every time LEH finds an improved solution, we add it as columns to the restricted master problem. These columns, in general, do not correspond to subproblem extreme points, but provide a good family of points to *warm-start* the column generation process. In addition, LEH outputs an initial upper bound, which is used in later stages of column generation. Algorithm 1 shows the design of the LEH procedure.

Algorithm 1 (Lot elimination heuristic)

Input: Problem Data
Output: Feasible solution
$[v^{UB}; y^f_{it} \in \{0,1\}, x^f_{it} \geq 0, \forall\, i \in I, t \in T]$

$\bar{v} \leftarrow$ {Optimal solution of (1)–(5): $y_{it} = 1,\ \forall\, i \in i,\ t \in T\}$; $y^f_{it} \leftarrow 1,\ \forall\, i \in I,\ t \in T$
for $t^* \in \{m, \ldots, 1\}$ **do**
  ▷ Start from last period, try to eliminate expensive setups
  $I_s \leftarrow \{i_1, \ldots, i_n\}$: $sc_{i_1,t^*} \geq sc_{i_2,t^*}, \ldots, \geq sc_{i_n,t^*}$
  ▷ Sort items in descending setup costs
  for $i^* \in I_s$ **do**
    $y^f_{i^*t^*} \leftarrow 0$
    $\bar{v}, \bar{x} \leftarrow$ {Optimal solution of (1)–(5): $y_{it} = y^f_{it},\ \forall\, i \in I,\ t \in T\}$
    if $\bar{v} < v^{UB}$ **then**
      $v^{UB} \leftarrow \bar{v}$; $x^f_{it} \leftarrow \bar{x}_{it}, \forall\, i \in i,\ t \in T$
      ▷ Store improved solution
      AppendToMaster(y^f_{it}, x^f_{it})
    else
      $y^f_{i^*t^*} \leftarrow 1$
      ▷ No improvement, keep setup open
    end if
  end for
end for
return $v^{UB}; y^f_{it}, x^f_{it}, \forall\, i \in I, t \in T$

5.2. Hybrid Column Generation and Stabilization

5.2.1. Subproblem Formulation. After initializing the restricted master program, we start generating columns. Specifically, from each subproblem we add the column that has the minimum reduced cost. The problem of finding the minimum reduced cost can be formulated as a CLST defined over each subproblem horizon. We denote by us_{itu}, uy_{itu}, and dc_u the dual values of (16), (18), and (19), respectively, and define the indicator variable

$$\delta_{tu} = \begin{cases} 1 & \text{if } t \in L_{u-1}, \\ -1 & \text{if } t \in L_u, \\ 0 & \text{else.} \end{cases}$$

The subproblem is then formulated as follows:

$$[\mathbf{SP}_u] \quad \text{minimize} \quad v_u = \sum_{i\in I}\sum_{t\in \bar{H}_u} (\alpha_{tu} sc_{it} + \delta_{tu} uy_{itu}) y^u_{it} + \sum_{i\in I}\sum_{t\in \bar{H}_u} \alpha_{tu} vc_{it} x^u_{it} + \sum_{i\in I}\sum_{t\in H_u} (\alpha_{tu} hc_{it} + \delta_{tu} us_{itu}) s^u_{it} - dc_u \tag{26}$$

s.t.

$$s^u_{it} + x^u_{it} = d_{it} + s^u_{i,t+1} \quad \forall\, i \in I,\ \forall\, t \in \bar{H}_u \tag{27}$$

$$x^u_{it} \leq M_{it} y^u_{it} \quad \forall\, i \in I,\ \forall\, t \in \bar{H}_u \tag{28}$$

$$\sum_{i\in I} st_{it} y^u_{it} + \sum_{i\in I} vt_{it} x^u_{it} \leq cap_t \quad \forall\, t \in \bar{H}_u \tag{29}$$

$$x^u_{it}, s^u_{it} \geq 0, y^u_{it} \in \{0,1\} \quad \forall\, i \in I,\ \forall\, t \in \bar{H}_u \tag{30}$$

$$0 \leq s^u_{i,m_1(u)} \leq sd_{im_1(u),m} \quad \forall\, i \in I. \tag{31}$$

Although $[\mathbf{SP}_u]$ is a CLST itself, it has smaller dimension than the original CLST (1)–(5) and it is usually easier to solve efficiently. Despite the fact that a smaller problem dimension does not necessarily imply increased efficiency, there are two arguments that justify this claim in the present context. First, given that the problem structure is the same, instances of small dimension will, on average, be solved to optimality faster than larger ones. Second, an early result by Manne (1958) implies that when the number of items is large compared to the number of periods, the single-item uncapacitated lot-sizing convex hull relaxation of CLST gives an optimal solution that is a

good approximation of the problem with integrality constraints, in the sense that the number of fractional variables that should be binary is limited. The latter convex hull is described by the (l, S) inequalities of Barany et al. (1984). Since most modern solvers are able to add the violated (l, S) inequalities as cutting planes (Belvaux and Wolsey 2000), problems of short periods have tight LP relaxations and can be solved efficiently. These observations are confirmed by our computational experiments, where subproblems were solved efficiently by a modern MIP solver.

5.2.2. Column Generation. When the optimal objective function value v_u is negative, we append the corresponding optimal solution vector as a column to the restricted master problem [**DW**]. Next, we resolve [**DW**] and use the resulting set of optimal dual values to resolve subproblems [$\mathbf{SP}_u$]. This procedure terminates when no columns price out, i.e., when $\sum_{u\in U} \min(v_u, 0) = 0$. It is worth noting that a valid lower bound on the original problem objective value is at hand throughout column generation. If v^r_{RMP} is the optimal objective value of the restricted master problem at iteration r, then a valid lower bound is $v^r_{LB} = v^r_{RMP} + \sum_{u\in U} \min(v_u, 0)$.

5.2.3. Stabilization and Algorithmic Refinements. It has been observed by many researchers that the primal solutions of the restricted master problem are usually degenerate (Vanderbeck 2005, Lübbecke and Desrosiers 2005, du Merle et al. 1999). This degeneracy harms the efficiency of column generation: it implies that the dual restricted master problem has multiple optimal solutions and therefore the dual optimal solution at hand might not be an accurate representation of the optimal dual space. If a dual optimal solution of bad quality is used to price out columns in the subproblems, then the generated columns may not be used in the optimal solution of the subsequent restricted master problem. In this case, column generation takes a degenerate step. This phenomenon has severe impact on the algorithmic performance, and it is usually magnified as the final optimal solution is approached, thereby called the *tailing-off* effect (Vanderbeck 2005).

We employ several techniques to stabilize column generation. During early iterations, we use a hybrid column generation-Lagrange relaxation scheme, similar to those described in Degraeve and Peeters (2003). More specifically, after using the dual values of the restricted master to price out new columns, we do not add the new columns to the master immediately but generate a new set of dual values via subgradient optimization (Fisher 2004). This updating process is deemed to lead to better quality dual prices, and it has the added benefit that no LP solution is required. It is called whenever column generation takes a degenerate step, i.e., when the optimal master objective remains the same in two consecutive iterations. We also adopt a two-phase approach, using both approximate and exact solutions. During phase I, we restrict the dual space of the restricted master program [**DW**] by introducing artificial variables on the primal space, as described in du Merle et al. (1999). This technique reduces the number of degenerate iterations via reducing the feasible dual space. In addition, during the early stages of column generation the aim is to generate columns that describe progressively more accurate inner representations of the primal space of [**DW**]. Toward this end, we solve the subproblems to feasibility, and we also append all feasible solutions that price out. Throughout phase I, valid lower bounds are calculated using the subproblem lower bounds: $v^r_{LB} = v^r_{RMP} + \sum_{u\in U} \min(\bar{v}_u, 0)$ at iteration r, where $\bar{v}_u$ is a lower bound of [$\mathbf{SP}_u$]. When $|v^r_{RMP} - v^r_{LB}| \le \epsilon$ for some given $\epsilon > 0$, we switch to phase II, where we apply standard column generation. In our implementation, $\epsilon = 0.05\%$ was found to strike a good balance of time allocated to fast approximate pricing and to exact pricing. Algorithm 2 outlines the steps of the hybrid column generation algorithm.

Algorithm 2 (Hybrid column generation)

Input: Problem Data, feasible solution
$[v^{UB}; y^f_{it} \in \{0, 1\}, x^f_{it} \ge 0, \forall i \in I, t \in T]$
Output: If $v_{LB} < v^{UB}$: Node relaxation
$[v_{LB}; y^r_{it} \in [0, 1], x^r_{it} \ge 0, \forall i \in I, t \in T]$,
otherwise v_{LB}

1: InitializeMaster
▷ Adds stabilization variables and feasible solutions from LEH
2: $v^0_M \leftarrow v^{UB}$; $c \leftarrow 1$; solutionMode = *Feasiblity*
3: **loop**
4: (v^c_M, duals) $\leftarrow$ SolveMaster
▷ Store the objective value and the dual prices
5: **if** $|v^{c-1}_M - v^c_M| > \epsilon$ **then**
▷ If step is not degenerate, solve subproblems
6: (pricedOut, v_{LB}, columns)
$\leftarrow$ SolveSubproblems(duals, solutionMode)
7: **else** ▷ Otherwise, solve Lagrange relaxation, store multiple columns
8: (pricedOut, v_{LB}, columns)
$\leftarrow$ LagrangeRelaxation(duals, solutionMode)
9: **end if**
10: **if** $v_{LB} > v^{UB}$ **then**
▷ Exit, the node will be pruned
11: **return** v_{LB}
12: **end if**
13: **if** pricedOut or $|v^c_M - v_{LB}| < \epsilon$ **then**
▷ Because of stabilization, $v_{LB} > v^c_M$ is possible

14: **if** solutionMode= *Optimality* **then**
15: **return** v_{LB}; y^r_{it}, x^r_{it} calculated from (21),(22)
16: **else**
17: RemoveStabilizationFromMaster
▷ After this step $v^c_M > v^{c-1}_M$ is likely
18: solutionMode= *Optimality*
19: **end if**
20: **end if**
21: UpdateMaster(columns)
▷ Add new columns to the master problem
22: $c \leftarrow c+1$
23: **end loop**

5.3. Branching

We branch on the original setup variables using (18) implicitly. Specifically, we impose the branching restrictions at the subproblem level, and we remove existing columns that do not adhere to the branching configuration of each node. We branch on the earliest fractional variable, which is an efficient selection rule for most lot-sizing problems (Van Vyve and Wolsey 2006). Branching occurs only when the node lower bound is lower than the incumbent value, otherwise we terminate column generation prematurely and prune the node. Finally, we adopt a best-first approach, i.e., we explore the node with the weakest lower bound first. This strategy is beneficial when the time spent at each node is large, because it minimizes the number of nodes explored in the branch-and-bound tree. In our computational experiments, we select horizon decompositions that deliver very strong lower bounds but the solution time of each node is rather large. Therefore, the combination of best-first search and tight lower bounds constitutes an efficient enumeration procedure.

The algorithm consists of three main parts: branching, column generation, and pruning. Whenever a node lower bound is lower than the incumbent upper bound v_{UB}, we branch and apply column generation to its children. If during column generation we calculate a lower bound greater than v_{UB}, we prune the node and delete the generated columns. This is in contrast with columns that do not adhere to branching decisions: we keep the latter in a pool and add them back when solving nodes in which they are feasible.

5.4. Heuristic Solutions

After employing the LEH procedure that gives a set of progressively better feasible solutions, we exploit the root node optimal solution to construct heuristic solutions using the concept of relaxation induced neighborhoods (RINS) of Danna et al. (2005). RINS is a versatile procedure that can be embedded easily in our scheme, and when the lower bound is strong, it tends to provide good quality feasible solutions. Specifically, we formulate the problem on the original space (1)–(5), select some $0.5 < l < 1$ and set $y_{it} = 1$ if $\bar{y}_{it} > l$, $y_{it} = 0$ if $\bar{y}_{it} < 1 - l$ and $y_{it} \in \{0, 1\}$ if $1 - l \le \bar{y}_{it} \le l$, where $(\bar{y}_{it})_{i \in I, t \in T}$ are the fractional setup variables obtained by column generation. We search aggressively for a feasible solution for 100 nodes and if we find one, we update the incumbent. This is an efficient strategy, but it can be time consuming if it is applied at every node. To account for this, we use it every 10 nodes, and employ a simple rounding heuristic at every other node. The latter rounds the fractional setup variables to the closest integer value and solves the resulting extended network flow problem in the continuous variables. It was observed that a strong lower bound at the root node usually leads to a high quality incumbent solution. This is in line with the theory developed in Larsson and Patriksson (2006) that argues that heuristic solutions constructed by near-optimal Lagrange relaxations are also near optimal. Algorithm 3 outlines the details of the RINS procedure.

Algorithm 3 (Relaxation-induced neighbourhood heuristic (Danna et al. 2005))

Input: Problem Data, feasible solution
$[v^{UB}; y^f_{it} \in \{0, 1\}, x^f_{it} \ge 0, \forall i \in I, t \in T]$,
fractional solution
$[v^r; y^r_{it} \in [0, 1], x^r_{it} \ge 0, \forall i \in I, t \in T]$
Output: Feasible solution
$[v^{UB}; y^f_{it} \in \{0, 1\}, x^f_{it} \ge 0, \forall i \in I, t \in T]$
for $(t, i) \in T \times I$ **do**
if $y^r_{it} \in [0, l]$ and $y^f_{it} = 0$ **then**
$\bar{y}_{it} \leftarrow 0$
else if $y^r_{it} \in [1 - l, 1]$ and $y^f_{it} = 1$ **then**
$\bar{y}_{it} \leftarrow 1$
else
$\bar{y}_{it} \in \{0, 1\}$
end if
end for
$v^{UB}, y^f, x^f \leftarrow$ {Incumbent solution of (1)–(5): $y_{it} = \bar{y}_{it}, \forall i \in I, t \in T$; objective value $\ge v^r$}
return v^{UB}; y^f_{it}, x^f_{it}, $\forall i \in I, t \in T$

6. Computational Experiments

In this section, we aim to shed light on four aspects. First, we investigate the trade-off between solution quality and CPU (central processing unit) time by exploring the efficiency of various combinations of subproblem sizes and horizon overlaps. To this end, we perform a full factorial experiment that delivers empirical insights on which configurations are efficient for which classes of problems. Second, we compare the strength of the lower bound obtained by a horizon decomposition to that of other approaches. Third, we devise a heuristic implementation based upon a horizon decomposition and show that it achieves competitive upper and lower bounds when compared to the best heuristic

approaches found in the literature. Finally, we benchmark a branch-and-price algorithm that implements horizon decomposition against a recent branch-and-price algorithm and a state-of-the-art branch-and-cut solver. All formulations were coded in C++ and the mixed-integer and linear programs were solved with CPLEX v12.6. We use a common subproblem size and horizon overlap, i.e., $H_u = H$ and $L_u = L$ for all $u \in U$ with the only possible exception the last subproblem, which consists of the remaining periods to the end of the horizon. Experiments were run on a Linux workstation running a single processor Intel® Xeon® X5675 @ 3.07 GHz with 96 GB memory. To devise a fairer comparison with alternative implementations, we report the relative speed of our machine, according to Standard Performance Evaluation Corporation (SPEC; http://www.spec.org). Summary tables, detailed computational results, and all data instances can be found in the online supplement (available as supplemental material at http://dx.doi.org/10.1287/ijoc.2016.0691).

6.1. Subproblem Length and Overlap

The usefulness of a horizon decomposition depends heavily on the subproblem size and on the horizon overlap. Long-horizon subproblems have the potential to lead to an improved lower bound, but it may be time consuming to solve these to optimality. Likewise, large horizon overlaps can also lead to improved lower bounds, but render the master programs degenerate and amplify the tailing-off effect (Vanderbeck 2005). We assess which configurations of horizon decompositions are efficient in solving challenging CLST problems. The criterion used to assess efficiency is the integrality gap, and time efficiency is measured by average CPU time.

6.1.1. Data Instances. Tuning and testing the algorithm in separate data sets is necessary to avoid any favorable bias during testing. Since the main focus of the paper is problems with few items and long horizons, we generated instances with two, six, and 10 items and 15, 30, and 60 periods, respectively, based on problems G30 and G30b from Trigeiro et al. (1989), which have six items and 15 periods. First, we created instances with 30 and 60 periods by replicating the demand of each item, and made the capacity constraints harder, so that the average lot-for-lot capacity utilization was about 120%. Problems with high capacity utilization are usually challenging to solve in practice, and therefore constitute a good test bed for our approach. Using this utilization level we generated new instances with two, six, and 10 items, and 15, 30, and 60 periods. Two new instances were generated for problems with 15 periods and six items, a total of four, together with G30 and G30b, and four instances for problems with 30 and 60 periods and two or 10 items. In total, 36 instances, four for each of the nine (item, period) combinations, were utilized and 684 runs were performed. We imposed a time limit of 20,000 seconds after which we kept the best lower bound if column generation was not completed.

6.1.2. Subproblem Length and Overlap. The first round of experiments aimed at identifying the influence of subproblem length and horizon overlap on the integrality gap and on CPU time. To this end, Figures 2(a) and 2(b) show the average integrality gap, calculated using the best upper bound found and the root node lower bound in all trials for each instance, and the average CPU time, respectively.

Some useful preliminary insights can be drawn from these figures. With respect to the integrality gaps, both large subproblems and more periods of overlap improve the solution quality. However, the three-period overlap configurations deliver larger gaps when compared to configurations with one or two periods of overlap. This happens because using a three-period overlap renders the restricted master programs

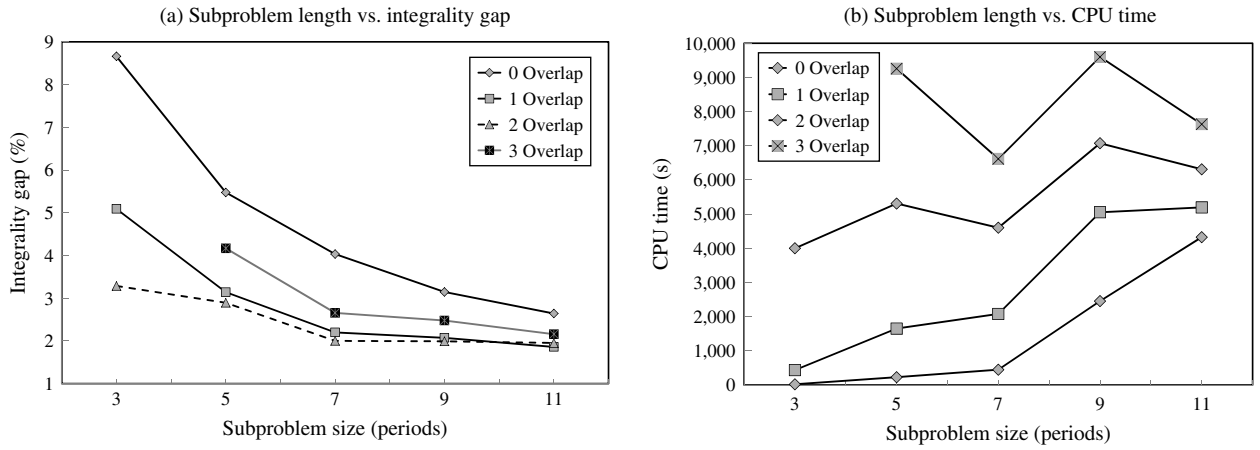

Figure 2 Sensitivity Analysis for Subproblem Size and Overlap Length

Note. Each point denotes the average measure obtained from 90 instances.

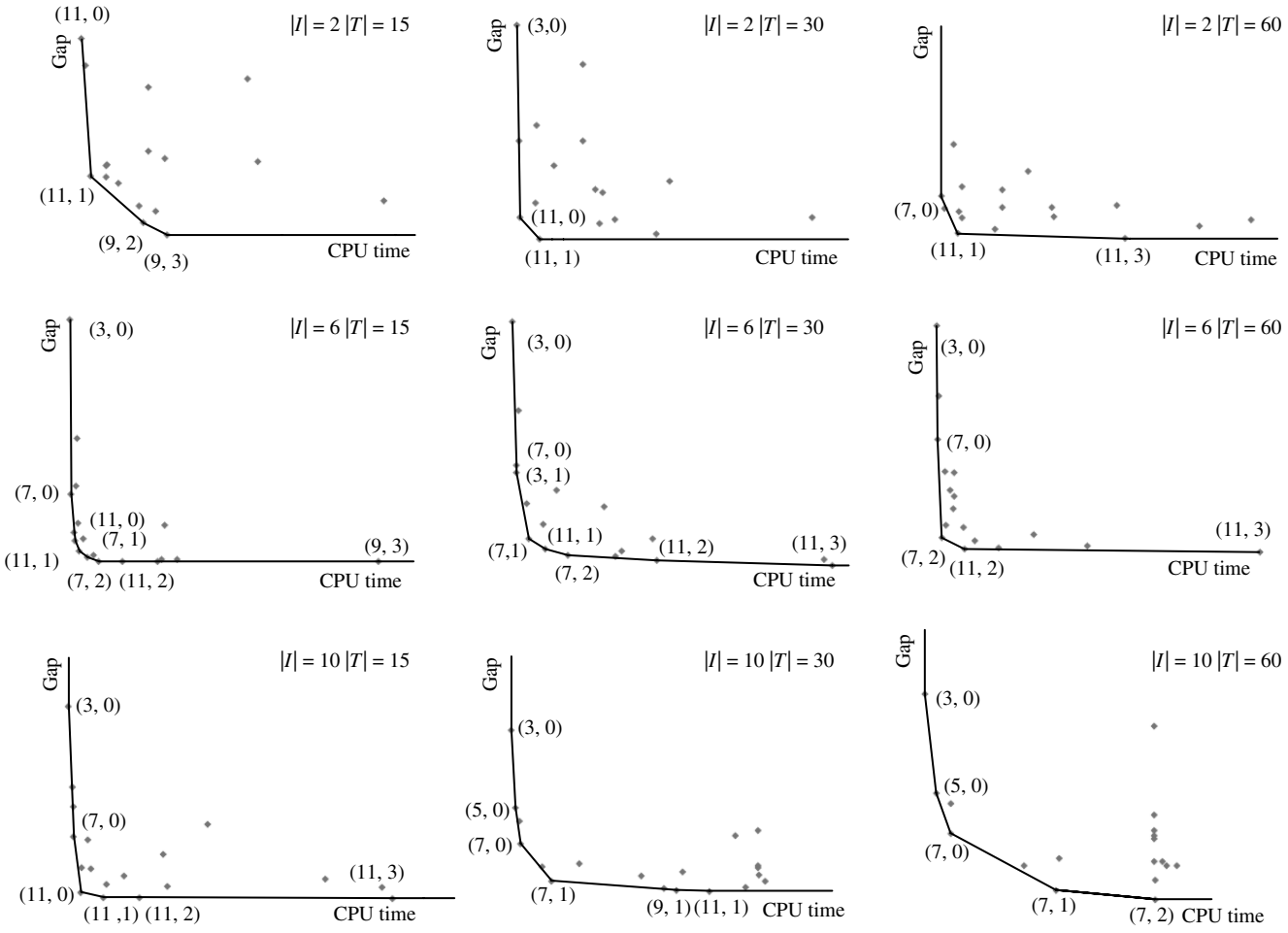

Figure 3 Pareto-Optimal Subproblem Size and Overlap Configurations for Various Item-Period Combinations

Note. The horizontal axis denotes the CPU time and the vertical axis the integrality gap.

degenerate, and therefore more simplex pivots are necessary to solve them to optimality, and also a larger number of column generation iterations. As a result, column generation may fail to terminate before the imposed time limit and the intermediate lower bound can be weaker than that obtained with fewer periods of overlap. The impact of subproblem size seems higher when no overlap exists and is smaller with two or three periods of overlap. Interestingly, one period of overlap leads to drastic reduction of integrality gaps, whereas a second period of overlap offers a significant improvement only when the subproblem size is small (three periods). In other cases, the gap improvement obtained by a second period of overlap is marginal. This leads to the conclusion that one period of overlap with a medium subproblem size, such as seven periods, constitutes a good configuration. Considering CPU times, there is an evident interaction between subproblem size and overlap length, which is revealed in configurations with two or three periods of overlap. Specifically, larger overlaps and subproblems lead to higher CPU times in general, but there are cases where small subproblems combined with large overlaps lead to poor column generation convergence and thereby high CPU times. This is the case for five period subproblems combined with two or three periods of overlap. On the one hand, larger subproblems imply fewer linking constraints for a given overlap and therefore better convergence, but on the other hand it may be time consuming to solve them optimally. This evidence, combined with the marginal gap increase from the inclusion of a third period of overlap, suggests that a third overlapping period may not lead to an efficient computation scheme.

We also investigated which horizon configuration achieves the best performance in which instances. To this end, Table A.1 (in the online supplement) and Figure 3 display a breakdown of the configurations that deliver the best performance in each instance category, characterized by the number of items and the number of periods.

Table A.1 in the online supplement, shows that nonoverlapping horizons induce better convergence behavior of column generation and therefore lead to

faster termination. To perform a more refined analysis, we consider the trade-off between the integrality gap and CPU time for various combinations of items and periods. We call a point *Pareto optimal* if there exists no convex combination of other points that has both a smaller gap and smaller CPU time. Figure 3 graphs the integrality gap and CPU time of each (length, overlap) configuration for various instance families, categorized upon their number of items and time periods. Moreover, it displays the (length, overlap) configurations that are Pareto optimal for each family of instances. In terms of integrality gaps, it is evident that large subproblem horizons achieve the best performance in all cases, with the only exception the bottom right graph, that refers to instances with 10 items and 60 periods, and a medium-sized subproblem delivered the best average gap. We also observe that the Pareto-optimal configurations of Figure 3 are relatively insensitive to the number of items or the number of periods. For all instance families, the leftmost part of the lower envelope is very steep, which means that there are certain configurations that yield significant gains in gaps, with a minor increase of CPU time. This is the case when one overlapping period is introduced, as, for example, in families ($|I| = 2$, $|T| = 15$) and ($|I| = 6$, $|T| = 30$), or a larger subpbroblem is selected, as in families ($|I| = 10$, $|T| = 15$) or ($|I| = 10$, $|T| = 60$). In addition, the rightmost part of most graphs is very also steep, implying that improving the integrality gap after a certain threshold requires a lot more CPU time. This is an important observation, especially for instances with 10 items, in which the CPU time of configurations with large overlaps or large subproblems becomes disproportionately large, and the gap improvement is often marginal.

An interesting question is to which extent our findings generalize to instances with a larger number of items and periods. To shed light on this, we note that configurations with three periods of overlap are not Pareto optimal for families with 10 items and 30 or 60 periods. In addition, many configurations with three-period overlaps took as much as 20,000 seconds of CPU time to converge when applied to instances with 10 items, and some did not converge even then, regardless of the selected subproblem size. This is because of the large number of linking constraints that are introduced in the master problem and degrade the convergence of column generation. Therefore, a computationally efficient approach would not employ three or more periods of overlap to instances with 10 or more items. In fact, Figure 2(a) shows that a three-period overlap does not generate the best integrality gaps, exactly because column generation was not able to terminate, even after 20,000 seconds of CPU time, and returned the best lower bound at hand. With respect to the subproblem size, we found that subproblems of 10 items and 20 or more periods could be time consuming to solve to optimality repeatedly within column generation. Thus, in our computational experiments, we select subproblems of at most 20 periods. We should note that, instances with very long horizons and a small number of items could have other efficient configurations that were not revealed by our computational experiments. For example, an instance with 300 periods and four items could be solved efficiently with a subproblem size of 40 periods, if the MIP solver is able to solve these large subproblems efficiently. Finally, other factors beyond the number of items and periods contribute to whether a specific decomposition configuration delivers strong or weak lower bounds. For example, an instance in which all items have zero starting inventory in periods $t_1, t_2, \ldots, t_k$ in an optimal solution, can be solved optimally if we select subproblems with horizons $\{1, \ldots, t_1 - 1\}, \{t_1, \ldots, t_2 - 1\}, \ldots, \{t_k, \ldots, m\}$, because extreme points of these subproblems can reconstruct the optimal solution.

Despite the aforementioned limitations, the computational study validates the importance of overlapping horizons and of large size subproblems. We use qualitative insights from this experiment to select appropriate horizon configurations in our subsequent experiments. Specifically, it is desirable to select Pareto-optimal configurations that lie on the lower left part of the efficient frontier, because they strike a good balance between CPU time and integrality gap. In practice, for hard problems we might want to sacrifice CPU time in order to improve the lower bound quality. Thus, when the number of items is small, we utilize horizon covers with large subproblems and overlaps. For a medium number of items, such as six items, it is preferable to select large subproblems and one or two periods of overlap. For problems with a larger number of items, medium subproblems and one period of overlap seem to constitute a good choice, unless the problem horizon is short, in which case it might also be efficient to select large subproblems. Finally, we note that for problems with more than 15 items, introducing one period of overlap can improve the integrality gap, but it also increases the CPU time disproportionally. For such problems, we select configurations without any overlap.

6.2. Lower Bounds

In this section, we compare the lower bounds obtained by horizon decomposition to that obtained by other approaches.

6.2.1. Trigeiro Instances. We use the seven instances of Trigeiro et al. (1989) that have been used by several other authors to demonstrate the strength of the lower bounds generated by horizon decomposition. To select a horizon configuration that delivers competitive lower bounds, we devise selection rules based on the conclusions of Section 6.1. To this end, we utilize single-period

overlaps for instances with six and 12 items, as Figure 3 suggests they are efficient, and no overlaps for instances with 24 items, since overlaps would introduce a large number of linking constraints that might lead to slow convergence. Since these problems have relatively short horizons, and instances with up to 15 periods are easy to solve, we select $|H| = \lceil T/2 \rceil$ as subproblem horizons. Alternatively, one could select from the configurations displayed in Figure 3. However, even simple selection policies have the potential to lead to competitive lower bounds. It is worth noting that $|H| = \lceil T/2 \rceil$ might not be a good selection policy for instances with longer horizons because the resulting subproblems might be hard to solve. Table A.2 in the online supplement compares the lower bound obtained by horizon (HD), item (DJ) (Degraeve and Jans 2007) and period decompositions (PD) (de Araujo et al. 2015, Jans and Degraeve 2004), and the approximate extended formulation (AEF) approach of Van Vyve and Wolsey (2006).

The comparison of lower bounds confirms the conclusions of our previous experiments. Specifically, horizon decomposition gives excellent lower bounds for problems with a relatively small number of items. For instance, for G30 in particular we are able to close the gap and obtain an integral solution using horizon decomposition. To the best of our knowledge, the lower bounds for five out of seven instances are the best known in the literature, and for the remaining two instances AEF obtains a better bound. However, any conclusions based on seven instances might be of limited validity. Therefore, the next paragraph considers a larger set of instances, namely, the challenging data set of Süral et al. (2009).

6.2.2. Süral. Süral et al. (2009) constructed a new set of challenging CLST instances by modifying the Trigeiro et al. (1989) test problems. Specifically, they used 20 problems with 12 and 24 items and 15 and 30 periods, and constructed new instances with 10 and 15 periods by truncating the horizons of the original problems. Further, they set the setup costs to zero for all problems and created two versions of each instance: a *homogeneous* version, in which holding costs are equal to 1, and a *heterogeneous* version, that uses the original holding costs. In total, 50 homogeneous and 50 heterogeneous instances were created. Süral et al. (2009) observed that, perhaps surprisingly, problems without setup costs but with setup times seem to be a lot more challenging to solve compared to their counterparts with positive setup costs. Specifically, they show that the algorithm of Trigeiro et al. (1989) delivers an average gap of 0.97% for problems with both setup times and setup costs, and a gap of 33.86% on problems with setup times but without setup costs. Further, they explain that much of this gap is due to poor lower bounds, and not so much due to bad feasible solutions. In their paper, Süral et al. (2009) obtain a lower bound by applying Lagrange relaxation to an extended formulation of the problem. The study of Müller et al. (2012) also uses these instances to investigate the performance of a large-scale neighborhood search heuristic. Although their focus is on upper bounds, they are able to obtain good quality lower bounds by iteratively feeding incumbent solutions to CPLEX (v12.1) and letting it solve the root node, exploiting that improved incumbents may lead to an improved lower bound during presolve. This is possible because some reduced cost fixing operations of the presolve phase make use of the structure of the incumbent. Table A.3 in the online supplement shows the CPU time and integrality gap for the horizon decomposition (HD), the neighborhood search heuristic of Müller et al. (2012) (ALNS), and the Lagrange-relaxation-based heuristic of Süral et al. (2009) (SDW). The integrality gap is measured using the best upper bound found by all approaches, and therefore constitutes a measure of the lower bound quality.

A first conclusion is that although HD requires more CPU time, the lower bound quality it delivers is superior to that obtained by SDW and ALNS. In particular, the average gap of HD is 33% and 50% less than that of SDW and ALNS, respectively, whereas this difference is amplified for problems with 10 or 15 periods. Because our implementation uses 150 seconds of CPU time, we need to strike a fine balance between the obtained lower bound quality and the CPU time used. To this end, Figure 3 shows that increasing the subproblem size while maintaining a zero overlap leads to improved lower bounds without consuming too much CPU time, for instances with 10 items. This is contrary to introducing a single period of overlap, in which case the CPU time increases considerably. For this reason, we have decided to select configurations with zero overlap and relatively large subproblems with $|H| = T/2$.

6.2.3. Müller. The study by Müller et al. (2012) extends the instances of Süral et al. (2009) to problems with longer horizons. In particular, the authors replicate the horizon of each of the original instances that Süral et al. (2009) used and construct (i) instances of 30 and 45 periods, by replicating the demand of the original 15 period instances, and (ii) instances of 60 and 90 periods, by replicating the demand of the original 30 period instances, for both homogeneous and heterogeneous cases. In total, they construct 80 more instances. Table A.4 in the online supplement reports the average integrality gap calculated by taking into account the best upper bound found by both approaches when the time limit for horizon decomposition is 300 (HD300) and 600 (HD600) seconds, respectively.

Table A.4 in the online supplement shows that HD attains systematically lower bounds of better quality, even when the number of periods is large, the only

exception being heterogeneous instances with 24 items and 30, 60, or 90 periods. A more general conclusion from the study of lower bounds is that some horizon decompositions are able to obtain very competitive lower bounds, even if the selection of subproblem size and overlap is made based on simple qualitative rules.

6.3. A Heuristic Implementation

In some production planning environments, it is useful to employ heuristics that find solutions of guaranteed quality in a short amount of time. In this section, we implement a heuristic version of our approach and compare it to the results of Müller et al. (2012), whose approach is the most competitive in the CLST literature. In our implementation, we stop the column generation after 300 or 600 seconds and apply the relaxation induced neighborhood heuristic, which then runs for at most 50 seconds. When column generation is not complete at the root node, we use the best lower bound obtained by Lagrange relaxation. As our objective is to generate good lower and upper bounds within a tight time limit, we do not use overlapping horizons in our heuristic implementation. We also report results on instances with 30 or more periods, since on problems with smaller horizons, both approaches produce results of similar quality. For instances with 30 periods, we use two subproblems of size 20 and 10 periods, respectively. For all other instances, we use a subproblem length of 30 periods, with only a possible exception of the last subproblem, which accommodates the remaining periods. Table A.5 in the online supplement shows the integrality gaps obtained by ALNS and horizon decomposition after 300 (HD300) and 600 (HD600) seconds, respectively. The integrality gap reported for each method is calculated using the best upper and lower bound of which that method returned and is used as a measure of overall performance. To indicate clearly the upper bound quality, Table A.6 in the online supplement reports the integrality gap calculated when the best lower bound is used, thereby providing a measure that directly assesses the upper bound quality.

The comparison suggests that HD delivers overall better integrality gaps, both for homogeneous and heterogeneous instances. Table A.5 in the online supplement shows that HD300 gives significantly better gaps than ALNS for all 12 item instances, whereas it also gives better gaps for some of the 24-item instances. HD600 always gives the best gaps, with the heterogeneous 24-item 60-period instances being the sole exception, in which ALNS is only marginally better. On the upper bound quality, Table A.6 in the online supplement shows that both HD approaches compare very favorably with ALNS, with HD300 being marginally worse and HD600 marginally better than ALNS. This result seems striking at first, because ALNS is a sophisticated heuristic designed specifically to obtain good quality upper bounds. However, it is in line with a result of Larsson and Patriksson (2006), that incremental heuristics that start from a near-optimal lower bound provide solutions of good quality.

6.4. Comparison with an Alternative Branch-and-Price Implementation

Pimentel et al. (2010) employ formulation (1)–(5) to formulate and compare three different decomposition schemes for capacitated lot sizing: (i) the item decomposition, where the capacity constraints (4) are considered as linking constraints and each subproblem is a singe-item uncapacitated lot-sizing problem; (ii) the period decomposition, where the demand balance constraints (2) are considered as linking constraints, and each subproblem is defined over a single period; and (iii) a multiple decomposition, in which they apply both (i) and (ii) simultaneously. In their computational experiments, the period decomposition was the most competitive branch-and-price implementation, despite the fact that multiple decomposition gives a better lower bound. Table A.7 in the online supplement compares the results of their product decomposition with a horizon decomposition implementation that uses zero overlap and 10-period subproblems. Pimentel et al. (2010) run their algorithms with a 3,600 CPU time limit, and used a Pentium IV to perform their experiments. According to SPEC (http://www.spec.org) our machine is about 70% faster, and we therefore pose a time limit of 1,000 seconds, to ensure that our results are comparable.

Table A.7 in the online supplement shows that horizon decomposition achieves a better performance in most cases, even when the selected configuration does not contain any overlaps, which could improve the lower bound. Specifically, the product decomposition delivers integrality gaps of better quality in eight of 36 instances, the horizon decomposition in 26 of 36 instances, and both implementations solve all instances of x11117 and x11127 to optimality. Pimentel et al. (2010) do not report individual lower and upper bounds for each instance, and therefore separate comparisons of upper and lower bounds are not possible. It should be noted that alternative configurations could possibly deliver better integrality gaps in the same amount of time. However, a zero overlap with a large subproblem size strikes a good balance between obtaining a good quality lower bound and directing the heuristics to obtain a good quality upper bound.

6.5. Comparison with Branch and Cut

We also compared horizon decomposition against CPLEX v12.6. The purpose of this comparison is to investigate whether a horizon decomposition approach delivers competitive results against a state-of-the-art commercial solver for certain classes of problems. Since

our algorithm uses CPLEX to solve the subproblems, the interpretation of our results should be that in some classes of hard problems it is more efficient to use branch-and-cut technology within a carefully selected branch-and-price horizon decomposition rather than as a stand-alone solver. Since the suggested methodology delivers a lower bound, the main focus on our experiments is the strength of the lower bound obtained by each approach. However, we also assess the final integrality gap by taking into account the best feasible solution that each method finds.

6.5.1. Data. We focus on problems with small items to periods ratios, since they seem to be the most challenging ones (see also Müller et al. 2012). Specifically, we generated sets of 10 problem instances, each with 2, 4, 6, 8, and 10 items and 100 periods. In total, 50 new problems were constructed. The average capacity utilization was 120%, with some instances that need initial inventory for feasibility. To the best of our knowledge, this is the first data set that includes instances that need initial inventory. Although it is well known that high capacity utilizations characterize hard problems, it is usually the case that the resulting data set is infeasible without initial inventory. Trigeiro et al. (1989, p. 359), who constructed the most widely used CLST data set, write the following:

> Rather than solve the NP complete feasibility test for each problem, we simply threw out problems for which no feasible solution was found by the heuristic. (...) This results in an unavoidable and unmeasurable bias in problem generation. It occurs mostly for tightly constrained problems.

Since then, the assessment of this class of problems has been neglected. Figure 4 graphs the average root gap of CPLEX v12.6 and the number of nodes explored in 3,600 seconds against the number of items $|I|$.

Two useful conclusions can be drawn. First, the root node integrality gaps are between 25% and 39%, suggesting that solving these instances to within an acceptable tolerance may be challenging. To put these numbers in perspective, for the seven instances from Trigeiro's G data set that are supposed to be among the hardest (Van Vyve and Wolsey 2006), CPLEX v12.6 has an average root gap of 2.54% and needs an average 60,000 nodes and 200 seconds to solve them to optimality. Moreover, for the instances examined in Müller et al. (2012) and Süral et al. (2009) the average gaps vary between 14% and 20%. Second, the average number of nodes explored generally decreases as the number of items increases, since the linear programs at each node become larger. This is not the case for 10-item instances, possibly because CPLEX might adopt a different branch selection strategy. With these conclusions at hand, it is useful to explore the performance of a horizon decomposition approach.

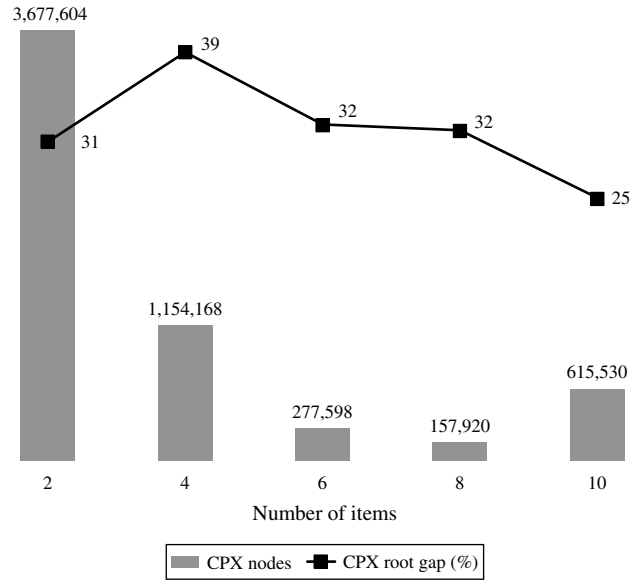

Figure 4 Average Root Integrality Gap and Number of Nodes Explored by CPLEX

Note. Time limit is 3,600 seconds.

6.5.2. Selection of a horizon configuration. To devise a good horizon configuration, we use insights from the computational experiment of Section 6.1. Our intention is to utilize configurations that achieve a good lower bound at the root node of the branch-and-bound tree. We utilize simple qualitative rules to determine a well-performing configuration, suggesting that horizon decompositions can achieve competitive performance without sophisticated selection rules. From our computational experiments, we observe that two-period overlaps achieve the smallest, overall, integrality gaps. To this end, for instances with a small number of items, namely, two or four items, we devise two-period overlaps, combined with a large subproblem size, namely, 12 periods. We choose 12 periods instead of 10 so that all subproblems have equal length. For problems with 6, 8, and 10 items, we select smaller subproblems and overlaps. In particular, we use medium-sized subproblems, with six periods, and a single period of overlap.

6.5.3. Horizon decomposition and branch and cut. Table A.8 in the online supplement reports the relative performance of CPLEX and horizon decomposition at the aforementioned problems. In particular, the small integrality gaps indicate that horizon decomposition constitutes a promising approach, particularly for problems with a small number of items. It is worth noticing that although CPLEX explores more nodes and therefore it is more likely to find good heuristic solutions, the final average gaps are in favor of horizon decomposition, due to the stronger lower bounds that it obtains. Finally, the maximum benefit of horizon decomposition is demonstrated in instances with two

items. Specifically, we were able to solve to optimality seven of 10 instances, three of which at the root node. The average CPU time for these 10 instances is 89 seconds. For the same instances, CPLEX obtained an average gap of 3.7% after one hour of CPU time.

7. Generalizations

In this section we demonstrate potential generalizations of the horizon decomposition approach. First, we present an extension that stems naturally from our work in the CLST that fits well to problems with sparse constraint matrices. Then, we consider an alternative approach that is deemed more appropriate for problems with dense constraint matrices. Both methods are applicable to generic mixed-integer linear programs, which we consider in the following form:

$$[\mathbf{P}] \quad \min \ c^T x \tag{32}$$

$$\text{s.t.} \ Ax = b \tag{33}$$

$$x \in X. \tag{34}$$

The set X describes trivial restrictions such as integrality constraints and range bounds on single variables. We let $I := \{1, \dots, c\}$ be the variable index set and $R := \{1, \dots, r\}$ be the row index set. For notational simplicity, we interpret indexing of a vector or matrix over a set as a reference to the quantities defined over this set. We show that each of the following generalizations takes advantage of different structural characteristics in order to decompose the problem efficiently.

7.1. Extension of the Horizon Decomposition Principle: Row Partitioning

The essence of horizon decomposition is about replicating variables that are in multiple constraints in such a way that the problem matrix is decomposed. We partition the row index set R into two mutually exclusive and exhaustive sets R^1 and R^2, i.e., $R = R^1 \cup R^2$ and $R^1 \cap R^2 = \varnothing$. The extension to more sets, and also the case with $R^1 \cap R^2 \neq \varnothing$ are straightforward and are omitted to ease the exposition. It is of interest to identify which variable indexes are common in sets R^1 and R^2. To this end, we define $\bar{V}_s = \{i \in I\colon \exists j \in R^s \text{ with } a_{ij} \neq 0\}$ for $s \in \{1, 2\}$, the index set of variables that appear in R^s, $V = \bar{V}_1 \cap \bar{V}_2$, the index set of variables that appear both in R^1 and R^2, and $V_s = \bar{V}_s \setminus V$ for $s \in \{1, 2\}$, the index set of variables that appear exclusively in V_s. Using this notation and selecting a $\lambda \in (0, 1)$, we can recast problem [**P**] as follows:

$$[\mathbf{P}_1] \quad \min \ \left\{ c^T_{V_1} x_{V_1} + c^T_{V_2} x_{V_2} + \lambda c^T_V x^1_V + (1-\lambda) c^T_V x^2_V \right\} \tag{35}$$

$$\text{s.t.} \ A_{R^1 V_1} x_{V_1} + A_{R^1 V} x^1_V = b_{R^1} \tag{36}$$

$$A_{R^2 V_2} x_{V_2} + A_{R^2 V} x^2_V = b_{R^2} \tag{37}$$

$$x^1_V - x^2_V = 0_V \tag{38}$$

$$x_{V1} \in X_{V1} \ x_{V2} \in X_{V2} \ x^1_V, x^2_V \in X_V. \tag{39}$$

[**P**] has structure that is amenable to Dantzig-Wolfe decomposition, and it constitutes a generalization of the lot-sizing formulation (6)–(14) presented in Section 3.2. Specifically, in our application sets R^1 and R^2 capture rows indexed over periods $\{1, \dots, k\}$ and $\{l, \dots, m\}$, respectively, for some $k, l \in T$ with $l \leq k+1$. Set V_1 captures the indexes of variables found exclusively in the first subproblem, i.e., $(x_{i1}, y_{i1}, s_{i1}, \dots, x_{il-1}, y_{il-1}, s_{il-1})$ for each item $i \in I$, and similarly set V_2 those found only in the second subproblem. Set V models the indexes of variables defined over the overlap, i.e., $(x_{il}, y_{il}, s_{il}, \dots, x_{ik}, y_{ik}, s_{ik}, s_{ik+1})$ for each item $i \in I$.

In classes of problems where the constraint matrix has an obvious block diagonal substructure, implementing a row partition is relatively straightforward: one has to regulate the subproblem size and horizon overlap based on empirical data. For problems with sparse matrices but no obvious structure, an issue that arises naturally in formulating [$\mathbf{P}_1$] is that of row partition selection. There is a stream of literature that considers the problem of rearranging the constraint matrix in such a way that it exhibits a block-triangular substructure. To the best of our knowledge, Martin (1999) was the first to formulate the problem of rearranging a matrix to decomposable format as a MIP. Specifically, he introduced the matrix decomposition problem as that of decomposing a matrix in bordered diagonal form, given the number of blocks and the size of each block. The recent work of Martin et al. (2011) formulates the same problem as a hypergraph partitioning problem. Moreover, they use the resulting solution within an automatic Dantzig-Wolfe reformulation approach and show experimentally that their approach delivers high-quality dual bounds for some challenging MIPLIB 2003 and MIPLIB 2010 instances. A key difference with our approach is that the algorithm they devise tries to identify hidden structures and come up with one decomposition, whereas the horizon decomposition approach utilizes a family of reformulations, from which the user can select the most suitable one for any particular instance. Although our formulation of [$\mathbf{P}_1$] does not require any structure, one can use the two aforementioned approaches to construct decomposable index sets and decide on the size of the overlaps accordingly.

7.2. An Alternative Generalization: Column Partitioning

The partitioning approach developed in Section 7.1 is well suited to problems with sparse constraint matrices. This is because the number of linking constraints is small and the column generation process exhibits better numerical properties (Martin et al. 2011). This section aims to present an alternative formulation that is well suited to problems with dense constraint matrices, such as set-partitioning problems. Again, we let I be the

variable index set, and consider variable index sets H_1, H_2, and L such that $H_1 \cup H_2 = I$ and $H_1 \cap H_2 = L$. The extension to more partitions is straightforward. For ease of notation, let x_i, c_i, A_i and X_i denote the components of each entity that refer to indexes in set H_i, and x_l, c_l, A_l the components of each entity that refer to indexes in set L. Then [**P**] can be reformulated such that it is amenable to Dantzig-Wolfe decomposition as follows:

$$[\mathbf{P}_2] \quad \min \ \{c_1^T x_1 + \bar{c}_2^T x_2\} \tag{40}$$

$$\text{s.t.} \quad A_1 x_1 - \lambda A_l x_l^1 + s_1 = b/2 \tag{41}$$

$$A_2 x_2 - (1-\lambda) A_l x_l^2 - s_2 = b/2 \tag{42}$$

$$x_l^1 = x_l^2 \tag{43}$$

$$s_1 = s_2 \tag{44}$$

$$x_1 \in X_1,\ x_2 \in X_2,\ x_l^1, x_l^2 \in X_l,\ s_1, s_2 \in \mathbb{R}^r \tag{45}$$

The variables s_1 and s_2 are continuous and their dimension equals the number of rows of matrix A. Also, λ is a fixed scalar and $\bar{c}_{2i} = 0$ if $i \in L$, and $\bar{c}_{2i} = c_{2i}$ otherwise. By dualizing constraints (43) and (44) $[\mathbf{P}_2]$ decomposes in two subproblems, defined over the index sets H_1 and H_2, respectively. This decomposition can be beneficial for problems with a large number of variables and relatively few constraints, or problems that exhibit structure over index sets of variables. The issue of selecting a suitable partition is relevant in this formulation as well. One needs to partition the variables in such a way that those with similar row coefficients belong to the same index set H. To the best of our knowledge, this problem has not been tackled in the literature, but variants of the methods of Martin (1999) and Martin et al. (2011) can also be applied for column partitioning. The effectiveness of such methods remains to be explored and benchmarked against alternative approaches.

8. Conclusions and Future Research

We present a horizon decomposition implementation to the multi-item capacitated lot-sizing problem with setup times. The problem is decomposed in contiguous horizons of smaller size, and the subproblems are of the same type as the original, but have smaller dimension. The developed methodology suggests a family of reformulations, which offer flexibility to regulate the master problem and subproblem size almost independently. A computational study gives empirical evidence on which configurations lead to efficient reformulations. Computational experiments show that the approach delivers strong lower bounds, and it outperforms one of the best heuristics in the literature and another state-of-the-art branch-and-price algorithm. Further experiments on generated data sets show its competitive performance against the branch-and-cut solver CPLEX v12.6. Moreover, the horizon decomposition methodology is generalizable and applicable to other classes of mixed-integer linear programs.

It is only recently that some researchers have started exploring horizon-like decompositions, and there exist many directions for future research. The study of Caprara et al. (2013) applies a similar form of horizon decomposition to the temporal knapsack problem, and Bergner et al. (2014) attempt to devise an automatic reformulation method that identifies hidden structures in the constraint matrix of each instance and performs a Dantzig-Wolfe decomposition with striking results for certain problem classes. The identification of classes of problems where horizon decomposition can be a viable alternative to branch and cut remains an open question. Moreover, an automatic procedure that selects the member of a family of decompositions that strikes the best balance between bound quality and CPU time performance is an area that has not been explored yet on its entirety. Finally, an alternative scheme that generates cutting planes from problem substructures, such as the one devised by Akartunali et al. (2016), is a promising direction that generates cuts from lower dimensional projections of the initial formulation and yet does not require an inner approximation of the feasible region, which is the case for column generation-based approaches.

Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/ijoc.2016.0691.

References

Akartunal K, Miller AJ (2012) A computational analysis of lower bounds for big bucket production planning problems. *Computational Optim. Appl.* 53(3):729–753.

Akartunali K, Fragkos I, Miller A, Wu T (2016) Local cuts and two-period convex hull closures for big-bucket lot-sizing problems. *INFORMS J. Comput.* Forthcoming.

Barany I, Van Roy T, Wolsey LA (1984) Uncapacitated lot-sizing: The convex hull of solutions. *Math. Programming Stud.* 22:32–43.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Belvaux G, Wolsey LA (2000) bc—prod: A specialized branch-and-cut system for lot-sizing problems. *Management Sci.* 46(5):724–738.

Ben Amor H, Desrosiers J, de Carvalho JMV (2006) Dual-optimal inequalities for stabilized column generation. *Oper. Res.* 54(3): 454–463.

Bergner M, Caprara A, Ceselli A, Furini F, Lübbecke ME, Malaguti E, Traversi E (2014) Automatic Dantzig-Wolfe reformulation of mixed integer programs. *Math. Programming* 149(1):391–424.

Caprara A, Furini F, Malaguti E (2013) Uncommon Dantzig-Wolfe reformulation for the temporal knapsack problem. *INFORMS J. Comput.* 25(3):560–571.

Danna E, Rothberg E, Le Pape C (2005) Exploring relaxation induced neighborhoods to improve MIP solutions. *Math. Programming* 102(1):71–90.

Dantzig GB, Wolfe P (1960) Decomposition principle for linear programs. *Oper. Res.* 8(1):101–111.

de Araujo SA, De Reyck B, Degraeve Z, Fragkos I, Jans R (2015) Period decompositions for the capacitated lot sizing problem with setup times. *INFORMS J. Comput.* 27(3):431–448.

Degraeve Z, Jans R (2007) A new Dantzig-Wolfe reformulation and branch-and-price algorithm for the capacitated lot-sizing problem with setup times. *Oper. Res.* 55(5):909–920.

Degraeve Z, Peeters M (2003) Optimal integer solutions to industrial cutting-stock problems: Part 2, benchmark results. *INFORMS J. Comput.* 15(1):58–81.

du Merle O, Villeneuve D, Desrosiers J, Hansen P (1999) Stabilized column generation. *Discrete Math.* 194(1–3):229–237.

Elhallaoui I, Villeneuve D, Soumis F, Desaulniers G (2005) Dynamic aggregation of set-partitioning constraints in column generation. *Oper. Res.* 53(4):632–645.

Eppen GD, Martin RK (1987) Solving multi-item capacitated lot-sizing problems using variable redefinition. *Oper. Res.* 35(6): 832–848.

Fisher ML (2004) The Lagrangian relaxation method for solving integer programming problems. *Management Sci.* 50(12_suppl.): 1861–1871.

Geoffrion AM (1974) Lagrangean relaxation for integer programming. Balinski M, ed. *Approaches to Integer Programming*, Mathematical Programming Studies, Vol. 2 (Springer, Berlin), 82–114.

Guignard M, Kim S (1987) Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Math. Programming* 39(2): 215–228.

Jans R, Degraeve Z (2004) Improved lower bounds for the capacitated lot sizing problem with setup times. *Oper. Res. Lett.* 32(2):185–195.

Kleindorfer PR, Newson EFP (1975) A lower bounding structure for lot-size scheduling problems. *Oper. Res.* 23(2):299–311.

Larsson T, Patriksson M (2006) Global optimality conditions for discrete and nonconvex optimization—With applications to Lagrangian heuristics and column generation. *Oper. Res.* 54(3):436–453.

Lübbecke ME, Desrosiers J (2005) Selected topics in column generation. *Oper. Res.* 53(6):1007–1023.

Manne AS (1958) Programming of economic lot sizes. *Management Sci.* 4(2):115–135.

Martin A (1999) Integer programs with block angular structure. PhD thesis, Konrad-Zuse-Zentrum für Informationstechnik Berlin.

Martin B, Lübbecke ME, Malaguti E, Traversi E (2011) Partial convexification of general MIPs by Dantzig-Wolfe reformulation. Günlük O, Woeginger GJ, eds. *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, Vol. 6655 (Springer-Verlag, Berlin), 39–51.

Miller A, Nemhauser GL, Savelsbergh MW (2000) Solving multi-item capacitated lot-sizing problems with setup times by branch-and-cut. CORE Discussion Papers 2000039, Université catholique de Louvain, Center for Operations Research and Econometrics, Belgium.

Müller LF, Spoorendonk S, Pisinger D (2012) A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *Eur. J. Oper. Res.* 218(3):614–623.

Pimentel CMO, Pereira e Alvelos F, de Carvalho JMV (2010) Comparing Dantzig-Wolfe decompositions and branch-and-price algorithms for the multi-item capacitated lot-sizing problem. *Optim. Methods Software* 25(2):299–319.

Pochet Y, Wolsey LA (2006) *Production Planning by Mixed Integer Programming*, Springer Series in Operations Research and Financial Engineering (Springer-Verlag, New York).

Süral H, Denizel M, Van Wassenhove LN (2009) Lagrangean relaxation based heuristics for lot sizing with setup times. *Eur. J. Oper. Res.* 194(1):51–63.

Trigeiro WW, Thomas LJ, McClain JO (1989) Capacitated lot sizing with setup times. *Management Sci.* 35(3):353–366.

Vanderbeck F (1998) Lot-sizing with start-up times. *Management Sci.* 44(10):1409–1425.

Vanderbeck F (2005) Implementing mixed integer column generation. Desaulniers G, Desrosiers J, Solomon M, eds. *Column Generation* (Springer, New York), 331–358.

Vanderbeck F (2011) Branching in branch-and-price: A generic scheme. *Math. Program.* 130(2):249–294.

Vanderbeck F, Savelsbergh MWP (2006) A generic view of Dantzig-Wolfe decomposition in mixed integer programming. *Oper. Res. Lett.* 34(3):296–306.

Van Vyve M, Wolsey LA (2006) Approximate extended formulations. *Math. Programming* 105(2–3):501–522.

Villeneuve D, Desrosiers J, Lübbecke ME, Soumis F (2005) On compact formulations for integer programs solved by column generation. *Ann. Oper. Res.* 139(1):375–388.

Wagner HM, Whitin TM (1958) Dynamic version of the economic lot size model. *Management Sci.* 5(1):89–96.