ECONSTOR
Make Your Publications Visible.

A Service of

ZBW Leibniz-Informationszentrum Wirtschaft
Leibniz Information Centre for Economics

Kimms, Alf

Working Paper — Digitized Version

Short-term management of lot production

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 480

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Kimms, Alf (1998) : Short-term management of lot production, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 480, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/147583

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.

WWW.ECONSTOR.EU

Mitglied der Leibniz-Gemeinschaft

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 480

Short–Term Management of Lot Production

A. Kimms

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 480

Short–Term Management of Lot Production

A. Kimms

September 199

Alf Kimms
Lehrstuhl für Produktion und Logistik, Institut für Betriebswirtschaftslehre, Christian–Albrechts–Universität zu Kiel, Olshausenstr. 40, 24118 Kiel, Germany
email: Kimms@bwl.uni–kiel.de
URL: http://www.wiso.uni–kiel.de/bwlinstitute/Prod
ftp://ftp.wiso.uni–kiel.de/pub/operations–research

Abstract

This contribution presents an efficient solution method for solving the major short–term planning problems which occur, if lot production has to be managed. In our focus are the two most important aspects, i.e. lot sizing and scheduling. Since both problems heavily interact, we propose an approach which solves these problems simultaneously. This is in contrast to what traditional Manufacturing Resource Planning (MRP II) logic suggests. But, since we pay special attention on complex product structures and several scarce resources, our approach dominates MRP II. A computational study reveals that suboptimal results within the 6.6% scope from the optimum are obtained within less than half a second for small instances.

Keywords: Production Management, MRP II, Lot Sizing, Scheduling, Complex Product Structures, Multiple Scarce Resources, Dynamic Demand

1 Introduction

Lot production occurs when several non–customized products are manufactured under a make–to–order regime. Managing this situation means that, besides some medium–term problems such as designing the product line, forecasting demand, providing sufficient capacity, and generating a master production schedule, hard planning problems remain to be solved on the short–term. These are lot sizing and scheduling.

Precedence constraints among the items which define a complex multi–level gozinto–structure, and several resources, say machines, which provide scarce capacity make these problems even harder in a real–world environment and must not be ignored. The traditional logic of MRP II [12, 39], which is probably the most widespread planning philosophy, suggests the following: Lot sizing and scheduling should be done in a stepwise fashion moving through three planning stages. Stage 1 performs lot sizing. To keep the problem easy at this stage, capacity restrictions are neglected. Starting with the end items, a sequence of single–item Wagner–Whitin lot sizing problems [3, 14, 37, 38] is solved. The result of the lot size decision for an item defines the demand of its immediate predecessors. Successively, lot sizing can thus be done for all items. Note that the computation of the lot sizes for an item is only affected by the decisions made for its successors. Preceding items do not affect the computation of lot sizes. Also note that the overall result is not optimal even for the uncapacitated multi–level lot sizing problem which requires much more sophisticated approaches [1, 2]. Given the result obtained so far, the second stage of MRP II takes over now. Stage 2 performs capacity requirements planning. Because capacity restrictions were ignored up to here, capacity demand at some points in time usually exceeds the available capacity. By shifting some lots forward or backward in time, the second phase tries to meet the capacity limits. Finally, Stage 3 determines a sequence in which orders are released to the shop floor.

As it became clear, the reason why practitioners complain about short–term production planning is inherent in the MRP II logic. Hence, an improved concept for lot sizing and scheduling in the presence of multi–level gozinto–structures and several scarce resources is needed. This is to be presented in this text. But, before we turn into more details, we should have a brief look at what other researchers have published.

Many authors have considered a multi–level Wagner–Whitin–type of problem, i.e. they ignored capacity constraints. Most of them have tested so–called improved heuristics where methods for the single–level Wagner–Whitin problem are applied level by level in order to construct a feasible plan. More sophisticated approaches are described in [18, 31, 32]. A sensitivity analysis is done in [36], and complexity results for uncapacitated, multi–level lot sizing are provided in [4].

Most authors who consider capacitated, multi–level lot sizing make restrictive assumptions. [34], for example, takes only a single bottleneck machine into account. [30] focuses on assembly gozinto–structures. The work in [6] is confined to two levels only. Multi–level lot sizing, where general gozinto–structures and multiple machines are taken into account, is dealt with in [17, 35].

The literature on multi–level lot sizing and scheduling is sparse. A hierarchical integration of some lot sizing and some scheduling procedure is discussed in [7, 29]. As pointed out in [11], most work on lot sizing and scheduling is confined to the single–level case.

Work of our own also started with restrictive cases. For multi–level lot sizing and scheduling with a single–machine, we refer to [21, 22, 23]. Methods for the multi–machine case are published in [25, 26] which are the ones we have to compete with. In [24] our efforts are summarized comprehensively.

The paper at hand extends the so–called demand shuffle algorithm described in [23] for the single–machine case to yield an efficient procedure for multi–level lot sizing and scheduling with multiple machines. The text is organized as follows: Section 2 introduces a mixed–integer program (MIP) to give a precise definition of the problem to solve. In Section 3, the sophisticated solution method is presented. A computational study in Section 4 shows that the procedure improves previous work decidedly. Concluding remarks in Section 5 complete the paper.

2 The Short–Term Planning Problem

The problem to be solved can be described as follows: Several items are to be produced in order to meet some known (or estimated) dynamic demand without backlogs and stockouts. Precedence relations among these items define an acyclic gozinto–structure of the general type. In contrast to many authors who allow demand for end items only, now, demand may occur for all items including component parts. The finite planning horizon is subdivided into a number of discrete time periods. Positive lead times are given due to technological restrictions such as cooling or transportation, for instance. Furthermore, items share common machines with scarce capacity. The capacities may vary over time. Producing one item requires an item–specific amount of the available capacity. All data are assumed to be deterministic.

Items which are produced in a period to meet some future demand must be stored in inventory and thus cause item–specific holding costs.

Each item requires one machine for which a setup state has to be taken into account. Production can only take place if a proper state is set up. Setting a machine up for producing a particular item incurs item–specific setup costs which are assumed to be sequence independent. Setup times are not considered here. Once a certain setup action is performed, the setup state is kept up until another setup changes the current state. Hence, same items which are produced having some idle time in–between do not enforce more than one setup action.

The most fundamental assumption here is that for each machine at most one setup may occur within one period. Hence, at most two items sharing a common machine may be produced per period. Due to this assumption, the problem is known as the proportional lot sizing and scheduling problem (PLSP) [10, 11, 16, 24]. By choosing the length of each time period appropriately small, the PLSP is a good approximation to a continuous time axis. It refines the well–known discrete lot sizing and scheduling problem (DLSP) [9, 11, 15, 28, 33] as well as the continuous setup lot sizing problem (CSLP) [5, 11, 19, 20]. Both assume that at most one item may be produced per period. All three models could be classified as small bucket models since only a few (one or two) items are produced per period. In contrast to this, the well–known capacitated lot sizing problem (CLSP) [8, 11, 13, 27] represents a large bucket model since many items can be produced per period. Note, the CLSP does not include sequence decisions.

Let us now introduce some notation. In Table 1 the decision variables are defined. Likewise, the parameters are explained in Table 2. Using this notation, we are now able to present a MIP–model formulation.

Symbol	Definition
I_{jt}	Inventory for item j at the end of period t.
q_{jt}	Production quantity for item j in period t.
x_{jt}	Binary variable which indicates whether a setup for item j occurs in period t ($x_{jt} = 1$) or not ($x_{jt} = 0$).
y_{jt}	Binary variable which indicates whether machine m_j is set up for item j at the end of period t ($y_{jt} = 1$) or not ($y_{jt} = 0$).

Table 1: Decision Variables for the PLSP

$$\min \sum_{j=1}^{J} \sum_{t=1}^{T} (s_j x_{jt} + h_j I_{jt}) \tag{1}$$

subject to

Symbol	Definition
a_{ji}	Gozinto–factor. Its value is zero if item i is not an immediate successor of item j. Otherwise, it is the quantity of item j that is directly needed to produce one unit of item i.
C_{mt}	Available capacity of machine m in period t.
d_{jt}	External demand for item j in period t.
h_j	Non–negative holding cost for having one unit of item j one period in inventory.
I_{j0}	Initial inventory for item j.
$\mathcal{J}_m$	Set of all items that share the machine m, i.e. $\mathcal{J}_m \stackrel{def}{=} \{j \in \{1,\ldots,J\} \mid m_j = m\}$.
J	Number of items.
M	Number of machines.
m_j	Machine on which item j is produced.
p_j	Capacity needs for producing one unit of item j.
s_j	Non–negative setup cost for item j.
$\mathcal{S}_j$	Set of immediate successors of item j, i.e. $\mathcal{S}_j \stackrel{def}{=} \{i \in \{1,\ldots,J\} \mid a_{ji} > 0\}$.
T	Number of periods.
v_j	Positive and integral lead time of item j.
y_{j0}	Unique initial setup state.

Table 2: Parameters for the PLSP

$$I_{jt} = I_{j(t-1)} + q_{jt} - d_{jt} - \sum_{i \in \mathcal{S}_j} a_{ji} q_{it} \qquad \begin{array}{l} j = 1,\ldots,J \\ t = 1,\ldots,T \end{array} \tag{2}$$

$$I_{jt} \geq \sum_{i \in \mathcal{S}_j} \sum_{\tau=t+1}^{\min\{t+v_j,T\}} a_{ji} q_{i\tau} \qquad \begin{array}{l} j = 1,\ldots,J \\ t = 0,\ldots,T-1 \end{array} \tag{3}$$

$$\sum_{j \in \mathcal{J}_m} y_{jt} \leq 1 \qquad \begin{array}{l} m = 1,\ldots,M \\ t = 1,\ldots,T \end{array} \tag{4}$$

$$x_{jt} \geq y_{jt} - y_{j(t-1)} \qquad \begin{array}{l} j = 1,\ldots,J \\ t = 1,\ldots,T \end{array} \tag{5}$$

$$p_j q_{jt} \leq C_{m_j t}(y_{j(t-1)} + y_{jt}) \qquad \begin{array}{l} j = 1,\ldots,J \\ t = 1,\ldots,T \end{array} \tag{6}$$

$$\sum_{j \in \mathcal{J}_m} p_j q_{jt} \leq C_{mt} \qquad \begin{array}{l} m = 1,\ldots,M \\ t = 1,\ldots,T \end{array} \tag{7}$$

$$y_{jt} \in \{0,1\} \qquad \begin{array}{l} j = 1,\ldots,J \\ t = 1,\ldots,T \end{array} \tag{8}$$

$$I_{jt}, q_{jt}, x_{jt} \geq 0 \qquad \begin{array}{l} j = 1,\ldots,J \\ t = 1,\ldots,T \end{array} \tag{9}$$

The objective (1) is to minimize the sum of setup and holding costs. Equations (2) are the inventory balances. At the end of a period t we have in inventory what was in there at the end of period $t-1$ plus what is produced minus external and internal demand. To fulfill internal demand we must respect positive lead times. Restrictions (3) guarantee so. Constraints (4) make sure that the setup state of each machine is uniquely defined at the end of each period. Those periods in which a setup happens are spotted by (5). Note that idle periods may occur in order to save setup costs. Due to (6) production can only take place if there is a proper setup state either at the beginning or at the end of a particular period. Hence, at most two items can be manufactured on each machine per period. Capacity constraints are formulated in

(7). Since the right hand side is a constant, overtime is not available. (8) define the binary–valued setup state variables, while (9) are simple non–negativity conditions. The reader may convince himself that due to (5) in combination with (1) setup variables x_{jt} are indeed zero–one valued. Hence, non–negativity conditions are sufficient for these. For letting inventory variables I_{jt} be non–negative backlogging cannot occur.

3 Demand Shuffle Heuristic

In this section we get acquainted with a novel heuristic to attack the PLSP with multiple machines. It is an extension of what has been presented in [23] for the single–machine case. For the procedure to be described, the name demand shuffle has been coined. Demand shuffle is a random sampling method that is combined with a data structure which affects the construction of production plans. Due to some problem specific data structure manipulations each iteration has a tendency to yield a different production plan. Hence, the approach can be seen as a diversified search for production plans with low objective function value. An example in the appendix illustrates the procedure.

The basic idea of this method is that upper bounds Q_{jt} for the sizes of lots for items $j = 1, \ldots, J$ in periods $t = 1, \ldots, T$ bring in a means to control the generation of production plans. Roughly speaking, the three ingredients of the method proposed here are a rule to compute the Q_{jt}–values, a construction scheme respecting these upper bounds, and a way to enforce that the upper bounds will likely change from iteration to iteration.

3.1 Data Structure

A graphical representation derived from the gozinto–trees helps to compute upper bounds Q_{jt} for the production quantities. The fundamental idea is that for each positive entry in the external demand matrix a gozinto–tree–like structure not only contains the information what (external or internal) demand occurs for what item, but also a deadline at which this demand is to be met.

More formally, the initial data structure Γ^{DS} can be defined as

$$\Gamma^{DS} \stackrel{def}{=} \bigcup_{j=1}^{J} \bigcup_{t=1}^{T} \tilde{\Gamma}_j^{DS}(t, d_{jt}, \omega_1(j,t), 0, -1) \tag{10}$$

where the function ω_1 is defined by

$$\omega_1(j,t) \stackrel{def}{=} \sum_{i=1}^{J} \sum_{\tau=1}^{t-1} \chi(d_{i\tau}) nodes_i + \sum_{i=1}^{j-1} \chi(d_{it}) nodes_i \tag{11}$$

and computes unique labels for the root nodes of the gozinto–trees with χ being an auxiliary function defined as

$$\chi(x) \stackrel{def}{=} \begin{cases} 1 & \text{, if } x > 0 \\ 0 & \text{, otherwise} \end{cases} . \tag{12}$$

The expression $nodes_j$ stands for the number of nodes in a gozinto–tree with item j being its root node. It can recursively be computed using

$$nodes_j \stackrel{def}{=} \sum_{i=1}^{J} nodes_{ji} \tag{13}$$

where

$$nodes_{ji} \stackrel{def}{=} \begin{cases} 1 & \text{, if } j = i \\ 0 & \text{, if } j \neq i \text{ and } i \notin \bar{\mathcal{P}}_j \\ \sum_{h \in S_i} nodes_{jh} & \text{, otherwise} \end{cases} \tag{14}$$

is the number of nodes with item index i in the gozinto–tree having item j as its root node. The function

$\tilde{\Gamma}_j^{DS}$ is given by

$$\tilde{\Gamma}_j^{DS}(period, demand, label, path, successor) \stackrel{def}{=} \begin{cases} \emptyset \quad , \text{if } demand = 0 \\ \{(j, period, period, demand, label, \\ \quad \alpha(j, path), \\ \quad successor, \\ \quad \bigcup_{i \in \mathcal{P}_j} \tilde{\Gamma}_i^{DS}(period - v_i, \\ \qquad\qquad a_{ij} demand, \\ \qquad\qquad \omega_2(j, i, label), \\ \qquad\qquad \alpha(j, path), \\ \qquad\qquad label)\} \\ \quad , \text{otherwise} \end{cases} \tag{15}$$

where the function ω_2 is defined by

$$\omega_2(j, i, label) \stackrel{def}{=} 1 + label + \sum_{\substack{k \in \mathcal{P}_j \\ k < i}} nodes_k \tag{16}$$

to compute unique labels for the internal demand nodes.

As introduced above (see (15)), each node is represented by an eight–tuple. The first entry is the number of the respective item and the last entry is the set of immediately preceding nodes. At the second position we find the deadline until which the demand represented by the node is to be met. This deadline is known for external demands only, hence we temporarily fill in a deadline derived from a lot–for–lot policy at this position for internal demands and update the entry at this position as we proceed. The third position also contains a period's index. It is the deadline of the node when scheduling is done on a lot–for–lot basis. Initially, the entries at position two and three are equal. At position four we have the demand size that is represented by the node. For root nodes this equals the external demand. The fifth position contains a unique label out of $\mathcal{L}$ to identify each node where $\mathcal{L}$ equals the interval

$$\mathcal{L} \stackrel{def}{=} [0, \ldots, \sum_{j=1}^{J} \sum_{t=1}^{T} \chi(d_{jt}) nodes_j - 1]. \tag{17}$$

At the sixth position we find an encoding of the path from the root node to the current node in the gozinto–tree. This encoding scheme is based on a J–ary code where

$$\alpha(j, path) \stackrel{def}{=} path \cdot (J + 1) + j. \tag{18}$$

In the sequel, for $h \in \mathcal{L}$, let $item(h)$ denote the item number of the node with label h, and let $deadline(h)$ a retrieval function that yields the deadline entry at the tuple position two. For retrieving the initial deadline entry at position three of a node h, $deadline^{L4L}(h)$ will be used. To access the demand size of node h we use $demand(h)$. Furthermore, let us use $path(h)$ to access the value defined by the function α when the node has label h. Note, in the data structure Γ^{DS} we may have several nodes with the same path value. This happens to be if there is more than one period in which external demand occurs for the same item. At position seven in the tuple we have the label of the immediate successor in the gozinto–tree. The value -1 marks root nodes. For $h \in \mathcal{L}$, let $succ(h)$ be a retrieval function for that value. Analogously, let $preds(h)$ denote the set of labels of immediately preceding nodes which are the nodes listed at position eight.

3.2 Data Structure Manipulations

An upper bound for the production quantity of an item j in a period t is

$$Q_{jt} \stackrel{def}{=} \sum_{\substack{h \in \mathcal{L} \\ item(h) = j \\ deadline(h) \geq t}} demand(h) - \sum_{\tau = t+1}^{T} q_{j\tau}. \tag{19}$$

Since the deadline entries in the initial data structure are computed under a lot–for–lot assumption, the cumulative demand for an item always is less than or equal to this upper bound in the first run of the construction scheme. But, modifying the deadline values after each construction phase (e.g. by assigning an earlier deadline) may give upper bounds smaller than the cumulative demand and therefore affects the construction of production plans.

What we need to discuss now is the way the deadline entries are modified. Choosing some nodes at pure random and updating their deadline values with random numbers out of the interval $[1, T]$ would be an easy–to–implement possibility, though not a very insightful one. The strategy that we use can be outlined as follows: First, choose a node on the basis of some priority rule. Then, decide if the deadline entry should be increased or decreased using problem specific insight. And finally, compute the new deadline. For the sake of convenience, let us from now on in this subsection assume that initial inventories do not exist. For all nodes in the data structure the deadline entries can thus be assumed to be positive.

The process of modifying deadlines becomes a bit more illustrative if we imagine a table with J rows and T columns similar to a demand matrix. At a position (j, t) of that matrix we find the sum of all demands that are represented by nodes with label h for which $item(h) = j$ and $deadline(h) = t$ holds. Modifying the deadline entry of a node then corresponds to shifting some demand to the left or to the right, respectively, to give a new table. When doing so over and over again, it looks like shuffling the demand. This coins the name demand shuffle as we like to call the heuristic as a whole.

Before more details are given, let us discuss what side constraints the modification of deadlines should respect. As preliminary studies revealed, shifting demands should not be done arbitrarily. Since we face scarce capacities, the condition

$$\sum_{\tau=1}^{t} C_{m\tau} \geq \sum_{\substack{h \in \mathcal{L} \\ m_{item(h)} = m \\ deadline(h) \leq t}} p_{item(h)} demand(h) \tag{20}$$

should be guaranteed to be true for $m = 1, \ldots, M$ and $t = 1, \ldots, T$. Initially, this ought to be true, because no feasible solution exists, otherwise. Note, we only need to test this condition when some demand is shifted to the left. Right shift operations cannot lead to a violation of this restriction.

Furthermore, the gozinto–structure defines some precedence relations that should be taken into account. Shifting some demand to the left (which corresponds to decreasing the deadline entry of a node) should respect the internal demand that is to be met. Stating this mathematically, a lower bound for a valid deadline entry of a node $h \in \mathcal{L}$ is:

$$deadline^{I}_{LB}(h) \stackrel{def}{=} \begin{cases} \max\{deadline(k) + v_{item(k)} \mid k \in preds(h)\} & \text{, if } preds(h) \neq \emptyset \\ 1 & \text{, otherwise} \end{cases} \tag{21}$$

Something similar applies when shifting demand to the right (which corresponds to increasing the deadline entry of a node). This time we compute an upper bound for the deadline entry of a node $h \in \mathcal{L}$ by

$$deadline^{I}_{UB}(h) \stackrel{def}{=} \begin{cases} deadline(succ(h)) - v_{item(h)} & \text{, if } succ(h) \neq -1 \\ deadline^{L4L}(h) & \text{, otherwise} \end{cases} . \tag{22}$$

Note, initially

$$deadline^{I}_{LB} = deadline(h) = deadline^{I}_{UB}(h)$$

holds for $h \in \{k \in \mathcal{L} \mid preds(k) \neq \emptyset\}$ and

$$deadline^{I}_{LB} = 1 \leq deadline(h) = deadline^{I}_{UB}(h)$$

is valid for $h \in \{k \in \mathcal{L} \mid preds(k) = \emptyset\}$.

In addition, we utilize the fact that without loss of generality two external demands for the same item are met so that production fulfilling the early demand takes place before the production for the late demand does (this is often called the earliest due date rule). In a multi–level case this result can be extended for internal demands in the following way: If two internal demands have the same position in a gozinto–tree representation then the earliest due date rule applies for these, too. In summary and in terms of our representation of a PLSP–instance, we can now state that only those production plans

need to be considered where for each node with label $h \in \mathcal{L}$ its demand is met by production in a certain period only when the demand for all nodes in the set

$$\mathcal{LW}(h) \stackrel{def}{=} \{k \in \mathcal{L} \mid path(k) = path(h) \wedge deadline^{L4L}(k) < deadline^{L4L}(h)\} \tag{23}$$

is met by production no later than that period. For $h \in \mathcal{L}$ we easily verify the property

$$k_1, k_2 \in \mathcal{LW}(h) \text{ and } k_1 < k_2 \Rightarrow \mathcal{LW}(k_1) \subset \mathcal{LW}(k_2) \subset \mathcal{LW}(h) \tag{24}$$

which enables us to avoid testing the complete set $\mathcal{LW}(h)$ to see if the earliest due date rule is respected when shifting the demand represented by node h to the left. It suffices to check the node with label $leftwing(h)$ where

$$leftwing(h) \stackrel{def}{=} \begin{cases} \max(\mathcal{LW}(h)) & \text{, if } \mathcal{LW}(h) \neq \emptyset \\ -1 & \text{, otherwise} \end{cases} \tag{25}$$

and where the value -1 indicates that no such test is to be made, because there simply is no demand that should be met prior to the demand represented by node h according to the earliest due date rule. As a result we have another lower bound for the deadline entry of a node $h \in \mathcal{L}$:

$$deadline^{II}_{LB}(h) \stackrel{def}{=} \begin{cases} deadline(leftwing(h)) & \text{, if } leftwing(h) \neq -1 \\ deadline^{L4L}(h) & \text{, otherwise} \end{cases} \tag{26}$$

A remarkable point to note is that in the case $leftwing(h) = -1$ the lower bound is not necessarily the value one as it would be reasonable to expect, because the demand could be left shifted to period 1 without violating the earliest due date rule. The explanation is that shifting demands for an item to the left should help to build large lots for that particular item. Left shifts can therefore be omitted if there is no other demand for that item in earlier periods, or, which is a bit more restrictive, if there is no demand for which the earliest due date rule must be respected. The lower bound $deadline^{L4L}(h)$ which equals the initial deadline entry of node h prevents a left shift of the demand represented by node h.

What is done for left shift operations can also be done for right shift operations with minor modifications. Due to the earliest due date rule we have a precedence relation between each node $h \in \mathcal{L}$ and every node in the set

$$\mathcal{RW}(h) \stackrel{def}{=} \{k \in \mathcal{L} \mid path(k) = path(h) \wedge deadline^{L4L}(k) > deadline^{L4L}(h)\} \tag{27}$$

or, which is more efficient, between each node $h \in \mathcal{L}$ and the node $rightwing(h)$ where

$$rightwing(h) \stackrel{def}{=} \begin{cases} \min(\mathcal{RW}(h)) & \text{, if } \mathcal{RW}(h) \neq \emptyset \\ -1 & \text{, otherwise} \end{cases} \tag{28}$$

and -1 again signals that there is no additional precedence relation due to the earliest due date rule. Note, for $h \in \mathcal{L}$ the following properties identify the $rightwing$ and the $leftwing$ function being reversal:

$$h = rightwing(leftwing(h)), \text{ if } leftwing(h) \neq -1 \tag{29}$$

$$h = leftwing(rightwing(h)), \text{ if } rightwing(h) \neq -1 \tag{30}$$

For $h \in \mathcal{L}$ an upper bound for the deadline entry is

$$deadline^{II}_{UB}(h) \stackrel{def}{=} \begin{cases} deadline(rightwing(h)) & \text{, if } rightwing(h) \neq -1 \\ deadline^{L4L}(h) & \text{, otherwise} \end{cases} . \tag{31}$$

As a result we have that shifting demand to the left or to the right by decreasing or increasing the deadline value of a node $h \in \mathcal{L}$ should be done so that the new entry $deadline(h)$ fulfills

$$\begin{array}{c} \max\{deadline^{I}_{LB}(h), deadline^{II}_{LB}(h)\} \\ \leq \\ deadline(h) \\ \leq \\ \min\{deadline^{I}_{UB}(h), deadline^{II}_{UB}(h)\}. \end{array} \tag{32}$$

Note, once the deadline entry of a node h is modified, not only the bounds for the deadline entry of the node h but also the bounds $deadline^{I}_{LB}(succ(h))$, $deadline^{II}_{LB}(rightwing(h))$ (if $rightwing(h) \neq -1$), $deadline^{I}_{UB}(k)$ for $k \in preds(h)$, and $deadline^{II}_{UB}(leftwing(h))$ (if $leftwing(h) \neq -1$) for deadline entries must be updated before another shift operation can be performed. All other bounds are kept unchanged.

In preliminary tests it turned out that it is best to perform full–size shifts. A left shift of the demand represented by a node h then yields the value

$$deadline(h) = \max\{deadline^{I}_{LB}(h), deadline^{II}_{LB}(h)\}. \tag{33}$$

In the case of a right shift, we get

$$deadline(h) = \min\{deadline^{I}_{UB}(h), deadline^{II}_{UB}(h)\}. \tag{34}$$

Let us now return to discuss how to select a node to modify its deadline entry. Assume the existence of at least one node $h \in \mathcal{L}$ with $leftwing(h) \neq -1$. Otherwise, demands cannot be shifted and the demand shuffle heuristic repeatedly passes the construction phase using the initial data structure again and again. Hence, consider the set of valid labels

$$\mathcal{VL} \stackrel{def}{=} \{h \in \mathcal{L} \mid leftwing(h) \neq -1\} \tag{35}$$

which identifies nodes that are allowed to be shifted. Attached to each node $h \in \mathcal{VL}$ is a priority value π_h which guides the choice of nodes. It is a random process that works as follows. First, choose a label $h^{(0)} \in \mathcal{VL}$ with uniform distribution. Then, iterate the following lines where the index $i = 1, 2, \ldots$ denotes the number of the iteration performed. Choose $h^{(i)} \in \mathcal{VL}$ with uniform distribution. If $\pi_{h^{(i)}} > \pi_{h^{(i-1)}}$ then start a new iteration. Otherwise, terminate the loop with node $h^{(i-1)}$ being the one whose deadline entry should be modified. Note, this process is guaranteed to terminate, because the priority values increase from iteration to iteration. The priority rule that is used in our tests compares the additional holding costs that are incurred when some demand is shifted to the left with the setup costs that are saved when a lot can be built. More formally, we use

$$\pi_h \stackrel{def}{=} \frac{demand(h)h_{item(h)}}{s_{item(h)}} \cdot (deadline(h) - deadline(leftwing(h)) + 1) \tag{36}$$

for $h \in \mathcal{VL}$.

Suppose now that we have chosen a node, say $h \in \mathcal{VL}$. The next decision to be made is whether the demand represented by that node should be shifted to the left or to the right. Basically, we do a random choice again where π_{left} and $\pi_{right} = 1 - \pi_{left}$, respectively, are probability values for doing a left or a right shift. Before we give a formal definition of these probabilities, let us motivate some properties. Remember that we perform full–size shift operations, i.e.

$$\Delta t_{left} \stackrel{def}{=} deadline(h) - \max\{deadline^{I}_{LB}(h), deadline^{II}_{LB}(h)\} \tag{37}$$

is the number of periods a demand would be shifted to the left, and

$$\Delta t_{right} \stackrel{def}{=} \min\{deadline^{I}_{UB}(h), deadline^{II}_{UB}(h)\} - deadline(h) \tag{38}$$

is the number of periods a demand would be shifted to the right. Note, $\Delta t_{left} \geq 0$ and $\Delta t_{right} \geq 0$ remains true. To ease the notation, let $Deadline(h)$ denote the period in which the production that meets the (internal or external) demand represented by node h is actually needed, i.e.

$$Deadline(h) \stackrel{def}{=} \begin{cases} deadline(succ(h)) & \text{, if } succ(h) \neq -1 \\ deadline^{L4L}(h) & \text{, otherwise} \end{cases}. \tag{39}$$

To start with, assume that $\Delta t_{left} + \Delta t_{right} > 0$ holds. This is the case in which the demand of the selected node h can indeed be shifted to some direction. The conditions

$$\Delta t_{left} = 0 \Rightarrow \pi_{left} = 0 \tag{40}$$

and

$$\Delta t_{right} = 0 \Rightarrow \pi_{right} = 0 \tag{41}$$

then should hold to enforce that the demand is shifted if it is possible to do so. Another relevant point is that left shift operations cause additional holding costs and thus right shifts should be preferred if Δt_{right} exceeds Δt_{left}. Mathematically, this means to fulfill the condition

$$\Delta t_{right} \geq \Delta t_{left} \Rightarrow \pi_{right} > \pi_{left}. \tag{42}$$

Furthermore, a key idea of the shifting of demands is to facilitate the building of lots, hence a large value Δt_{left} or Δt_{right}, respectively, should result in a high probability π_{left} or π_{right}. Eventually, the importance of deciding for a left or a right shift decreases as the difference $Deadline(h) - deadline(h)$ increases. A definition of π_{left} (and π_{right}) that meets all the points mentioned above is

$$\pi_{left} \stackrel{def}{=} \frac{1}{1 + \frac{\sum_{\Delta t=1}^{\Delta t_{right}} \frac{1}{Deadline(h)-deadline(h)-\Delta t+1}}{\sum_{\Delta t=1}^{\Delta t_{left}} \frac{1}{Deadline(h)-deadline(h)+\Delta t+1}}}, \tag{43}$$

if $\Delta t_{left} > 0$, and $\pi_{left} = 0$, otherwise.

The case $\Delta t_{left} + \Delta t_{right} = 0$ virtually means that the demand of the selected node cannot be shifted to any direction. But we do not give up soon and decide if we would like to shift that demand to the left or to the right by means of a random choice again. Assume that we decided for a left shift. We then perform recursive full–size left shift operations with all preceding nodes in the gozinto–tree starting with the leafs of the tree. Afterwards, evaluating Δt_{left} again may give a positive value. If this is so, we shift the demand to the left and we are done. If Δt_{left} still is zero, we keep the left shifted internal demand at its new position, but do not modify the deadline entry of the original node h. We proceed analogously when a right shift operation should be performed. This time, the successor nodes are recursively shifted to the right starting with the root node of the gozinto–tree containing node h. If we end up with a positive value Δt_{right}, the demand represented by node h is right shifted, too. Otherwise, we keep the changes but do not shift the original demand. What is left, is a definition of the probabilities π_{left} (and π_{right}) with which we decide for a left or a right shift, respectively. In contrast to the definition above, we now use

$$\pi_{left} \stackrel{def}{=} \frac{1}{1 + \frac{(deadline^{L4L}(h) - deadline(h)) demand(h) h_{item(h)}}{s_{item(h)}}}. \tag{44}$$

```
if ( at least one feasible solution has been found
     during former iterations)
   repeat SHIFTOPS times
          choose h ∈ VL.
          decide whether h should be shifted to the left or
                to the right.
       if (h should be shifted to the left)
             leftshift(h).
       else
             rightshift(h).
```

Table 3: The Data Structure Manipulation

This formula is chosen so that we face a fifty–fifty chance for a left shift or a right shift, respectively, if the costs for holding the quantity $demand(h)$ in inventory up to the initial period $deadline^{L4L}(h)$ equal the setup costs for $item(h)$. The greater the difference between the node's current deadline and its initial deadline, the more probable is a right shift of that demand.

In summary, Table 3 describes the manipulation of the data structure as it happens after each construction phase. The parameter $SHIFTOPS$ is specified by the user and determines the number of shift operations to be executed. Note, due to the definition of the initial data structure, the very first data structure manipulations actually are left shift operations only. Since shifting demand to the left bears the risk of infeasibility, we only enter the data structure manipulation phase if at least one feasible solution has been found in previous construction phases.

Table 4 gives a precise definition of how to shift the demand represented by a node $h \in \mathcal{VL}$ to the left.

$\Delta t_{left} :=$ $deadline(h)$
$- \max\{deadline^{I}_{LB}(h), deadline^{II}_{LB}(h)\}.$
if $(\Delta t_{left} = 0)$
$treeleftshift(h).$
else
if ($\forall deadline(h) - \Delta t_{left} \leq t < deadline(h)$:

$$\sum_{\tau=1}^{t} C_{m_{item(h)}\tau} \geq p_{item(h)} demand(h) + \sum_{\substack{k \in \mathcal{L} \\ deadline(k) \leq t \\ m_{item(k)} = m_{item(h)}}} p_{item(k)} demand(k))$$

$deadline(h) := \max\{deadline^{I}_{LB}(h), deadline^{II}_{LB}(h)\}.$

Table 4: The Left Shift Procedure $leftshift(h)$

Table 5 provides the details of shifting the demand represented by node h and all the internal demand for fulfilling it to the left.

if $(preds(h) \neq \emptyset)$
for $k \in preds(h)$
$treeleftshift(k).$
$\Delta t_{left} :=$ $deadline(h)$
$- \max\{deadline^{I}_{LB}(h), deadline^{II}_{LB}(h)\}.$
if $(\Delta t_{left} \neq 0)$
if ($\forall deadline(h) - \Delta t_{left} \leq t < deadline(h)$:

$$\sum_{\tau=1}^{t} C_{m_{item(h)}\tau} \geq p_{item(h)} demand(h) + \sum_{\substack{k \in \mathcal{L} \\ deadline(k) \leq t \\ m_{item(k)} = m_{item(h)}}} p_{item(k)} demand(k))$$

$deadline(h) := \max\{deadline^{I}_{LB}(h), deadline^{II}_{LB}(h)\}.$

Table 5: The Left Shift Procedure $treeleftshift(h)$

Similarly, Table 6 provides the details of how to shift the demand represented by a node $h \in \mathcal{VL}$ to the right.

$\Delta t_{right} :=$ $\min\{deadline^{I}_{UB}(h), deadline^{II}_{UB}(h)\}$
$-deadline(h).$
if $(\Delta t_{right} = 0)$
$treerightshift(h).$
else
$deadline(h) := \min\{deadline^{I}_{UB}(h), deadline^{II}_{UB}(h)\}.$

Table 6: The Right Shift Procedure $rightshift(h)$

See Table 7 for a specification of shifting the demand represented by node h and all its successors to the right.

3.3 Construction of Production Plans

We are now in the position to describe the details of the construction of a production plan. The construction scheme is a backward oriented procedure which schedules items period by period starting with period T and ending with period one.

```
if (succ(h) ≠ −1)
    treerightshift(succ(h)).
Δt_right := min{deadline^I_UB(h), deadline^II_UB(h)}
            −deadline(h).
if (Δt_right ≠ 0)
    deadline(h) := min{deadline^I_UB(h), deadline^II_UB(h)}.
```

Table 7: The Right Shift Procedure $treerightshift(h)$

Roughly speaking, when period t is the current focus of attention, the scheme works as follows: First, the setup state at the end of period t, which is the beginning of period $t+1$, of course, is chosen for each machine. Second, items are scheduled at the beginning of period $t+1$ such that the setup state just determined is respected. Third, items are scheduled at the end of period t. Afterwards period $t-1$ is the new focus of attention and we return to the first step.

To define the details, we choose here a recurrent representation which enables us to develop the underlying ideas in a stepwise fashion. Now, let us assume that *DS-construct*$(t, \Delta t, m)$ is the procedure to be defined and $t + \Delta t$ is the period and m is the machine under concern. $\Delta t \in \{0, 1\}$ where $\Delta t = 1$ indicates that the setup state for machine m at the beginning of period $t + 1$ is to be fixed next and $\Delta t = 0$ indicates that we already have chosen a setup state at the end of period t. The symbol j_{mt} will denote the setup state for machine m at the end of period t. Assume $j_{mt} = 0$ for $m = 1, \ldots, M$ and $t = 1, \ldots, T$ initially.

Note that from the problem parameters, we can easily derive $\mathcal{P}_j$, the set of the immediate predecessors of item j, and $\bar{\mathcal{P}}_j$, the set of all predecessors of item j. Also, nr_j, the net requirement of item j, and id_{ji}, the internal demand for item i that is directly or indirectly caused by producing one unit of item j, are easy to compute.

Before the construction mechanism starts, the decision variables y_{jt} and q_{jt} are assigned zero for $j = 1, \ldots, J$, $m = 1, \ldots, M$, and $t = 1, \ldots, T$. Remember, given the values for y_{jt} and q_{jt} the values for x_{jt} and I_{jt} are implicitly defined. Furthermore, assume auxiliary variables $\tilde{d}_{jt}$ and CD_{jt} for $j = 1, \ldots, J$ and $t = 1, \ldots, T$. The former ones represent the entries in the demand matrix and thus are initialized with $\tilde{d}_{jt} = d_{jt}$. The latter ones stand for the cumulative future demand for item j which is not been met yet. As we will see, the cumulative demand can be efficiently computed while moving on from period to period. For the sake of convenience we introduce $CD_{j(T+1)} = 0$ for $j = 1, \ldots, J$. The remaining capacity of machine m in period t is denoted as RC_{mt}. Initially, $RC_{mt} = C_{mt}$ for $m = 1, \ldots, M$ and $t = 1, \ldots, T$. Additionally, we have upper bounds Q_{jt} for $j = 1, \ldots, J$ and $t = 1, \ldots, T$ where $Q_{j(T+1)} = 0$ for $j = 1, \ldots, J$ is assumed for the sake of notational convenience.

Starting with the program given in Table 8 we begin with fixing the setup states of the machines at the end of period T.

```
choose j_mT ∈ I_mT.
if (j_mT ≠ 0)
    y_{j_mT T} := 1.
if (m = M)
    DS-construct(T, 0, 1).
else
    DS-construct(T, 1, m + 1).
```

Table 8: Evaluating *DS-construct*$(T, 1, \cdot)$

A definition of $\mathcal{I}_{mt}$ and a rule to choose j_{mt} out of it will be given later. All we need to know at this point is that $\mathcal{I}_{mt} \subseteq \mathcal{J}_m \cup \{0\}$ for $m = 1, \ldots, M$ and $t = 1, \ldots, T$ is the set of items among which items are chosen. Item 0 is a dummy item.

After all machines are assigned certain setup states at the end of period t, the procedure given in Table 9 takes over.

for $j \in \mathcal{J}_m$
$CD_{jt} := \min\left\{CD_{j(t+1)} + \tilde{d}_{jt}, \max\{0, nr_j - \sum_{\tau=t+1}^{T} q_{j\tau}\}\right\}$.
$Q_{jt} := \min\{\ Q_{j(t+1)} + \sum_{\substack{h \in \mathcal{L} \\ item(h)=j \\ deadline(h)=t}} demand(h),$
$\max\{0, nr_j - \sum_{\tau=t+1}^{T} q_{j\tau}\}\}$.
if $(j_{mt} \neq 0)$
$q_{j_{mt}t} := \min\left\{CD_{j_{mt}t}, Q_{j_{mt}t}, \frac{RC_{mt}}{p_{j_{mt}}}\right\}$.
$Q_{j_{mt}t} := Q_{j_{mt}t} - q_{j_{mt}t}$.
$CD_{j_{mt}t} := CD_{j_{mt}t} - q_{j_{mt}t}$.
$RC_{mt} := RC_{mt} - p_{j_{mt}} q_{j_{mt}t}$.
for $i \in \mathcal{P}_{j_{mt}}$
if $(t - v_i > 0$ and $q_{j_{mt}t} > 0)$
$\tilde{d}_{i(t-v_i)} := \tilde{d}_{i(t-v_i)} + a_{ij_{mt}} q_{j_{mt}t}$.
if $(m = M)$
DS-construct$(t-1, 1, 1)$.
else
DS-construct$(t, 0, m+1)$.

Table 9: Evaluating *DS-construct*$(t, 0, \cdot)$ where $1 \leq t \leq T$

Note, the production quantity $q_{j_{mt}t}$ is not only constrained by the available capacity and the cumulative demand for item j_{mt} but also by the upper bound $Q_{j_{mt}t}$ which is derived from the data structure Γ^{DS} (see also (19)).

Table 10 specifies how to evaluate calls of the form *DS-construct*$(t, 1, \cdot)$ where $1 \leq t < T$.

choose $j_{mt} \in \mathcal{I}_{mt}$.
if $(j_{mt} \neq 0)$
$y_{j_{mt}t} := 1$.
if $(j_{mt} \neq j_{m(t+1)})$
$q_{j_{mt}(t+1)} := \min\left\{CD_{j_{mt}(t+1)}, Q_{j_{mt}(t+1)}, \frac{RC_{m(t+1)}}{p_{j_{mt}}}\right\}$.
$Q_{j_{mt}(t+1)} := Q_{j_{mt}(t+1)} - q_{j_{mt}(t+1)}$.
$CD_{j_{mt}(t+1)} := CD_{j_{mt}(t+1)} - q_{j_{mt}(t+1)}$.
$RC_{m(t+1)} := RC_{m(t+1)} - p_{j_{mt}} q_{j_{mt}(t+1)}$.
for $i \in \mathcal{P}_{j_{mt}}$
if $(t + 1 - v_i > 0$ and $q_{j_{mt}(t+1)} > 0)$
$\tilde{d}_{i(t+1-v_i)} := \tilde{d}_{i(t+1-v_i)} + a_{ij_{mt}} q_{j_{mt}(t+1)}$.
try to increase the capacity utilization in period $t+1$.
if $(m = M)$
DS-construct$(t, 0, 1)$.
else
DS-construct$(t, 1, m+1)$.

Table 10: Evaluating *DS-construct*$(t, 1, \cdot)$ where $1 \leq t < T$

An important point is the part which tries to increase the capacity utilization in period $t+1$. Table 11 provides more details. For the sake of notational convenience, let us assume that $CD_{0(t+1)} = 0$, $Q_{0(t+1)} = 0$, $q_{0(t+1)} = 0$, and $p_0 = 0$. Testing whether or not $j_{mt} \neq 0$ and $j_{m(t+1)} \neq 0$ to ensure well-defined variable accesses can then be left out.

$\hat{t} := t$.
while ($(CD_{j_{mt}(t+1)} > Q_{j_{mt}(t+1)}$ or $CD_{j_{m(t+1)}(t+1)} > Q_{j_{m(t+1)}(t+1)})$
and $RC_{m(t+1)} > 0$ and $\hat{t} \geq 1$)
if $(\exists h \in \mathcal{L}:\ (item(h) = j_{mt} \vee item(h) = j_{m(t+1)})$
$\wedge deadline(h) = \hat{t}$
$\wedge \min\{\ deadline^{I}_{UB}(h),$
$deadline^{II}_{UB}(h)\} \geq t+1$
$\wedge CD_{item(h)(t+1)} > Q_{item(h)(t+1)})$
$deadline(h) := t+1$.
$Q_{item(h)(t+1)} := \min\{\ Q_{item(h)(t+1)} + demand(h),$
$\max\{0, nr_{item(h)} - \sum_{\tau=t+1}^{T} q_{item(h)\tau}\}\}$.
$\Delta q_{item(h)(t+1)} := \min\{\ CD_{item(h)(t+1)}, Q_{item(h)(t+1)},$
$\frac{RC_{m(t+1)}}{p_{item(h)}}\}$.
$q_{item(h)(t+1)} := q_{item(h)(t+1)} + \Delta q_{item(h)(t+1)}$.
$Q_{item(h)(t+1)} := Q_{item(h)(t+1)} - \Delta q_{item(h)(t+1)}$.
$CD_{item(h)(t+1)} := CD_{item(h)(t+1)} - \Delta q_{item(h)(t+1)}$.
$RC_{m(t+1)} := RC_{m(t+1)} - p_{item(h)} \Delta q_{item(h)(t+1)}$.
for $i \in \mathcal{P}_{item(h)}$
if $(t+1-v_i > 0$ and $\Delta q_{item(h)(t+1)} > 0)$
$\tilde{d}_{i(t+1-v_i)} := \tilde{d}_{i(t+1-v_i)}$
$+ a_{i(item(h))} \Delta q_{item(h)(t+1)}$.
else
$\hat{t} := \hat{t} - 1$.

Table 11: A Procedure to Increase the Capacity Utilization in Period $t+1$

The idea behind this piece of code is the following: Suppose, we have scheduled two items j_{mt} and $j_{m(t+1)}$ on machine m in period $t+1$ where $q_{j_{mt}(t+1)}$ and $q_{j_{m(t+1)}(t+1)}$, respectively, are the corresponding production quantities. It may then happen that the remaining capacity $RC_{m(t+1)}$ of machine m is positive. Furthermore, assume that in period $t+1$ the cumulative demand for item j_{mt}, or $j_{m(t+1)}$, or both is not totally met. Such situations may occur due to left shift operations which have modified the initial data structure. Since the machine is now set up for these items anyhow, there seems to be no reason why we should not use the capacity that is left over in period $t+1$ to fulfill some additional demand. For the reason of consistency, we try to manipulate the data structure in order to shift some demand back to the right. To do so, we scan the data structure for nodes that represent demand for these two items and, if feasible, perform right shifts to make additional production quantities $\Delta q_{j_{mt}(t+1)}$ or $\Delta q_{j_{m(t+1)}(t+1)}$, respectively, available in period $t+1$ until all cumulative demand is fulfilled or until no free capacity remains. As a side effect of increasing the deadline entry of a node, the upper bounds for the deadline entries of some other nodes may change giving the opportunity to set their deadline value to $t+1$, too. Thus, we consider the nodes in a certain order. That is, nodes with a high current *deadline*-value are tested before nodes with a low value are, because shifting the demand that is represented by the latter ones to the right does not have an impact on whether or not demand that is represented by the former ones can be right shifted. The auxiliary variable $\hat{t}$ is used to implement this aspect. However, this defines only a partial order among the nodes. In our implementation, remaining ties are broken by favoring nodes with a high label.

Turning back to the piece of code in Table 9 now, we execute the code given in Table 12 when *DS-construct* $(0,1,\cdot)$ is called.

---

if $(j_{m0} \neq j_{m1})$
  $q_{j_{m0}1} := \min\left\{CD_{j_{m0}1}, \frac{RC_{m1}}{p_{j_{m0}}}\right\}$.
  $CD_{j_{m0}1} := CD_{j_{m0}1} - q_{j_{m0}1}$.
if $(m = M)$
  DS-construct$(0,0,1)$.
else
  DS-construct$(0,1,m+1)$.

Table 12: Evaluating *DS-construct*$(0,1,\cdot)$

A call to *construct*$(0,0,\cdot)$ terminates the construction phase. What is left is a final feasibility test where

$$nr_j = \sum_{t=1}^{T} q_{jt} \tag{45}$$

must hold for $j = 1,\ldots,J$ for being a feasible solution. Eventually, the objective function value of a feasible solution can be determined.

It should be emphasized again that the construction scheme described above does not necessarily generate an optimum solution. It does not even guarantee to find a feasible solution if there exists one. But, as the computational study will show, the method that will be derived from the construction scheme will find a (good) feasible solution for over 99% of the instances in the test-bed.

The open question that remains to be discussed is the way an item j_{mt} is chosen out of $\mathcal{I}_{mt}$, and $\mathcal{I}_{mt}$ itself needs to be defined, of course (see Tables 8 and 10). The set of items machine $m = 1,\ldots,M$ may be set up for at the end of period $t = 1,\ldots,T$ can be defined as the set of items with demand in period t or $t+1$, and with some production being allowed regarding the information in the data structure. More formally,

$$\begin{aligned}\mathcal{I}_{mt} \stackrel{def}{=} & \{j \in \mathcal{J}_m \mid CD_{j(t+1)} + \tilde{d}_{jt} > 0\} \\ & \cap \{j \in \mathcal{J}_m \mid Q_{j(t+1)} > 0 \vee \exists h \in \mathcal{L} : (\ item(h) = j \wedge deadline(h) = t)\}.\end{aligned} \tag{46}$$

As before, we choose $j_{mt} = 0$, if $\mathcal{I}_{mt} = \emptyset$.

For selecting an item $j_{mt} \in \mathcal{I}_{mt}$ we make a random choice where the probability to choose an item $j \in \mathcal{I}_{mt}$ is defined on the basis of a priority value

$$\pi_{jt} \stackrel{def}{=} h_j Q_{j(t+1)} \tag{47}$$

which is an easy–to–compute estimate of the holding costs that are charged if item j is not scheduled in period $t+1$. If

$$\sum_{j \in \mathcal{I}_{mt}} \pi_{jt} = 0$$

a random choice with uniform distribution is done. Otherwise,

$$\frac{\pi_{jt}}{\sum_{i \in \mathcal{I}_{mt}} \pi_{it}}$$

is the probability to choose j out of $\mathcal{I}_{mt}$.

4 Computational Study

All test are conducted on a Pentium computer with 120 MHz. The demand shuffle algorithm is implemented in C running under a LINUX operating system. We perform 1,000 iterations, i.e. 1,000 production plans are (tried to be) generated, and choose the method parameter $SHIFTOPS = 10$.

Two kinds of tests are done in this study. First, we tested the performance on large instances. Unfortunately, optimum results or good lower bounds are not available for these instances. Hence, all we can do is to compare the results of the demand shuffle method with those obtained with other procedures. As already stated in the introduction, other procedures for multi–level lot sizing and scheduling than those developed by ourselves do not exist.

Second, we tested the performance on small instances for which optimum results are known. Since the first part of the study reveals that the demand shuffle algorithm dominates the other heuristics, the second part of the study now focuses on demand shuffle and provides in–depth information on what problem parameters have a significant impact on the performance.

In our studies, five problem parameters are investigated:

- M, the number of machines.
- C, the complexity of the gozinto–structure. C is from the interval $[0,1]$ where values close to 1 indicate many precedence constraints among the items. We refer to [24] for a formal definition.
- $(T_{macro}, T_{micro}, T_{idle})$, the demand pattern. We define $T = T_{macro} \cdot T_{micro}$. External demand occurs only at the end of macro periods which are subdivided into T_{micro} micro periods each. In the first T_{idle} macro periods, there is no demand.
- $COSTRATIO$, the ratio of setup and holding costs.
- U, the capacity utilization.

4.1 Tests with Large Instances

Since large instances are those which are of utmost importance in real–world applications, the demand shuffle heuristic must proof its dominance over other procedures especially for large instances. That is what we like to find out in this subsection with a preliminary computational study.

The test–bed that we use consists of nine parameter level combinations of J and T which are assumed to be an adequate measure for the size of an instance. The number of machines M is chosen to be $M = \frac{2}{5}J$. Gozinto–structures have a complexity of $C = 0.2$. The demand pattern is defined by $(T_{macro}, T_{micro}, T_{idle}) = (\frac{T}{5}, 5, 2)$. The ratio of setup and holding costs is $COSTRATIO = 900$. Since the formerly developed methods have problems in finding feasible solutions if the capacity utilization is high, we choose $U = 50$ to give them a chance to find feasible solutions we may compare the results of the demand shuffle procedure with. For each of the nine parameter level combination we generated five instances at random. This gives a total of 45 large instances.

As a performance measure we consider the average deviation from the demand shuffle result where the deviation is computed using

$$deviation \stackrel{def}{=} 100 \frac{UB_H - UB_{DS}}{UB_{DS}} \tag{48}$$

with UB_H being the upper bound of an instance that is computed by means of the heuristic $H \in \{DS, RR, CA, GA, TS\}$. The short–hand notation DS stands for demand shuffle, RR for randomized regret based sampling [21, 25], CA for cellular automaton [24], GA for genetic algorithm [26], and TS for disjunctive arc based tabu search [22]. In this definition we use the result of the demand shuffle procedure as a point of reference, because (good) lower bounds are not available for large instances.

Table 13 summarizes the results of different heuristics when applied to instances of different size. For each combination of heuristic and parameter levels for J and T this table provides the average deviation from the demand shuffle result. The number of instances for which a feasible solution is found is given, too. The demand shuffle method is able to find a feasible solution for every large instance in the test–bed.

J	T	RR	CA	GA	TS
5	50	-4.03 [5]	22.52 [5]	24.38 [5]	14.39 [5]
10	50	3.47 [5]	42.37 [5]	31.35 [5]	42.57 [5]
20	50	17.47 [5]	38.92 [1]	29.24 [5]	— [0]
5	100	-13.09 [5]	58.07 [5]	27.68 [5]	18.03 [5]
10	100	-1.38 [5]	38.56 [3]	30.80 [5]	— [0]
20	100	11.22 [5]	34.74 [1]	30.81 [5]	— [0]
5	500	-16.02 [5]	59.86 [5]	25.72 [5]	— [0]
10	500	-4.40 [5]	31.47 [4]	28.72 [5]	— [0]
20	500	13.81 [5]	27.64 [2]	24.26 [5]	— [0]

Table 13: Performance for Large Instances

It turns out that only the randomized regret based sampling method and the genetic algorithm also find feasible solutions for all instances in the test–bed. The capability of the disjunctive arc based tabu search method to find feasible solutions for large instances is disastrous. In terms of average deviation from the demand shuffle result, both, the cellular automaton and the genetic algorithm give disappointing results, though the genetic algorithm shows better performance in almost all cases. Remarkable to note, the randomized regret based method reveals itself as a competitive candidate for large instances. Especially for instances with a small number of items, i.e. $J \leq 10$, this method convinces. However, if the number of items is large, i.e. $J > 10$, the results of the demand shuffle procedure cannot be reached.

4.2 Tests with Small Instances

The demand shuffle procedure is now tested on the basis of the 1,033 small PLSP–instances used in [25, 26] which are systematically generated using a full factorial design. These instances are small enough to be solved with standard software and large enough to be non–trivial.

The effect of changing certain parameter levels is analyzed on the basis of aggregated data. Table 14 shows what happens if the number of machines is varied. Only slight differences can be measured for the average deviation from the optimum. Remarkable to note is the infeasibility ratio. For all instances with $M = 1$ a feasible solution is found and only 1.16% of the instances with $M = 2$ remain unresolved. The run–time increases if the number of machines is increased.

	$M = 1$	$M = 2$
Average Deviation	6.28	6.92
Infeasibility Ratio	0.00	1.16
Average Run–Time	0.38	0.50

Table 14: The Impact of the Number of Machines on the Performance

The complexity of the gozinto–structure and its impact on the performance is studied in Table 15. It can be seen that a high complexity gives significantly larger deviations from the optimum than a low complexity. The infeasibility ratio, however, is almost unaffected. Furthermore, the more complex the gozinto–structures are, the more run–time is needed to solve the instances.

	$C = 0.2$	$C = 0.8$
Average Deviation	5.43	7.80
Infeasibility Ratio	0.57	0.59
Average Run–Time	0.41	0.47

Table 15: The Impact of the Gozinto–Structure Complexity on the Performance

Table 16 presents different results for different demand patterns. Though the average deviation from the optimum varies among the parameter levels, there is no clear tendency which could prove that sparsely–filled demand matrices give lower or higher average deviations than matrices with many non–zeroes. With respect to the run–time, sparsely–filled demand matrices give best results. The infeasibility ratio shows that only for $(T_{macro}, T_{micro}, T_{idle}) = (5, 2, 2)$ some instances remain unresolved.

	$(T_{macro}, T_{micro}, T_{idle}) =$		
	$(10, 1, 5)$	$(5, 2, 2)$	$(1, 10, 0)$
Average Deviation	7.07	5.80	6.83
Infeasibility Ratio	0.00	1.85	0.00
Average Run–Time	0.53	0.50	0.30

Table 16: The Impact of the Demand Pattern on the Performance

The effect of varying the cost structure is analyzed in Table 17. If the ratio of setup and holding costs is high, the demand shuffle method performs best in terms of the average deviation from the optimum. The infeasibility ratio is unaffected by changes in the cost structure. Surprisingly, the average run–time declines if setup costs grow.

	$COSTRATIO =$		
	5	150	900
Average Deviation	9.68	5.25	4.85
Infeasibility Ratio	0.58	0.58	0.58
Average Run–Time	0.46	0.45	0.40

Table 17: The Impact of the Cost Structure on the Performance

To see if the capacity utilization has an impact on the performance, let us have a look at Table 18. We see that if the capacity utilization inclines, the average deviation from the optimum does so, too. Those few instances for which no feasible solution is detected, are instances with a high capacity utilization. The run–time is more or less unaffected.

In summary, only six out of the 1,033 instances in the test–bed remain unresolved which is an amazing result. The overall infeasibility ratio is 0.58%. The average run–time is 0.44 CPU–seconds. The overall average deviation from the optimum objective function value is an exciting 6.60%.

Most of the intelligence of the demand shuffle heuristic lies in the way a node is selected for a shift operation, in the way to decide whether a left or a right shift should be performed, and in the way we select a setup state for a machine. To show that the presented rules do indeed make a contribution, Table 19 shows what happens if we select a node with uniform distribution and try to shift it then, assume a fifty–fifty chance for a left or a right shift, and select the setup state of a machine with uniform distribution among the valid candidates.

In terms of the run–time performance it turns out that we have to pay a cheap price only for using intelligent decision rules. The advantage of these rules is that on the basis of both, the average deviation from the optimum and the infeasibility ratio, a better performance is gained. However, using no intelligence in the demand shuffle method still gives very good results when compared with other heuristics as shown in Table 20.

	$U = 30$	$U = 50$	$U = 70$
Average Deviation	5.15	6.76	8.06
Infeasibility Ratio	0.00	0.00	1.88
Average Run–Time	0.45	0.45	0.42

Table 18: The Impact of the Capacity Utilization on the Performance

	Average Deviation	Infeasibility Ratio	Average Run–Time
Intelligent Decision Rules	6.60	0.58	0.44
Simple Decision Rules	7.71	0.87	0.40

Table 19: The Impact of the Decision Rules on the Performance

5 Conclusion

In this paper we have dealt with multi–level lot sizing and scheduling which is a short–term planning problem in the presence of lot production. We pointed out that traditional MRP II logic does not satisfy, because important aspects such as scarce capacities and multi–level gozinto–structures are not taken into account appropriately. Consequently, we formulated a mixed–integer model to give a precise statement of what is to be solved. By having a look at the literature we found that this approach is indeed new, because no other publications (out of some of our own) on this thorny problem exist. A so–called demand shuffle procedure has then been introduced to solve the problem heuristically. It is an extension of the single–machine procedure specified in [23] recently. A computational study revealed that demand shuffle clearly outperforms the existing methods and thus defines the new state–of–the–art.

Acknowledgement

We are indebted to Andreas Drexl for his steady support.

References

[1] AFENTAKIS, P., GAVISH, B., (1986), Optimal Lot–Sizing Algorithms for Complex Product Structures, Operations Research, Vol. 34, pp. 237–249

[2] AFENTAKIS, P., GAVISH, B., KARMARKAR, U., (1984), Computationally Efficient Optimal Solutions to the Lot–Sizing Problem in Multistage Assembly Systems, Management Science, Vol. 30, pp. 222–239

[3] AGGARWAL, A., PARK, J.K., (1993), Improved Algorithms for Economic Lot–Size Problems, Operations Research, Vol. 41, pp. 549–571

[4] ARKIN, E., JONEJA, D., ROUNDY, R., (1989), Computational Complexity of Uncapacitated Multi–Echelon Production Planning Problems, Operations Research Letters, Vol. 8, pp. 61–66

[5] BITRAN, G.R., MATSUO, H., (1986), Approximation Formulations for the Single–Product Capacitated Lot Size Problem, Operations Research, Vol. 34, pp. 63–74

[6] BRÜGGEMANN, W., JAHNKE, H., (1994), DLSP for 2–Stage Multi–Item Batch Production, International Journal of Production Research, Vol. 32, pp. 755–768

[7] DAUZÈRE–PÉRES, S., LASSERRE, J.B., (1994), An Integrated Approach in Production Planning and Scheduling, Lecture Notes in Economics and Mathematical Systems, Berlin, Springer, Vol. 411

	Average Deviation	Infeasibility Ratio	Average Run–Time
RR	10.33	9.68	0.45
CA	10.94	12.97	0.54
GA	19.89	17.52	0.10
TS	17.59	35.62	0.50
DS	6.60	0.58	0.44

Table 20: Summary of Heuristic Procedures

[8] DIABY, M., BAHL, H.C., KARWAN, M.H., ZIONTS, S., (1992), A Lagrangean Relaxation Approach for Very–Large–Scale Capacitated Lot–Sizing, Management Science, Vol. 38, pp. 1329–1340

[9] DINKELBACH, W., (1964), Zum Problem der Produktionsplanung in Ein– und Mehrproduktunternehmen, Würzburg, Physica, 2nd edition

[10] DREXL, A., HAASE, K., (1995), Proportional Lotsizing and Scheduling, International Journal of Production Economics, Vol. 40, pp. 73–87

[11] DREXL, A., KIMMS, A., (1997), Lot Sizing and Scheduling — Survey and Extensions, European Journal of Operational Research, Vol. 99, pp. 221–235

[12] DREXL, A., KIMMS, A., (eds.), (1998), Beyond Manufacturing Resource Planning (MRP II) — Advanced Models and Methods for Production Planning, Berlin, Springer

[13] EPPEN, G.D., MARTIN, R.K., (1987), Solving Multi–Item Capacitated Lot–Sizing Problems Using Variable Redefinition, Operations Research, Vol. 35, pp. 832–848

[14] FEDERGRUEN, A., TZUR, M., (1991), A Simple Forward Algorithm to Solve General Dynamic Lot Sizing Models with n Periods in $O(n \log n)$ or $O(n)$ Time, Management Science, Vol. 37, pp. 909–925

[15] FLEISCHMANN, B., (1990), The Discrete Lot–Sizing and Scheduling Problem, European Journal of Operational Research, Vol. 44, pp. 337–348

[16] HAASE, K., (1994), Lotsizing and Scheduling for Production Planning, Lecture Notes in Economics and Mathematical Systems, Vol. 408, Berlin, Springer

[17] HELBER, S., (1995), Lot Sizing in Capacitated Production Planning and Control Systems, OR Spektrum, Vol. 17, pp. 5–18

[18] JONEJA, D., (1991), Multi–Echelon Assembly Systems with Non–Stationary Demands: Heuristics and Worst Case Performance Bounds, Operations Research, Vol. 39, pp. 512–518

[19] KARMARKAR, U.S., KEKRE, S., KEKRE, S., (1987), The Deterministic Lotsizing Problem with Startup and Reservation Costs, Operations Research, Vol. 35, pp. 389–398

[20] KARMARKAR, U.S., SCHRAGE, L., (1985), The Deterministic Dynamic Product Cycling Problem, Operations Research, Vol. 33, pp. 326–345

[21] KIMMS, A., (1996), Multi–Level, Single–Machine Lot Sizing and Scheduling (with Initial Inventory), European Journal of Operational Research, Vol. 89, pp. 86–99

[22] KIMMS, A., (1996), Competitive Methods for Multi–Level Lot Sizing and Scheduling: Tabu Search and Randomized Regrets, International Journal of Production Research, Vol. 34, pp. 2279–2298

[23] KIMMS, A., (1997), Demand Shuffle — A Method for Multi–Level Proportional Lot Sizing and Scheduling, Naval Research Logistics, Vol. 44, pp. 319–340

[24] KIMMS, A., (1997), Multi–Level Lot Sizing and Scheduling — Methods for Capacitated, Dynamic, and Deterministic Models, Heidelberg, Physica

[25] KIMMS, A., (1998), A Genetic Algorithm for Multi–Level, Multi–Machine Lot Sizing and Scheduling, Working Paper, University of Kiel

[26] KIMMS, A., DREXL, A., (1998), Proportional Lot Sizing and Scheduling: Some Extensions, Networks, to appear

[27] KIRCA, Ö., KÖKTEN, M., (1994), A New Heuristic Approach for the Multi–Item Dynamic Lot Sizing Problem, European Journal of Operational Research, Vol. 75, pp. 332–341

[28] LASDON, L.S., TERJUNG, R.C., (1971), An Efficient Algorithm for Multi–Item Scheduling, Operations Research, Vol. 19, pp. 946–969

[29] LASSERRE, J.B., (1992), An Integrated Model for Job-Shop Planning and Scheduling, Management Science, Vol. 38, pp. 1201–1211

[30] MAES, J., MCCLAIN, J.O., VAN WASSENHOVE, L.N., (1991), Multilevel Capacitated Lotsizing Complexity and LP–Based Heuristics, European Journal of Operational Research, Vol. 53, pp. 131–148

[31] POCHET, Y., WOLSEY, L.A., (1991), Solving Multi–Item Lot–Sizing Problems Using Strong Cutting Planes, Management Science, Vol. 37, pp. 53–67

[32] ROUNDY, R.O., (1993), Efficient, Effective Lot Sizing for Multistage Production Systems, Operations Research, Vol. 41, pp. 371–385

[33] SALOMON, M., KROON, L.G., KUIK, R., VAN WASSENHOVE, L.N., (1991), Some Extensions of the Discrete Lotsizing and Scheduling Problem, Management Science, Vol. 37, pp. 801–812

[34] SALOMON, M., KUIK, R., VAN WASSENHOVE, L.N., (1993), Statistical Search Methods for Lotsizing Problems, Annals of Operations Research, Vol. 41, pp. 453–468

[35] TEMPELMEIER, H., DERSTROFF, M., (1996), A Lagrangean–Based Heuristic for Dynamic Multi–Level Multi–Item Constrained Lotsizing with Setup Times, Management Science, Vol. 42, pp. 738–757

[36] VÖRÖS, J., (1995), Setup Cost Stability Region for the Multi–Level Dynamic Lot Sizing Problem, European Journal of Operational Research, Vol. 87, pp. 132–141

[37] WAGELMANS, A., VAN HOESEL, S., KOLEN, A., (1992), Economic Lot Sizing: An $O(n \log n)$ Algorithm that Runs in Linear Time in the Wagner–Whitin Case, Operations Research, Vol. 40, pp. S145–S156

[38] WAGNER, H.M., WHITIN, T.M., (1958), Dynamic Version of the Economic Lot Size Model, Management Science, Vol. 5, pp. 89–96

[39] ZÄPFEL, G., MISSBAUER, H., (1993), New Concepts for Production Planning and Control, European Journal of Operational Research, Vol. 67, pp. 297–320

A An Example

Consider the gozinto–structure given in Figure 1 and the parameters in Table 21. Furthermore, assume $M = 2$, $m_1 = m_4 = 1$, $m_2 = m_3 = 2$, and $C_{1t} = C_{2t} = 15$ for $t = 1, \ldots, 10$. For illustrating the construction of a production plan we do not need any information about setup and holding costs.

Suppose, the data structure Γ^{DS} is given to be

$$
\begin{aligned}
\Gamma^{DS} = \{&(1,7,7,20,0,1,-1, \quad \{(2,6,6,20,1,7,0,\\
&\qquad\qquad\qquad\qquad \{(4,5,5,20,2,39,1,\emptyset)\}),\\
&\qquad\qquad\quad (3,6,6,20,3,8,0,\\
&\qquad\qquad\qquad\qquad \{(4,5,5,20,4,44,3,\emptyset)\})\})\\
&\quad (4,8,8,5,5,4,-1,\emptyset),\\
&\quad (1,10,10,20,6,1,-1,\{\ (2,9,9,20,7,7,6,\\
&\qquad\qquad\qquad\qquad \{(4,8,8,20,8,39,7,\emptyset)\}),
\end{aligned}
$$

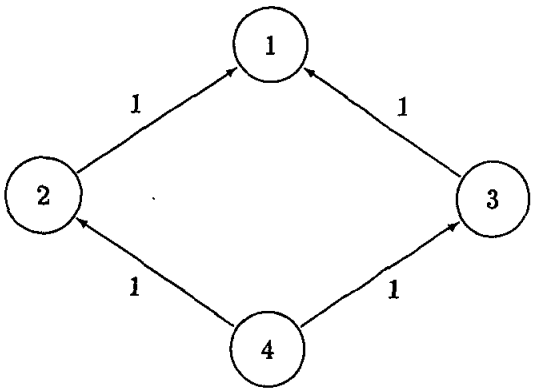

Figure 1: A Gozinto–Structure with Four Items

d_{jt}	$t=1$	...	5	6	7	8	9	10	p_j	v_j	y_{j0}	I_{j0}
$j=1$					20			20	1	1	1	0
$j=2$									1	1	1	0
$j=3$									1	1	0	0
$j=4$						5			1	1	0	0

Table 21: Parameters of the Example

$$(3,6,9,20,9,8,6,$$
$$\{(4,5,8,20,10,44,9,\emptyset)\})\})$$

}

which can be illustrated as in Table 22 where an entry of the form $L : D^{LB/UB}$ in row j and column t depicts D units of demand for item j that is to be met no later than period t. The node in the data structure that represents this demand has the label L. If the data structure is manipulated, this demand cannot be shifted to a period prior to period LB or after period UB.

Γ^{DS} $t=$	...5	6	7	8	9	10
$j=1$			$0:20^{7/7}$			$6:20^{10/10}$
$j=2$		$1:20^{6/6}$			$7:20^{9/9}$	
$j=3$		$3:20^{6/6}$				
		$9:20^{6/9}$				
$j=4$	$2:20^{5/5}$			$5:5^{8/8}$		
	$4:20^{5/5}$			$8:20^{5/8}$		
	$10:20^{5/5}$					

Table 22: A Data–Structure for the Demand Shuffle Heuristic

For instance, let us have a look at the node with label 9. It represents a demand of $demand(9) = 20$ units of item $item(9) = 3$ which is to be met no later than period $deadline(9) = 6$. Furthermore, the information in the data structure tells us that this demand is an internal demand caused by producing the demand represented by node $succ(9) = 6$ which is external demand, because $succ(6) = -1$. In turn, fulfilling the demand represented by node 9 causes internal demand that is represented by the nodes in the set $preds(9) = \{10\}$. Hence, the path from the root node of the gozinto–tree to the node 10 contains the items $item(6) = 1$, $item(9) = 3$, and $item(10) = 4$. The position of the node 10 is thus uniquely

identified by $path(10) = (\underline{1} \cdot 5 + \underline{3}) \cdot 5 + \underline{4} = 44$ which equals the position of node 4. An additional shift operation may move the demand of node 9 to no period prior to period

$$\begin{aligned}&\max\{deadline^{I}_{LB}(9), deadline^{II}_{LB}(9)\}\\ &= \max\{deadline(10) + v_4, deadline(3)\}\\ &= 6\end{aligned}$$

and to no period later than period

$$\begin{aligned}&\min\{deadline^{I}_{UB}(9), deadline^{II}_{UB}(9)\}\\ &= \min\{deadline(6) - v_3, deadline^{L4L}(9)\}\\ &= 9.\end{aligned}$$

Although it appears to be that the demand represented by the node 8 could be left shifted to period 5 next, we would not do so, because the condition (20) would no longer be valid for period 5 and machine 1, i.e.

$$\begin{aligned}&\sum_{\tau=1}^{5} C_{1\tau}\\ &= 75 \not\geq 80\\ &= p_{item(8)} demand(8) + \sum_{k \in \{2,4,10\}} p_{item(k)} demand(k),\end{aligned}$$

which indicates that no feasible solution can be found.

Table 23 provides the first few steps of a run of the construction scheme when using the data structure given above.

Step	$(t, \Delta t, m)$	$\vec{d}_{jt}$	$\vec{CD}_{j(t+\Delta t)}$	$\vec{Q}_{j(t+\Delta t)}$	$\mathcal{I}_{mt}$	j_{mt}	$q_{j_{mt}(t+\Delta t)}$
1	(10,1,1)	(20,0,0,0)	(0,0,0,0)	(0,0,0,0)	{1}	1	
2	(10,1,2)	(20,0,0,0)	(0,0,0,0)	(0,0,0,0)	$\emptyset$	0	
3	(10,0,1)	(20,0,0,0)	(20,0,0,0)	(20,0,0,0)			$q_{1T} = 15$
4	(10,0,2)	(20,0,0,0)	(5,0,0,0)	(5,0,0,0)			
5	(9,1,1)	(0,15,15,0)	(5,0,0,0)	(5,0,0,0)	{1}	1	
6	(9,1,2)	(0,15,15,0)	(5,0,0,0)	(5,0,0,0)	{2}	2	$q_{2T} = 0$
7	(9,0,1)	(0,15,15,0)	(5,0,0,0)	(5,0,0,0)			$q_{19} = 5$
8	(9,0,2)	(0,15,15,0)	(0,15,15,0)	(0,20,0,0)			$q_{29} = 15$
9	(8,1,1)	(0,5,5,20)	(0,0,15,0)	(0,5,0,0)	{4}	4	$q_{49} = 0$
10	(8,1,2)	(0,5,5,20)	(0,0,15,0)	(0,5,0,0)	{2}	2	
11	(8,0,1)	(0,5,5,20)	(0,0,15,20)	(0,5,0,25)			$q_{48} = 15$
12	(8,0,2)	(0,5,5,20)	(0,5,20,5)	(0,5,0,10)			$q_{28} = 5$
13	(7,1,1)	(20,0,0,5)	(0,0,20,5)	(0,0,0,10)	{4}	4	
14	(7,1,2)	(20,0,0,5)	(0,0,20,5)	(0,0,0,10)	$\emptyset$	0	
...							

Table 23: A Protocol of the Construction Scheme of the Demand Shuffle Procedure

A possible outcome of the construction phase is shown in Figure 2.

Some points of interest are worth to be highlighted:

Step 1: $1 \in \mathcal{I}_{1T}$, because $\vec{d}_{1T} > 0$, $deadline(6) = 10$, and $item(6) = 1$.

Step 6: Although $\vec{d}_{39} > 0$ holds, item 3 is not contained in the set $\mathcal{I}_{29}$. This is because $Q_{3T} = 0$ and there is no node h in the data structure which fulfills $deadline(h) = 9$ and $item(h) = 3$. In other words, the demand for item 3 was shifted to the left.

A case in which $Q_{j(t+1)} > 0$ and $CD_{j(t+1)} = 0$ holds for an item j and a period t does not occur in this example. It is, however, remarkable to note that we may face such situations. Assume for instance that there would not be any external demand for item 4 and that all (internal) demand is shifted leftmost (let us suppose this could be done without violating the capacity constraints (20)). Then, in Step 9, the set $\mathcal{I}_{18}$ would be empty. That is, the presented procedure would handle such cases correctly.

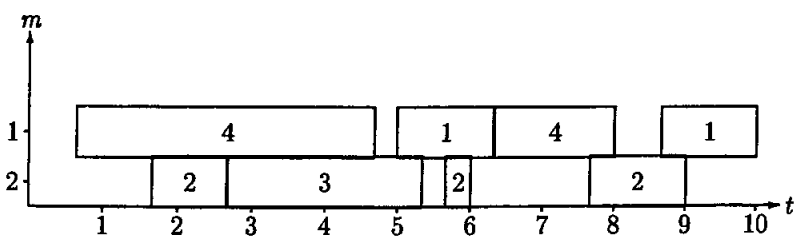

Figure 2: A Possible Outcome of the Run in the Protocol