Kimms, Alf

Demand shuffle - a novel method for multi-level proportional lot sizing and scheduling

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 355

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

No. 355

Demand Shuffle – A Novel Method for Multi-Level Proportional Lot Sizing and Scheduling

A. Kimms

November 1994

Dipl.-Inform. Alf Kimms, Lehrstuhl für Produktion und Logistik, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstraße 40, 24118 Kiel, Germany

Abstract: This contribution acquaints the reader with a model for multi-level single-machine proportional lot sizing and scheduling problems (PLSPs) that appear in the scope of short-term production planning. It is one of the first papers that deals with dynamic capacitated multi-level lot sizing and scheduling which is of great practical importance. The PLSP model refines well-known mixed-integer programming formulations for dynamic capacitated lot sizing and scheduling as for instance the DLSP or the CSLP. A special emphasis is given on a new method called „demand shuffle" to solve multi-level PLSP instances efficiently but suboptimal. Although the basic idea is very simple, it becomes clear that in the presence of precedence and capacity constraints many non-trivial details are to be concerned. Computational studies show that the presented approach decidedly improves recent results.

Keywords: Production planning, lot sizing, scheduling, multi-level, PLSP, heuristics

1 Introduction

Although being a crucial short-term production planning problem, recent research has almost neglected the development of models and methods for simultaneous lot sizing and scheduling under general multi-level precedence constraints. For most authors who consider multi-level lot sizing the scheduling aspect is apart from their focus [Afentakis et al. 1984, Afentakis and Gavish 1986, Bahl et al. 1987, Helber 1994a and 1994b, Kuik and Salomon 1990, Kuik et al. 1993, Maes et al. 1991, Roll and Karni 1991, Rosling 1986, Roundy 1993, Salomon 1991, Salomon et al. 1993, Stadtler 1994, Tempelmeier and Derstroff 1993a and 1993b, Tempelmeier and Helber 1994]. Multi-level lot sizing and scheduling literature is rare [Domschke et al. 1993, Tempelmeier 1992] and restrictive (e.g. [Brüggemann and Jahnke 1994] consider two levels only and [El-Najdawi 1994, El-Najdawi and Kleindorfer 1993, Iyogun and Atkins 1993] assume stationary demand). In summary, there seems to be no contribution to dynamic capacitated multi-level lot sizing and scheduling yet. Hence, we did some research by ourselves under the following assumptions:

(a) The planning horizon is finite and subdivided into several discrete periods of time.
(b) External demands are deterministically known and time-variant (dynamic).
(c) Shortages are not allowed (neither backorders nor stockouts).
(d) The „gozinto"-structure that defines the precedence relation among the items is a general multi-level structure without cycles.
(e) All items are to be produced on one single machine.
(f) The production facility offers a limited capacity per time period.
(g) The production speed is finite, i.e. producing one item needs an item-specific amount of the available capacity.

(h) More than one changeover must not take place within a period. In other words, no more than two different items may be manufactured within a period.

Some interesting aspects shall be discussed in a little more detail: While most of the above assumptions are more or less standard for dynamic capacitated lot sizing (and scheduling) problems, we like to point out that assumption (h) refines what is known from „classic" formulations, since the discrete lot sizing and scheduling problem (DLSP) [Fleischmann 1990, Salomon 1991] as well as the continuous setup lot sizing problem (CSLP) [Karmarkar and Schrage 1985, Salomon 1991] consider the production of at most one item per period. The single-level pendant of the problem above was first stated in [Drexl and Haase 1992, Haase 1994] who coined the name proportional lot sizing and scheduling problem (PLSP). A model with no notion of periods, i.e. a model with a continuous time axis (similar to the economic lot scheduling problem (ELSP) [Elmaghraby 1978] where stationary demand is a prerequisite), would be the extreme of dynamic models in terms of time granularity.

First efforts to solve multi-level PLSPs (assumption (d)) are reported in [Kimms 1993a] followed up by [Kimms 1993b and 1994a]. The relevance of considering multi-level structures in Manufacturing Resource Planning (MRP II) is exhaustively studied in [Drexl et al. 1993]. Multi-machine environments where the assignment of machines to items is a many-to-one mapping, i.e. in the case that some (if not all) items share a common machine but there is no freedom on which machine an item is to be manufactured, can be seen as a relaxation of assumption (e) which justifies the importance of the single-machine case. The extreme where machines are dedicated to items can be solved optimally with a greedy algorithm as shown in [Kimms 1994b].

This text is organized as follows: Chapter 2 contains a mixed-integer formulation of the multi-level PLSP in order to provide a precise definition of the problem under consideration. Next in chapter 3, we present a new method to tackle this problem in guidance with an example that helps to understand the details. A computational study in chapter 4 reveals the quality of this method. Concluding remarks then finish the paper.

2 Multi-Level Single-Machine Proportional Lot Sizing and Scheduling

The problem we are concerned about was originally formulated in [Kimms 1993a] as a mixed-integer program. For the sake of being self-contained the program should be listed here again, but first we introduce the notation used:

Decision variables:

I_{jt} is the quantity of item j held in inventory at the end of period t;

q_{jt} is the quantity of item j to be produced in period t;

x_{jt} is a (binary) variable indicating whether a setup for item j occurs in period t ($x_{jt} = 1$) or not ($x_{jt} = 0$);

y_{jt} is a binary variable indicating whether the machine is setup for item j at the end of period t ($y_{jt} = 1$) or not ($y_{jt} = 0$).

Instance specific data:

a_{ji} is the „gozinto"-factor, i.e. the quantity of item j that is needed to produce one item i;

B is a large number greater than $\frac{\max \{ C_t \mid t = 1 .. T \}}{\min \{ p_j \mid j = 1 .. J \}}$;

C_t is the capacity of the machine in period t;

d_{jt} is the (external) demand for item j in period t;

h_j are the (non-negative) costs for holding one item j one period in inventory;

I_{j0} is the initial inventory

J is the number of items;

p_j is the amount of capacity consumed by producing one item j;

s_j are the (non-negative) setup costs for item j;

S(j) is the set of successors of item j, i.e. the set of items i where $a_{ji} > 0$;

T is the number of periods;

v_j is the (integral) lead time of item j ($v_j \geq 1$);

y_{j0} is the unique initial setup state.

Using this notation, the multi-level single-machine proportional lot sizing and scheduling problem can now be described as follows:

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

subject to

$$I_{jt} = I_{j(t-1)} + q_{jt} - d_{jt} - \sum_{i \in S(j)} (a_{ji} q_{it}) \qquad (j = 1 .. J, t = 1 .. T) \qquad (2)$$

$$I_{jt} \geq \sum_{\tau=t+1}^{\min\{t+v_j,T\}} \sum_{i \in S(j)} (a_{ji} q_{i\tau}) \qquad (j = 1 .. J, t = 0 .. T-1) \qquad (3)$$

$$\sum_{j=1}^{J} y_{jt} \leq 1 \qquad (t = 1 .. T) \qquad (4)$$

$$B (y_{jt} + y_{j(t-1)}) - q_{jt} \geq 0 \qquad (j = 1 .. J, t = 1 .. T) \qquad (5)$$

$$x_{jt} - y_{jt} + y_{j(t-1)} \geq 0 \qquad (j = 1 .. J, t = 1 .. T) \qquad (6)$$

$$\sum_{j=1}^{J} (p_j q_{jt}) \leq C_t \qquad (t = 1 .. T) \qquad (7)$$

$$y_{jt} \in \{ 0 , 1 \} \qquad (j = 1 .. J, t = 1 .. T) \qquad (8)$$

$$I_{jt} \geq 0, q_{jt} \geq 0, x_{jt} \geq 0 \qquad (j = 1 .. J, t = 1 .. T) \qquad (9)$$

The objective is to minimize the sum of setup and holding costs as it is written down in (1). The inventory balances are formulated in (2) where the inventory at the end of a period contains what was in inventory the period before plus what is produced in that period minus what is used up to meet external or internal demand. To meet an internal demand, an item must be stored in inventory for at least some item-specific lead time (see (3)). With (4) the setup state of the machine is uniquely defined at the end of each period, so it is at the beginning of each period. Due to (5) production of an item may only take place in a certain period if the machine is setup for this item either at the beginning or at the end of a period or both. Those periods in which a setup occurs are spotted by (6). Note, that due to the minimize objective in combination with (9) the setup variables x_{jt} are indeed zero-one valued. Since capacity is

scarce, (7) must hold to guarantee that capacity limits are not exceeded. The setup state variables are defined as binary variables in (8) while non-negative constraints (9) are sufficient for the remaining decision variables. Subsequently, we assume the initial inventory to be equal to zero for all items. [Kimms 1993a] discusses positive initial stock levels.

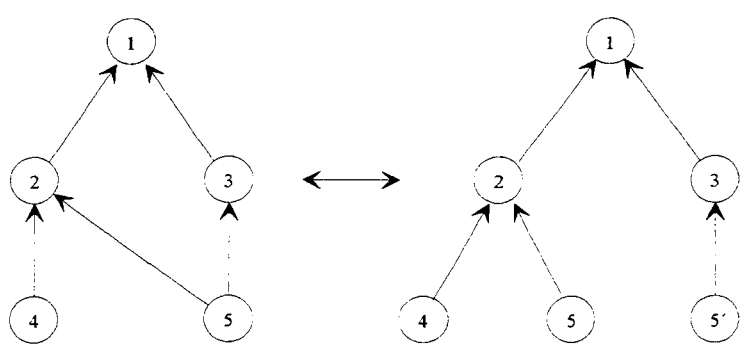
3 A Two-Phase Method

Following the lines of this chapter will provide insight into the details of a heuristic for multi-level single-machine PLSPs. As it will become clear, the basic idea of the heuristic's working principle is very simple, but, the presence of precedence and capacity constraints needs to concern about many non-trivial details. Thus, an example is used throughout this chapter helping to make things clear. The details of this example – as far as it is got to be known right now – shall be given first. Then, we describe the heuristic step by step.

3.1 The Example

Assume a „gozinto"-structure consisting of five items where the preferred way to look at this structure should be as a „gozinto"-tree rather than a graph (see Figure 1). To discriminate between those units of item 5 that are needed to produce item 2 and those that are used to produce item 3, we write 5′ instead of 5 in the latter case. Here, we have only one end item which is item 1. If we would face a structure with more than one end item, we would come up with a forest not with a single tree, of course. For our purposes let all „gozinto"-factors as well as all lead times be equal to one.

Figure 1: The „gozinto"-structure of the example



Furthermore, suppose the external demand be defined by Table 1. Other positive external demands than those three given in Table 1 shall not exist. For the sake of simplicity, we assume external demand for the end item only in our example. But in general, external demand may occur for any item.

Table 1: The external demand matrix of the example

	...	T-6	T-5	T-4	T-3	T-2	T-1	T
j = 1	...			5		15		10
j = 2	...							
j = 3	...							
j = 4	...							
j = 5	...							

3.2 Definition of the Data Structure

Besides the heuristic algorithm which is to be specified, there is a certain data structure on which this algorithm operates on and which is the aspect we begin our explanations with. This data structure we are talking about somehow looks like a representation of a production plan in that it contains all external and internal demands. Initially, it is derived from the information provided by the external demand matrix and the „gozinto"-trees. Roughly speaking, the initial data structure defines a lot-for-lot production in the absence of capacity restrictions. Table 2 shows a matrix representation of the initial data structure derived from the data of our example (upper indices can be ignored – they become meaningful as we proceed on).

Table 2: A matrix representation of the initial data structure for the example

	...	T-6	T-5	T-4	T-3	T-2	T-1	T
j = 1	...			1:5 $^{0/0}$		1:15 $^{0/0}$		1:10 $^{0/0}$
j = 2	...		2:5 $^{0/0}$		2:15 $^{0/0}$		2:10 $^{0/0}$	
j = 3	...		3:5 $^{0/0}$		3:15 $^{0/0}$		3:10 $^{0/0}$	
j = 4	...	4:5 $^{0/0}$		4:15 $^{2/0}$		4:10 $^{2/0}$		
j = 5	...	5:5 $^{0/0}$		5:15 $^{2/0}$		5:10 $^{2/0}$		
		5':5 $^{0/0}$		5':15 $^{2/0}$		5':10 $^{2/0}$		

Using (j , t) as a short-hand notation for the position „row number j, column number t" in the matrix representation of the data structure, we will briefly explain what can be seen in Table 2. At position (1 , T) for instance we have a „1:10" that represents the external demand for item 1 in period

T. The „2:10" at position (2 , T-1) denotes the internal demand for item 2 in period T-1 that is to be met if 10 units of item 1 are indeed produced in period T. Analogously, we find two entries at position (5 , T-2). One entry corresponds to an internal demand for item 5 caused by item 2, the other entry corresponds to an internal demand for item 5´ caused by item 3.

This data structure does of course not define a (feasible) production plan, at least because of two reasons: Capacity limits are not considered and sequence decisions are not made (there is for example no sequence defined in which the items 1, 4 and 5 are to be produced in period T-2). Nevertheless, the information contained in this data structure will (as we will see later on) be used to construct a plan. Until now, let us be satisfied with the following interpretation: A positive entry „j:n" at position (j , t) says that

there is a demand for n units of item j which must not be met later than period t.

3.3 Outline of the Heuristic

On the basis of this data structure, our heuristic alternatingly repeats two phases in order to find a „good" production plan. The first phase is a construction scheme which turns the information contained in the data structure into a feasible production plan – at least it tries to find a feasible plan. The second phase shuffles the entries in the matrix representation of the data structure by shifting some of the demands to the left or to the right. This latter mechanism actually coins the name „demand shuffle" as we like to call the two-phase procedure as a whole. This scheme is repeated over and over again until a predefined number of iterations is performed. In summary, the procedure works as follows:

1. *Generate the initial data structure*
2. ***Phase 1:** (Try to) construct a feasible production plan*
   by making use of the information contained in the current data structure
3. *Evaluate the plan and memorize it if it improves the best plan so far found*
4. ***Phase 2:** Shift some demands to modify the current data structure*
5. *Go to step 2 until a predefined number of iterations is performed*
6. *Display the best plan*

Both phases are to be explained in more detail, especially the motivation for shifting demands needs to be given. We start with the second phase since this continues our introduction to the data structure. A detailed discussion of phase number one is delayed until later.

3.4 Phase 2: Shifting Demands

Why makes shuffling demands sense? This certainly is the most crucial question needed to be answered before we go into all these details that give phase two a well-defined shape. To facilitate lot sizing is the answer and shall be explained. Consider Table 2 again where you can see that there are some demands for item 2 for instance which are to be met no later than period T-3 and T-1, respectively. It is likely to happen now that item 2 is produced in period T-1 and T-3 to keep holding costs for item 2 low, but, as soon as there are any other items being scheduled in period T-2 (which would save holding costs for these) setup costs for item 2 are to be charged twice. This features the trade-off between low setup costs and low holding costs which is the problem lot sizing and scheduling is all about. What are the alternatives to overcome this problem?

First, one could try to integrate idle periods into a production plan. We don't follow this idea although it might help in some cases. Sure, from a theoretical point of view idle periods may be necessary to find an optimal solution. But, one of the most shortcomings of advanced lot sizing and scheduling algorithms is that they are not commonly accepted by practitioners who do not seldom stay with traditional approaches like the one of Wagner-Whitin, the EOQ-formula or a lot-for-lot production [Fleischmann 1988, Zoller and Robrade 1988]. One of the main reasons for this is that some of the results generated by state-of-the-art procedures are contrary to the practitioner's idea of „good" production plans. Practitioners do not have an objective function in mind that minimizes the sum of setup and holding costs. They think in other terms, too. One of these is a high capacity utilization. Keeping a machine idle for some time is not what managers find well to do. So, what else can we do?

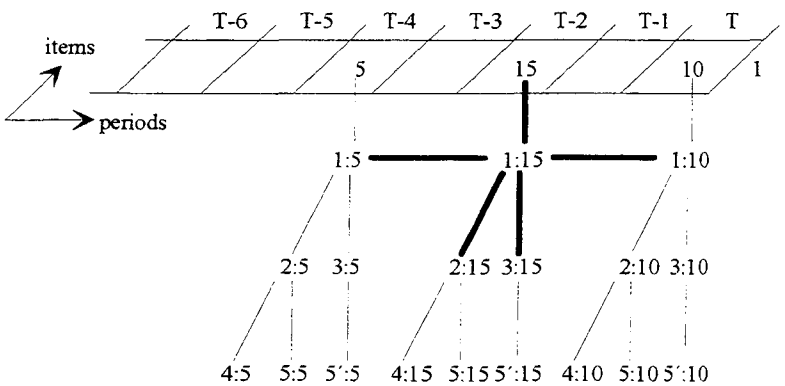
The second alternative we have to avoid setups is building lots. Let us back up to the example above, instead of scheduling item 2 in period T-1 and period T-3 we could schedule a lot built up in period T-3. This is the basic idea we want to support. And it is done by shifting the demand which is to be met in period T-1 to the left (see Table 3). Shifting the entry „2:10" by two periods to the left means that we must fulfill a demand for 25 units of item 2 no later than period T-3. The splitting of these 25 units into two separate data structure entries representing 15 and 10 units, respectively, will be of relevance for being able to shift the 10 units back to the right by subsequent shift operations as we will understand quite soon.

Table 3: Facilitating lot building

	...	T-6	T-5	T-4	T-3	T-2	T-1	T
...								
j = 2	...		2:5		2:15		□	
					2:10 ←			
...								

Up to now, we only sketched out the very basic idea. Many open questions still remain and got to be discussed in the subsequent part. The first question we will answer is, what are the bounds that limit the distance (i.e. number of periods) a data structure entry may be shifted. To understand these limits let us have a look at the relationship between the external demand matrix and the data structure entries by studying our running example again. Figure 2 graphically displays what is done to derive the (initial) data structure from the external demand matrix. According to what was said above, each positive entry in the external demand matrix (see Table 1) induces a treelike substructure (compare Figure 1) of the data structure containing the external demand and all the internal demand that occurs.

Figure 2: Deriving the data structure entries from the external demand matrix

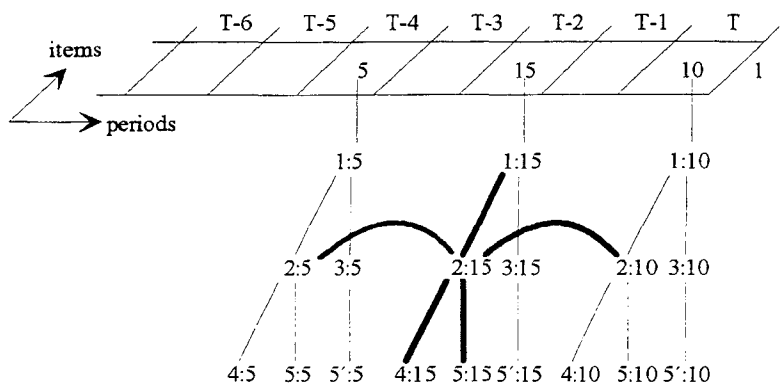


If we look at data structure entries – „1:15" for example – that represent external demand we can figure out some bounds that are depicted by bold lines in Figure 2. The properties which allow us to determine these bounds are the following: First, shortages are not allowed and external demands are to be met promptly. Hence, the entry „1:15" must not be shifted to the right beyond period T-2. Second, we face a multi-level „gozinto"-structure. Left shifts of „1:15" are thus limited by immediate predecessors in the „gozinto"-tree, i.e. by the entries „2:15" and „3:15". Remember, that positive lead times were assumed. Last, if there is some production of an item j in a certain period, say t, to meet some demand for item j in period t_d where $t \leq t_d$ must hold since shortages are not allowed, then there is no production of item j in a period t' earlier than t which meets a demand later than t_d. In other words, a schedule in which „1:15" is scheduled earlier than „1:5" or later than „1:10" need not to be considered.

Data structure entries which do not directly correspond to an external demand – like „2:15" for instance – do have similar bounds (see Figure 3). First, these entries must respect the „gozinto"-structure as well. Left shifts are thus limited by entries that represent immediate predecessors in the „gozinto"-tree (in our case these are „4:15" and „5:15") while right shifts are limited by the entry that represents the unique successor in the „gozinto"-tree („1:15" in the example). Again, one must not forget the positive lead times. Second, there are bounds defined by entries that correspond to the same

position in the „gozinto"-tree but which stem from another external demand. So, „2:15" need not to be left shifted beyond „2:5" or right shifted beyond „2:10". It is important to note, that a precedence relation can only be defined among entries that correspond to same positions (or paths) in the „gozinto"-tree starting from the root that corresponds to the external demand. This is to say, that for instance „5:15" is bounded by „5:5" and „5:10" but not by „5′:5", „5′:15" or „5′:10". For the same reason, an external demand for item 2 no matter in what period would introduce no other bounds than those depicted in Figure 3. Since we need to refer to the entries that define these non-„gozinto"-bounds later again, let us introduce the terms left wing entry and right wing entry here which denote uniquely defined entries: In our example, „1:5" is the left wing entry of „1:15", and „1:10" is the right wing entry of „1:15". Analogously, „2:5" and „5:5" are the left wing entries, „2:10" and „5:10" are the right wing entries of „2:15" and „5:15", respectively.

Figure 3: Bounds for data structure entries that correspond to internal demand



Once more, studying our example gives some additional insight. As we know, Table 2 provides the matrix representation of the initial data structure. This time look at the upper indices attached to each entry in the matrix. Each entry has the form „j:n $^{left/right}$" where left denotes the maximal distance the entry can be shifted to the left and right denotes the maximal distance the entry can be shifted to the right.

Some points of interest are worth to be highlighted. The entry „1:15" for instance cannot be shifted to the right because it is close to its deadline. Neither can this entry be shifted to the left since the entries that represent the immediate „gozinto"-predecessors, namely „2:15" and „3:15", reside in column number T-3 (both item 2 and item 3 have a lead time of one period). Analogously, „2:15" and „3:15" cannot be right shifted. For a similar rationale the entry „4:15" cannot be moved to the right. But, „4:15" can be left shifted since it represents a leaf in the „gozinto"-tree. „4:15" is left-bounded by the entry „4:5" which is two periods away. Why do have the entries „4:5", „5:5" and „5′:5" upper indices 0/0 instead of T-7/0? These entries stem from a leftmost external demand for item 1, i.e. we assumed no

external demand for item 1 earlier than T-4. Since shifting demands was introduced to support a building of lots, all the entries that are derived from a leftmost external demand for an item need not be considered for a left shift. In other words, entries without a left wing entry have initial indices 0/0. It is remarkable to note, that such entries will never be shifted since right shifts are impossible for being close to their deadlines.

Assume that we perform a two-period left shift of the entry „5′:15". Table 4 gives the changes marked as underscored entries.

Table 4: The data structure for the example after 5′:15 has been left shifted

	...	T-6	T-5	T-4	T-3	T-2	T-1	T
j = 1	...			1:5 $^{0/0}$		1:15 $^{0/0}$		1:10 $^{0/0}$
j = 2	...		2:5 $^{0/0}$		2:15 $^{0/0}$		2:10 $^{0/0}$	
j = 2	...		3:5 $^{0/0}$		3:15 $^{2/0}$		3:10 $^{0/0}$	
j = 4	...	4:5 $^{0/0}$		4:15 $^{2/0}$		4:10 $^{2/0}$		
j = 5	...	5:5 $^{0/0}$		5:15 $^{2/0}$		5:10 $^{2/0}$		
		5′:5 $^{0/0}$		□		5′:10 $^{4/0}$		
		5′:15 $^{0/2}$						

Now, we have the following situation: „5′:15" may be shifted back to the right. Alternatively, the entry „3:15" may be shifted to the left. „5′:10" which is left-bounded only by „5′:15" can be moved to the left by at most four periods now. The rest of the data structure is kept unchanged.

A formal definition of how upper indices are updated needs to be given now in order to avoid ambiguousness.

Symbols:

„j:n" – the entry that has been shifted;

$t_{j:n}$ – the period the entry „j:n" is currently (after the shift) positioned in;

$t_{leftwing}$ – the period the right wing entry of „j:n" is currently positioned in;

$t_{rightwing}$ – the period the right wing entry of „j:n" is currently positioned in;

t_{demand} – the period the (internal or external) demand for „j:n" occurs in

pred (j:n) – the set of entries that correspond to „gozinto"-predecessors of „j:n" (which is an empty set if „j:n" is a leaf of the „gozinto"-tree);

Case I: „j:n“ corresponds to an external demand

$$left := \min \{ t_{j:n} - t_{leftwing} ; \min \{ t_{j:n} - t_{k:m} - v_k \mid k:m \in pred(j:n) \} \}$$

and

$$right := \min \{ t_{demand} - t_{j:n} ; t_{rightwing} - t_{j:n} \}$$

are the new upper indices of „j:n“. If there is no right wing entry, the new value of right is more simply be computed as $t_{demand} - t_{j:n}$. Furthermore, some of the indices of entries which define bounds for „j:n“ are to be updated as well. These are:

- The index left of „j:n“'s right wing entry (if such entry exists) according to Case I.
- The index right of the left wing entry of „j:n“ (remember, that a left wing entry must exist, otherwise „j:n“ could not have been shifted) according to Case I.
- The indices right of all entries $k:m \in pred(j:n)$ according to Case II. By definition pred (j:n) only contains entries which correspond to internal demands.

Case II: „j:n“ corresponds to an internal demand

$$left := \min \{ t_{j:n} - t_{leftwing} ; \min \{ t_{j:n} - t_{k:m} - v_k \mid k:m \in pred(j:n) \} \}$$

and

$$right := \min \{ t_{demand} - t_{j:n} - v_j ; t_{rightwing} - t_{j:n} \}$$

are the new upper indices of „j:n“. Note, that t_{demand} is the period the unique „gozinto“-successor is currently positioned in. Again, the new value of right is easier to compute if there is no right wing entry (it is $t_{demand} - t_{j:n} - v_j$ then). Similar to Case I, the indices of entries which define bounds for „j:n“ are also to be updated:

- The index left of „j:n“'s right wing entry (if such entry exists) according to Case II.
- The index right of the left wing entry of „j:n“ (remember, that a left wing entry must exist, otherwise „j:n“ could not have been shifted) according to Case II.
- The indices right of all entries $k:m \in pred(j:n)$ according to Case II.
- The index left of the entry which corresponds to the unique „gozinto“-successor according to Case II.

While we have chosen a two-period left shift of the entry „5′:15" by arbitration in order to explain what is to be done to keep the bound indices up to date when shifting an entry, more problem oriented rules to select and to shift an entry certainly help to improve the quality of the algorithm. In our studies the selection of an entry is governed by the following regime: Attached to each entry „j:n" we maintain a priority value, e.g.

$$\text{priority}_{j:n} := \frac{h_j \bullet n \bullet (1 + t_{j:n} - t_{\text{leftwing}})}{s_j}$$

where entries with no left wing entry are assigned to the value zero instead. Then, we randomly select one entry among all the entries with a uniform distribution. Afterwards, we select another entry in the same way. In the case that the priority value of the second entry is greater than the priority value of the first one we proceed to select entries. This is continued step by step until the corresponding priority values do not increase any further. The entry with the largest priority value (which is the second but last selected) is the one we like to shift.

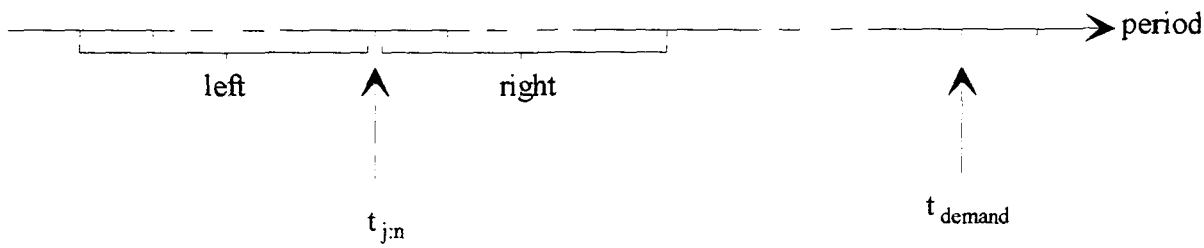
At this point of time, some remarks concerning the way to select an entry are to be made. First, the definition of the priority value is just one example out of a set of several senseful alternatives. In our studies, the definition given above was among the most successful ones. Note, that, if we use this definition, when an entry is shifted we only need to update the priority value of the entry itself and of its left and right wing entries. The values of all the other entries in the data structure are kept unchanged which certainly is a key prerequisite to fast implementations. Second, other procedures of course may run instead to find a „good" entry to shift. Some ideas are listed here: One could (randomly?) select a certain prespecified number of entries (maybe all? – but this is very time consuming) and choose the one with the highest priority value. The priority values could also be used to define a random distribution for choosing an entry. It would be then, so to say, more probable – but not sure – to choose an entry with a high priority value. Again, these ideas were inferior when tested in our study.

Eventually, when an entry „$j:n^{\text{left/right}}$" is chosen to be shifted it is still unclear whether to perform a left shift or a right shift (if both are feasible) and how many periods to shift the entry. For the latter we advice to perform either a „full-size" left shift or a „full-size" right shift, i.e. to move the entry either left periods to the left or right periods to the right. Smaller moves turned out to yield worse results in our tests. Left now, is a discussion of how to decide if a left shift or a right shift operation shall be executed. In fact, this is not as clear as it might seem since it may have happened that an entry was chosen with indices 0/0 – such entries must not immediately be shifted neither to the left nor to the right. Before we will turn back to this case, let us consider the more simple situation first. Assume, that

$$\text{left} + \text{right} > 0,$$

i.e. the entry may be shifted either to the left or to the right or both. Figure 4 gives a graphical representation. The entry „j:n$^{left/right}$ represents an (external or internal) demand that actually occurs in period t_{demand} but which is to be fulfilled no later than period $t_{j:n}$. Note, that $0 \leq right \leq t_{demand} - t_{j:n}$ and $0 \leq left \leq t_{j:n} - 1$ holds. If the entry corresponds to an internal demand, the upper bound of the value right can be lowered to $t_{demand} - t_{j:n} - v_j$.

Figure 4: The situation after an entry „j:n$^{left/right}$" is chosen with left + right > 0



A random choice is made to decide for the direction of the shift operation. To do so, we compute two probabilities $prob_{left}$ and $prob_{right} := 1 - prob_{left}$. Then, we draw a random number RN with a uniform distribution from the interval [0 ; 1]. If $RN \leq prob_{left}$ we perform a „full-size" shift to the left, otherwise a „full-size" right shift is initiated. To make sense, the probabilities ought to be chosen with respect to the following two conditions:

$$left = 0 \Rightarrow prob_{left} = 0$$

and

$$right = 0 \Rightarrow prob_{right} = 0$$

In our studies, several rationales lead to the definition of probabilities $prob_{left}$ and $prob_{right}$. Since storing items in inventory causes holding costs, it should be more probable to perform a right shift than a left shift if the value of right is at least as high as the value of left, i.e.

$$right \geq left \Rightarrow prob_{right} > prob_{left}$$

should hold. Moreover, large values of right indicate high holding costs and therefore should lead to a high probability $prob_{right}$ while on the other hand large values of left express that it is likely to be necessary to facilitate lot building with the left wing entry and thus should result in a high probability

$prob_{left}$. Another aspect is, that important differences between a left and a right shift vanish if $t_{demand} - t_{j:n}$ is large – in such cases it may not matter if we perform a left or a right shift.

On the basis of these considerations, we define the following:

$$\pi_{left} := \sum_{\Delta t = 1}^{left} \frac{1}{t_{demand} - t_{j:n} + \Delta t + 1}$$

and

$$\pi_{right} := \sum_{\Delta t = 1}^{right} \frac{1}{t_{demand} - t_{j:n} - \Delta t + 1}$$

This renders it possible to define

$$prob_{left} := \frac{\pi_{left}}{\pi_{left} + \pi_{right}}$$

and

$$prob_{right} := \frac{\pi_{right}}{\pi_{left} + \pi_{right}}.$$

It is quite easy to verify, that the conditions above are fulfilled.

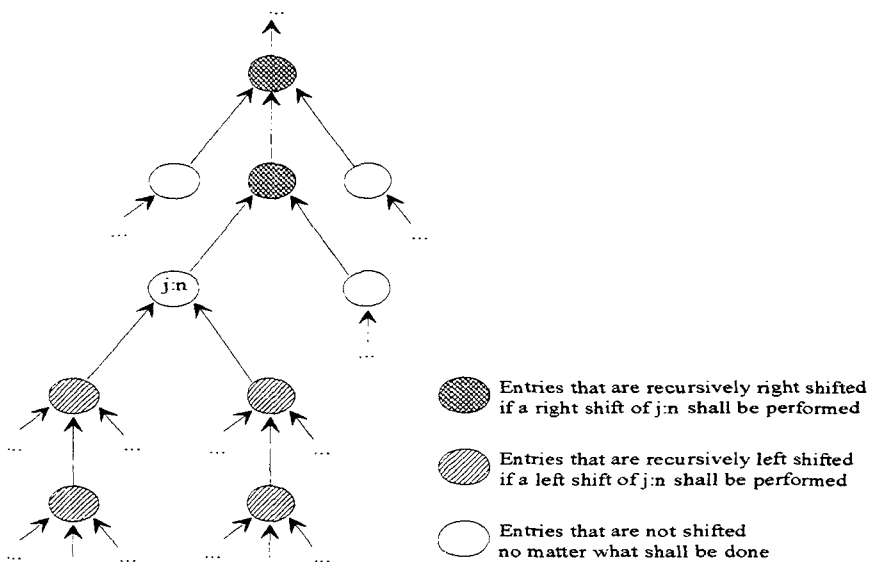
Let us now back up to the situation that once we have chosen an entry „$j{:}n^{left/right}$“,

$$left + right = 0$$

appears to be the case. It is impossible to shift such an entry immediately. But we do not give up soon and do the following: First, we decide if we like to shift the entry to the left or to the right (how this is done will be explained in just a moment). Imagine that we decided for a left shift. We then recursively perform „full-size“ left shifts with all the entries which are predecessors in the same „gozinto“-tree (see Figure 5) starting at the leafs of the tree. Afterwards, the entry has upper indices left′/0 where left′ may be positive now. If this eventually happens to be, we do a „full-size“ left shift. Suppose, that we decided for a right shift. This time we recursively perform „full-size“ right shifts with all the successive entries which belong to the same „gozinto“-tree (see Figure 5 again) beginning at the root of the tree. We then end up with new upper indices 0/right′ where right′ may be positive now. According to what we intended to do, a „full-size“ right shift is eventually executed. In both cases it may happen, that the

upper indices stay 0/0. Then, the entry itself is not shifted at all, though some of the entries in the corresponding „gozinto"-tree were. However, these entries are not shifted back to their old positions.

Figure 5: Entries that are recursively shifted if left + right = 0



Traversing the „gozinto"-tree in order to perform recursive shift operations needs to be specified in more detail. As usual in recurrent computing, a stack is temporarily used for implementation with push and pop as standard operations to add/remove an entry to/from the top of the stack.

Case I: A left shift of „j:n" shall be performed

1. *Pile „j:n" and all its predecessing entries up on a stack*
   (call subroutine pile_up_for_left_shift (j:n))
2. *Perform „full-size" left shifts with all entries on the stack*
   (call subroutine stack_left_shift())

where

```
subroutine pile_up_for_left_shift (entry)
{  push (entry)
   for all k:m ∈ pred (entry)
         pile_up_for_left_shift (k:m)
}
```

and

subroutine stack_left_shift ()
{ **while** „stack is not empty" do
{ entry := pop ()
if „left index of entry is greater zero" **then**
„perform a „full-size" left shift with entry including index updating"
} }

Case II: A right shift of „j:n" shall be performed

1. *Pile „j:n" and all its successing entries up on a stack*
(call subroutine pile_up_for_right_shift (j:n))
2. *Perform „full-size" right shifts with all entries on the stack*
(call subroutine stack_right_shift())

where

subroutine pile_up_for_right_shift (entry)
{ **if** „entry does no correspond to an external demand" **then**
pile_up_for_right_shift („the unique „gozinto"-successor")
push (entry)
}

and **subroutine** stack_right_shift () almost equals **subroutine** stack_left_shift (), but, this time we test if the right index of entry is greater than zero and perform a „full-size" right shift if it is.

Unclear up to now is, how to decide for a direction of the recursive shift operation. Similar to what was done if left + right > 0 holds, we suggest a random choice with probabilities $prob_{left}$ and $prob_{right}$. But this time, another definition is used:

$$prob_{left} := \frac{1}{1 + \frac{h_j \bullet n \bullet (t_{init} - t_{j:n})}{s_j}}$$

and

$$prob_{right} := 1 - prob_{left}$$

Here, t_{init} denotes the period the entry „j:n" is initially positioned in (compare Table 2). The formulas above are chosen such that if holding costs up to t_{init} equal setup costs, we face a fifty-fifty chance for a left or a right shift, respectively. A shift to the right becomes more probable if the distance to the entry's initial period of time increases.

The process of shifting demand entries is now totally defined and we may turn to the phase one, namely the construction of production plans on the basis of the information contained in the current data structure.

3.5 Phase 1: Construction of a Production Plan

The construction scheme is a backward oriented one since this is appropriate for multi-level „gozinto"-structures [Kimms 1993a]. Best to do is to explain the details by means of studying our example again. Let us assume, that the data structure looks like Table 5 which, by the way, can be achieved by performing six left shift operations using the initial data structure (Table 2) as a starting point. From now on, the upper indices left/right are out of interest in all but one cases and are thus left away unless necessary to ease the representation. On the left hand side, we have the usual matrix representation of the data structure where a $\Downarrow$-symbol above the upper row indicates the focus of attention. On the right hand side, we find a table that contains two bits of information. In the demand column, the cumulative demand in periods between the current period of attention and period T that has not been met yet is found. A „10" in row j = 1 thus indicates, that 10 units of item 1 – the external demand for item 1 in period T – are to be produced. In the available column, the cumulative number of items which is allowed to be produced at maximum is contained. This information is derived from the data structure. A „10" in row j = 1 stems from the entry „1:10" which tells us, that 10 units of item 1 are to be produced no later than period T. More formally, if period T is the focus of attention, then the values found in row j, denoted as $demand_j$ and $available_j$, respectively, are defined as

$$demand_j := d_{jT}$$

and

$$available_j := data\_structure_{jT}$$

where $data\_structure_{jt}$ stands for the sum of available units in row j and column t of the data structure's matrix representation. The other symbols equal those defined in chapter 2.

Table 5: The data structure on the basis of which a production plan shall be constructed

							⇓			
	...	T-6	T-5	T-4	T-3	T-2	T-1	T	demand	available
j = 1	...			1:5		1:15		1:10	10	10
j = 2	...		2:5		2:15		2:10			
j = 3	...		3:5				3:10			
			3:15							
j = 4	...	4:5		4:10						
		4:15								
j = 5	...	5:5		5':10		5:10				
		5':5								
		5:15								
		5':15								

In the sequel, we will assume that 18 capacity units are available in each period and that producing one item – no matter which one – needs exactly one of these units.

Coming back to Table 5, we are about to decide which – if any – items to produce in period T. The table on the right shows that there is some demand only for item 1. Hence, let us schedule item 1 in period T. A setup for this item can also be done in period T. Table 6 provides the situation afterwards. The new focus of attention is period T-1 now. 10 units of item 1 are scheduled in period T which is indicated by the entry „10" at position (1 , T). Note, the contents of the table on the right has changed. The entries in row j = 1 are deleted since the demand for item 1 has been met. But, two new entries appear in rows j = 2 and j = 3. In the demand column there is a „10" in each of these rows which stand for the internal demands for item 2 and item 3 that is to be met until period T-1. The entries in the available column correspond to the entries „2:10" and „3:10" in the data structure. Formally, the entries $demand_j$ and $available_j$ are computed as follows:

Case I: The current focus of attention is period t = T. (The starting situation – see the definition above.)

Case II: The current focus of attention is period t where $T - v_j < t < T$. (Internal demand cannot exist.)

$$demand_j := demand_j - q_{j(t+1)} + d_{jt}$$

and

$$available_j := available_j - q_{j(t+1)} + data\_structure_{jt}$$

Case III: The current focus of attention is period t where $t \leq T - v_j$.

$$demand_j := demand_j - q_{j(t+1)} + d_{jt} + \sum_{i \in S(j)} (a_{ji}\, q_{i(t+v_j)})$$

and

$$available_j := available_j - q_{j(t+1)} + data\_structure_{jt}$$

Table 6: The situation after the first backward step

⇓

	...	T-6	T-5	T-4	T-3	T-2	T-1	T	demand	available
j = 1	...			1:5		1:15		10		
j = 2	...		2:5		2:15		2:10		10	10
j = 3	...		3:5				3:10		10	10
			3:15							
j = 4	...	4:5		4:10						
		4:15								
j = 5	...	5:5		5':10		5:10				
		5':5								
		5:15								
		5':15								

Now, there is a choice. Since we have scarce capacities (18 units), we cannot fulfill both demands in period T-1. This would need 20 capacity units. Assume that we decide for item 2 first and use the remaining capacity to schedule item 3, too. How such decisions are made is not important at this point and will be discussed later. Note, that at most one setup is allowed within each period. Hence, setting the machine up for item 2 can be done in T-1, but, the setup for item 3 must happen in T-2 (or earlier). We have to remind this since it will affect our decisions in period T-2. Table 7 shows what has happened then.

Table 7: The situation after the second backward step

⇓

	...	T-6	T-5	T-4	T-3	T-2	T-1	T	demand	available
j = 1	...			1:5		1:15		10	15	15
j = 2	...		2:5		2:15		10			
j = 3	...		3:5				8		2	2
			3:15							
j = 4	...	4:5		4:10					10	
		4:15								
j = 5	...	5:5		5':10 2/2		5:10			18	10
		5':5								
		5:15								
		5':15								

Period T-2 is the focus of our attention now. It is very instructive to study the table on the right. An interesting aspect can be seen in row j = 4. An internal demand for 10 units of item 4 which is caused by the 10 units of item 2 scheduled in period T-1 is to be met, but, no units of item 4 are available. Here, the shifting of demands comes in useful for the first time. The data structure entry „4:10" that corresponds to this internal demand was left shifted to period T-4. Concluding, only items 1, 3 or 5 may be scheduled in period T-2. Item 2 may not since there is no demand for it and item 4 may not because there is demand but no items are available. Remember now, that we scheduled item 3 in period T-1 but a setup for this item had not been done in T-1. So, in period T-2 item 1 and item 5 may not be scheduled both, otherwise more than one setup would occur in period T-2 which violates our basic PLSP assumption. Assume, that we choose to schedule item 3 and then use the remaining capacity to produce item 5. Keep in mind that the setup for item 5 must occur in period T-3 (or earlier).

Now, another aspect which is very important to the quality of our algorithm (as computational tests have revealed) appears. There is more demand for item 5 (18 units) then there is available (10 units). But, once that we have chosen to schedule item 5, it would be unwise not to try to fulfill the whole demand (or at least as much as remaining capacity allows to) because a setup occurs now anyhow and holding costs could be saved this way. In general, we suggest the following: If an item is scheduled with less units available – at least one unit must be available – than there is demand for, then look to the left and shift all those entries back to the right that correspond to the same item number and that have an upper index right which allows to reach the current period of attention. Stop, if the demand can be satisfied. If we are in a period in which two items are scheduled, we do the right shift operations after having scheduled the available amounts of both items and use the remaining capacity to „fill the schedule up". If no such capacity remains, the right shifts are not performed. Note, that shifting an entry

to the right affects the upper indices of other entries. So, more entries may be shifted to the right than it first looks like. In our example the entry „5′:10" currently positioned in period T-4 may be and in fact is right shifted to period T-2. No other entry with item number 5 can be right shifted to T-2 and even if it could it would not since the demand for 18 units can now be met. So, we are done having 20 units of item 5 available. Table 8 provides the result. A piece of code refines the explanations. Assume that period t is the current focus of attention and that q_{jt} units of item j are already scheduled in t (a second item may be scheduled as well). Furthermore, suppose that $demand_j > available_j$ holds and that some capacity units are not used up (i.e. $q_{jt} = available_j$). Then, execute the piece of program given below. For the sake of simplicity let us assume that a second item which may be scheduled in t, too, does not have more demand than available. If this does not hold, the if-clause in the code must test for this second item as well. In the case that more entries than just one fulfill the if-clause, choose any of them.

```
t′ := t-1
while „there is remaining capacity and demand_j > available_j holds" do
{       if „there is an entry j:n^left/right at position ( j , t′ ) in the data structure with right ≥ t - t′" then
        {       „perform a right shift to period t with that entry including the update of upper indices"
                available_j := available_j + n
                „increase q_jt with respect to capacity limits and demand_j"
        }
        else    t′ := t′ - 1
}
```

Table 8: The situation after the third backward step

⇓

	...	T-6	T-5	T-4	T-3	T-2	T-1	T	demand	available
j = 1	...			1:5				10	15	15
j = 2	...		2:5		2:15		10			15
j = 3	...		3:5			2	8			
			3:15							
j = 4	...	4:5		4:10					10	
		4:15								
j = 5	...	5:5		—		16			4	4
		5′:5								
		5:15								
		5′:15								

Now, back to the example, period T-3 is the focus of attention. Row j = 2 is of special interest because for the first time up to now, there are items available (15 units of item 2 that stem from the entry „2:15") for which no demand has yet occurred. This is the case since the entry „2:15" represents an internal demand, but, as long as we do not schedule an item 1 again (or as long as no external demand for item 2 occurs), no demand uses these available units up. In summary, items 1 and 5 are the only ones that may be scheduled in period T-3 since only these two have demand and available units. Suppose, that item 5 is chosen first and the remaining capacity is used to produce item 1. The setup for item 1 must occur in period T-4 (or earlier) which has to be kept in mind. Table 9 shows the current situation.

Table 9: The situation after the fourth backward step

⇓

	...	T-6	T-5	T-4	T-3	T-2	T-1	T	demand	available
j = 1	...			1:5	14			10	6	6
j = 2	...		2:5				10		14	15
j = 3	...		3:5			2	8		14	
			3:15							
j = 4	...	4:5		4:10					10	10
		4:15								
j = 5	...	5:5			4	16				
		5':5								
		5:15								
		5':15								

At this point we stop our explanations of the construction scheme since nothing new happens from now on. A feasible plan is found after the T-th backward step, if the demand and the available column contain nothing but zeros.

Anything but the way to decide which items to schedule is well-defined up to here. Again, several variants are possible to choose an item among those with positive demand and positive availability. In our studies, the following turned out to be best: Let $available_j$ denote the entry in the available column and $demand_j$ denote the entry in the demand column of item j. The worst that could happen to item j is that the available items are not scheduled, thus causing holding costs. On the basis of this insight, assign a value $priority_j$ to each item defined as follows:

$$\text{priority}_j := \begin{cases} 0 & \text{, if available}_j = 0 \text{ or demand}_j = 0 \\ h_j \bullet \text{available}_j & \text{, if available}_j > 0 \text{ and demand}_j > 0 \end{cases}$$

Then, make a random choice where the higher the priority value the more probable the corresponding item is chosen which tends to keep holding costs low. More formally, the probability to choose item j is

$$\text{prob}_j := \frac{\text{priority}_j}{\sum_{i=1}^{J} \text{priority}_i}.$$

Note, that if for all items j $\text{priority}_j = 0$ holds, then there is no need to compute these probabilities since no item can be scheduled. The formula giving prob_j is thus well-defined. In periods in which two items can be scheduled, we evaluate these probabilities again after the first item is chosen and before the second is, because it should not happen that the first item is randomly selected again in the second step. Computing all the probabilities again is necessary for the second choice since the probability for the first chosen item is set to zero which affects the probabilities for the other items.

4 Computational Study

To test the demand shuffle heuristic we implemented it in C and run it on a 486 PC with 25 MHz. The test-bed that was used contains 144 problem instances consisting of 5 items and 10 periods of time. These instances were first defined in [Kimms 1993a] and serve as a standard test-bed for all of our studies today. The problem size is small enough to be solved optimally with a standard MIP-solver within reasonable time. It is large enough to provide non-trivial instances.

We start by giving a specification of the test-bed. Then, we provide the results that were recently reached by a so-called randomized regret based procedure to have a point of reference other than the optimal objective function values. Finally, the results of the presented heuristic are revealed.

The basis of our test-bed is formed by four different types of „gozinto"-structures (see Figure 6). In some instances external demand occurs for all end items and no other but those. And, in some instances external demand occurs for all items. The external demand matrix is filled according to the following patterns:

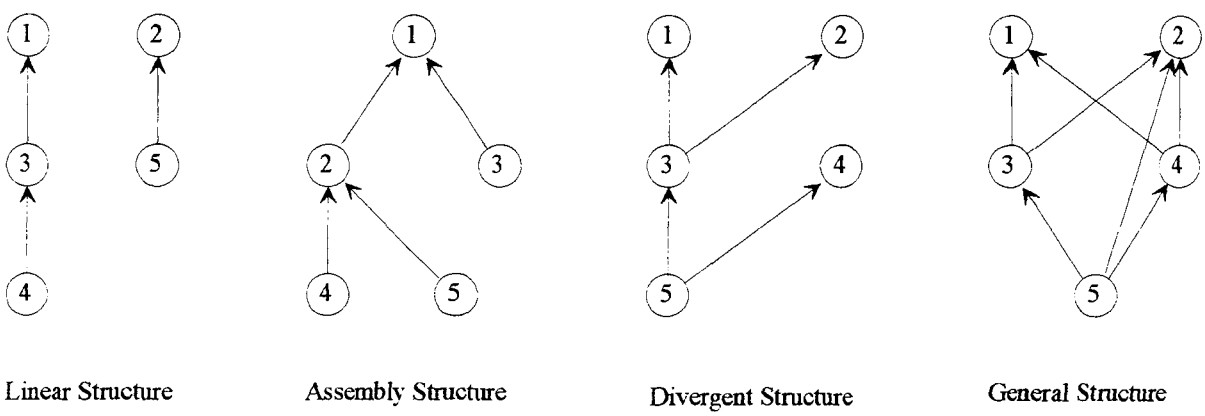
(1) External demand in period 10.

(2) External demand in periods 6 and 10.

(3) External demand in periods 6, 8 and 10.

In the sequel, a triple of the form $\sigma/\pi/\nu$ abbreviates the structure of an instance where $\sigma \in \{L, A, D, G\}$ (which denotes a Linear, Assembly, Divergent or General „gozinto"-structure), $\pi \in \{E, A\}$ (which denotes external demand for End items only or external demand for All items) and $\nu \in \{1, 2, 3\}$ (which stands for one of the demand patterns above). For example, D/A/3 denotes an instance with the divergent „gozinto"-structure, external demand for all items and external demand pattern number (3).

Figure 6: Product structures in the test-bed



Each of the instance structures $\sigma/\pi/\nu$ (24 in total) is combined with six different sets of data to complete the definition of an instance. This leads to the aforementioned 144 instances. The data sets are given here:

(a) Holding and setup costs are defined as in Table 10. Lead times and production coefficients are equal to one in all cases, i.e. $v_j = 1$ and $a_{ji} = 1$. The production of one item (of any kind) consumes one capacity unit, i.e. $p_j = 1$ for all items, while the capacity of the machine is assumed to be constant over time. More precisely, we assume the capacity constraints per time period to be defined as in Table 11. External demands per period (with respect to the demand pattern ν) are assumed to be 20 per item in the case that $\pi = E$ and 10 per item in the case that $\pi = A$.

(b) The same data set as (a) except setup costs being multiplied by 20.

(c) The same data set as (a) except external demand sizes being half of what is defined in (a), i.e. 10 items if $\pi = E$ and 5 items if $\pi = A$.

(d) The same data set as (b) except external demand sizes being chosen as in (c).

(e) The same data set as (c) except $p_j = 0.5$ for all items and all production coefficients are doubled, i.e. $a_{ji} = 2$ for all items j and i with respect to the product structures defined above.

(f) The same data set as (d) except p_j and a_{ji} being chosen alike (e).

Table 10: Setup and holding costs

item	setup costs	holding costs
1	30	5
2	20	4
3	20	3
4	10	2
5	10	1

Table 11: *Capacity constraints per period of time*

	$\pi = E$			$\pi = A$		
	$\nu = 1$	$\nu = 2$	$\nu = 3$	$\nu = 1$	$\nu = 2$	$\nu = 3$
$\sigma = L$	35	35	100	35	35	100
$\sigma = A$	35	35	100	35	35	100
$\sigma = D$	35	100	100	35	100	100
$\sigma = G$	35	100	200	100	100	200

In [Kimms 1994a] we presented a so-called randomized regret based heuristic (as well as a tabu search heuristic) and applied it to the same test instances. Roughly speaking, this heuristic can be seen as a very special case of the demand shuffle heuristic in that it does not perform any shifting operations. In other words, leave the phase 2 of the presented method out and you are close to what was done. An important difference lies in the way an item is chosen to be scheduled. Though a random choice is performed, too, the priority rule on the basis of which a decision is made is much more complex and thus more time-consuming to be computed. Besides holding costs the priority rule is defined by setup costs, the „depth" of an item in the multi-level „gozinto"-structure and the capacity needs to produce an item and all its predecessors. A formal definition is out of the scope of this paper – we refer to [Kimms 1994a]. Nevertheless, we present the average computational results again since they belong to the best ones so far known and are the ones we have to compete with. The results are based on 1000 repetitions, i.e. 1000 production plans were (tried to be) constructed from which the best one is chosen. The randomized regret based heuristic took between 4 to 9 seconds per instance.

The demand shuffle heuristic as it was presented above took 3 to 9 seconds per instance to perform 1000 iterations (that is 1000 repetitions of phase 1 and 2) each. In this study 10 data structure

entries were randomly selected to be shifted during phase 2. Shifting less than 10 entries turned out to be worse, while shifting more than 10 entries did not improve the results. We suggest to determine the number of entries shifted during phase 2 in dependence from the problem size J • T, e.g.

$$\left\lceil \frac{J \bullet T}{5} \right\rceil$$

shift operations.

Tables 12 to 14 provide the results achieved by the demand shuffle heuristic. The deviation of the heuristic results from the optimal results measured as

$$\text{deviation} := 100 \bullet \frac{F^*_H - F^*_{opt}}{F^*_{opt}}$$

where F^*_H is the objective function value computed with the heuristic and F^*_{opt} is the optimal objective function value, is given in detail for each of the 144 instances. Comparing the average results with those that correspond to the randomized regret based procedure ([RR] row / [RR] column) shows the superiority of the demand shuffle method. In the case $\nu = 1$ both heuristics are competitive. But, for $\nu = 2$ where external demands for items occur with a large number of „no-demand" periods in-between, the demand shuffle heuristic drastically pays off (a 9.17% average gap to the optimal solution instead of a 22.05% gap). The demand shuffle heuristic outperforms the randomized-regret based procedure especially in those cases in which setup costs are large in comparison with holding costs, i.e. in problem classes (b), (d) and (f). Here, the results are much more convincing (the gaps are 7.05% instead of 27.24%, 3.15% instead of 37.07% and 2.60% instead of 32.65%). If $\nu = 3$ which is the case in which the external demand matrix is „full of entries", the demand shuffle method again gives drastic improvements (the average gap to the optimal solution is 10.46% instead of 18.76%). Once more, the most amazing results are achieved when problems with large setup costs and low holding costs are solved.

Table 12: Deviation of the demand shuffle heuristic, $\nu = 1$

	(a)	(b)	(c)	(d)	(e)	(f)	average	[RR]
L/E/1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
L/A/1	17.24	2.50	13.16	1.32	5.88	0.76	6.81	4.53
A/E/1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
A/A/1	8.57	1.46	6.82	0.00	4.48	0.73	3.68	5.12
D/E/1	9.09	1.86	8.33	1.03	5.13	0.92	4.39	4.54
D/A/1	19.05	2.93	14.63	1.57	10.91	0.99	8.35	8.26
G/E/1	1.41	2.27	0.00	0.00	2.19	2.43	1.38	3.64
G/A/1	13.33	2.78	11.11	1.52	12.79	2.80	7.39	6.91
average	8.59	1.73	6.76	0.68	5.17	1.08	4.00	
[RR]	8.64	4.90	5.64	0.61	4.01	0.96		4.13

Table 13: Deviation of the demand shuffle heuristic, $\nu = 2$

	(a)	(b)	(c)	(d)	(e)	(f)	average	[RR]
L/E/2	28.21	0.00	0.00	0.00	5.00	0.00	5.53	24.82
L/A/2	19.05	12.23	19.23	6.70	14.56	3.33	12.52	20.70
A/E/2	2.10	9.17	0.00	1.13	8.43	0.00	3.47	26.69
A/A/2	31.25	14.92	22.34	4.67	19.15	2.90	15.87	22.07
D/E/2	14.61	3.82	7.55	2.44	8.54	1.37	6.39	20.89
D/A/2	31.34	6.83	23.81	4.15	16.67	2.17	14.16	24.12
G/E/2	0.00	0.00	0.00	0.00	4.58	3.57	1.36	14.53
G/A/2	18.75	9.43	18.92	6.10	23.53	7.47	14.03	22.57
average	18.16	7.05	11.48	3.15	12.56	2.60	9.17	
[RR]	10.74	27.24	10.88	37.07	13.72	32.65		22.05

Table 14: Deviation of the demand shuffle heuristic, $\nu = 3$

	(a)	(b)	(c)	(d)	(e)	(f)	average	[RR]
L/E/3	6.19	0.00	7.14	0.00	5.62	0.36	3.22	20.99
L/A/3	28.89	12.82	20.95	8.13	13.48	6.49	15.13	15.79
A/E/3	1.83	3.31	1.59	0.83	0.78	0.00	1.39	29.55
A/A/3	31.36	8.38	21.48	12.17	18.93	7.20	16.59	14.54
D/E/3	15.89	7.18	15.48	3.23	15.20	2.82	9.97	17.99
D/A/3	22.33	15.06	15.00	14.20	25.30	8.08	16.66	17.92
G/E/3	3.17	0.00	0.00	0.00	3.43	4.46	1.84	15.76
G/A/3	20.41	14.40	19.63	17.38	26.48	14.83	18.86	17.55
average	16.26	7.64	12.66	6.99	13.65	5.53	10.46	
[RR]	15.55	20.13	13.81	24.85	15.76	22.46		18.76

5 Conclusion

This paper dealt with dynamic capacitated multi-level lot sizing and scheduling in a single-machine environment. It introduced the multi-level single-machine PLSP mixed-integer model. A novel method called demand shuffle was presented. This method successfully combines fast random sampling with problem specific data structure manipulations. In a computational study, it amazed with competitive execution time and very good improvements of former results. A new benchmark for future work has been set.

Acknowledgement

Andreas Drexl is an outstanding advisor and an excellent proofreader.

References

Afentakis, P., Gavish, B., (1986), Optimal Lot-Sizing Algorithms for Complex Product Structures, Operations Research, Vol. 34, pp. 237-249

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

Bahl, H. C., Ritzman, L. P., Gupta, J. N. D., (1987), Determining Lot Sizes and Resource Requirements: A Review, Operations Research, Vol. 35, pp. 329-345

Brüggemann, W., Jahnke, H., (1994), DLSP for 2-Stage Multi-Item Batch Production, International Journal of Production Research, Vol. 32, pp. 755-768

Domschke, W., Scholl, A., Voß, S., (1993), Produktionsplanung – Ablauforganisatorische Aspekte, Heidelberg, Springer

Drexl, A., Haase, K., (1992), A New Type of Model for Multi-Item Capacitated Dynamic Lotsizing and Scheduling, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 286

Drexl, A., Haase, K., Kimms, A., (1993), Losgrößen- und Ablaufplanung in PPS-Systemen auf der Basis randomisierter Opportunitätskosten, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 333, Zeitschrift für Betriebswirtschaft, to appear

Elmaghraby, S. E., (1978), The Economic Lot Scheduling Problem (ELSP): Review and Extensions, Management Science, Vol. 24, pp. 587-598

El-Najdawi, M. K., (1994), A Job-Splitting Heuristic for Lot-Size Scheduling in Multi-Stage, Multi-Product Production Processes, European Journal of Operational Research, Vol. 75, pp. 365-377

El-Najdawi, M. K., Kleindorfer, P. R., (1993), Common Cycle Lot-Size Scheduling for Multi-Product, Multi-Stage Production, Management Science, Vol. 39, pp. 872-885

Fleischmann, B., (1988), Operations-Research-Modelle und -Verfahren in der Produktionsplanung, Zeitschrift für Betriebswirtschaft, Vol. 58, pp. 347-372

Fleischmann, B., (1990), The Discrete Lot Sizing and Scheduling Problem, European Journal of Operational Research, Vol. 44, pp. 337-348

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

Helber, S., (1994a), Kapazitätsorientierte Losgrößenplanung in PPS-Systemen, Stuttgart, M&P

Helber, S., (1994b), Lot Sizing in Capacitated Production Planning and Control Systems, Working Paper, University of Munich, OR Spektrum, to appear

Iyogun, P., Atkins, D., (1993), A Lower Bound and an Efficient Heuristic for Multistage Multiproduct Distribution Systems, Management Science, Vol. 39, pp. 204-217

Karmarkar, U. S., Schrage, L., (1985), The Deterministic Dynamic Product Cycling Problem, Operations Research, Vol. 33, pp. 326-345

Kimms, A., (1993a), Multi-Level, Single-Machine Lot Sizing and Scheduling (with Initial Inventory), Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 329, European Journal of Operational Research, to appear

Kimms, A., (1993b), A Cellular Automaton Based Heuristic for Multi-Level Lot Sizing and Scheduling, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 331

Kimms, A., (1994a), Complementary, Competitive Methods for Multi-Level Lot Sizing and Scheduling: Tabu Search and Randomized Regrets, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 348

Kimms, A., (1994b), Optimal Multi-Level Lot Sizing and Scheduling with Dedicated Machines, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 351

Kuik, R., Salomon, M., (1990), Multi-Level Lot-Sizing Problem: Evaluation of a Simulated-Annealing Heuristic, European Journal of Operational Research, Vol. 45, pp. 25-37

Kuik, R., Salomon, M., van Wassenhove, L. N., Maes, J., (1993), Linear Programming, Simulated Annealing and Tabu Search Heuristics for Lotsizing in Bottleneck Assembly Systems, IIE Transactions, Vol. 25, No. 1, pp. 62-72

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

Roll, Y., Karni, R., (1991), Multi-Item, Multi-Level Lot Sizing with an Aggregate Capacity Constraint, European Journal of Operational Research, Vol. 51, pp. 73-87

Rosling, K., (1986), Optimal Lot-Sizing for Dynamic Assembly Systems, in: Axsäter, S., Schneeweiß, C., Silver, E. A., (eds.), Multi-Stage Production Planning and Inventory Control, Berlin, Springer, pp. 119-131

Roundy, R. O., (1993), Efficient, Effective Lot Sizing for Multistage Production Systems, Operations Research, Vol. 41, pp. 371-385

Salomon, M., (1991), Deterministic Lot Sizing Models for Production Planning, Lecture Notes in Economics and Mathematical Systems, Vol. 355, Berlin, Springer

Salomon, M., Kuik, R., van Wassenhove, L.N., (1993), Statistical Search Methods for Lotsizing Problems, Annals of Operations Research, Vol. 41, pp. 453-468

Stadtler, H., (1994), Mixed Integer Programming Model Formulations for Dynamic Multi-Item Multi-Level Capacitated Lotsizing, Working Paper, Technical University of Darmstadt

Tempelmeier, H., (1992), Material-Logistik – Grundlagen der Bedarfs- und Losgrößenplanung in PPS-Systemen, Berlin, Springer, 2nd edition

Tempelmeier, H., Derstroff, M., (1993a), Mehrstufige Mehrprodukt-Losgrößenplanung bei beschränkten Ressourcen und genereller Erzeugnisstruktur, OR Spektrum, Vol. 15, pp. 63-73

Tempelmeier, H., Derstroff, M., (1993b), A Lagrangean-Based Heuristic for Dynamic Multi-Level Multi-Item Constrained Lot Sizing, Working Paper, University of Cologne

Tempelmeier, H., Helber, S., (1994), A Heuristic for Dynamic Multi-Item Multi-Level Capacitated Lotsizing for General Product Structures, European Journal of Operational Research, Vol. 75, pp. 296-311

Zoller, K., Robrade, A., (1988), Efficient Heuristics for Dynamic Lot Sizing, International Journal of Production Research, Vol. 26, pp. 249-265