Kimms, Alf

Improved lower bounds for the proportional lot sizing and scheduling problem

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 414

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

Suggested Citation: Kimms, Alf (1996) : Improved lower bounds for the proportional lot sizing and scheduling problem, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 414, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/149045

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 414

Improved Lower Bounds for the Proportional Lot Sizing and Scheduling Problem

A. Kimms

No. 414

Improved Lower Bounds for the Proportional Lot Sizing and Scheduling Problem

A. Kimms

October 1996

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

Where standard MIP–solvers fail to compute optimum objective function values for certain MIP–model formulations, lower bounds may be used as a point of reference for evaluating heuristics. In this paper, we compute lower bounds for the multi–level proportional lot sizing and scheduling problem with multiple machines (PLSP–MM). Four approaches are compared: Solving LP–relaxations of two different model formulations, solving a relaxed MIP–model formulation optimally, and solving a Lagrangean relaxation.

Keywords: Multi–level lot sizing, scheduling, lower bounds, PLSP

1 Introduction

The problem we are focussing at, 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 resources. Some (maybe all) of them are scarce. 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. Most authors assume that the holding costs for an item must be greater than or equal to the sum of the holding costs for all immediate predecessors. They argue that holding costs are mainly opportunity costs for capital which occurs no matter a component part is assembled or not. Two reasons persuade us to make no particular assumptions for holding costs. First, as it is usual in the chemical industry for instance, keeping some component parts in storage may require ongoing additional effort such as cooling, heating, or shaking. While these parts need no special treatment when processed, storing component parts might be more expensive than storing assembled items. Second, operations such as cutting tin mats for instance make parts smaller and often easier to handle. The remaining "waste" can often be sold as raw material for other manufacturing processes. Hence, opportunity costs may decrease when component parts are assembled. However, it should be made clear that the assumption of general holding costs is the most unrestrictive one. All models and methods developed under this assumption work for more restrictive cases as well.

Each item requires one resource 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 resource up for producing a particular item incurs item–specific setup costs which are assumed to be sequence independent. Setup times are not considered. 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. To get things straight, note that some authors use the word changeover instead of setup in this context.

The most fundamental assumption here is that for each resource at most one setup may occur within one period. Hence, at most two items sharing a common resource for which a setup state exists may be produced per period. Due to this assumption, the problem is known as the proportional lot sizing and scheduling problem (PLSP) [4, 12, 23]. 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) [3, 8, 17, 25, 31] as well as the continuous setup lot sizing problem (CSLP) [1, 18, 19]. 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) [2, 5, 10, 16, 24, 28, 29] represents a large bucket model since many items can be produced per period. Remember, the CLSP does not include sequence decisions and is thus a much "easier" problem. An extension of the single–level CLSP with partial sequence decisions can be found in [11]. In [14] a large bucket single–level lot sizing and scheduling model is discussed.

A comprehensive review of the multi–level lot sizing literature is given in [23] where it is shown that most authors do not take capacity restrictions into account and that they make restrictive assumptions such as linear or assembly gozinto–structures. If scarce capacities are considered, the work is mostly confined to single–machine cases. The most general methods are described in [35, 36] where the multi–level CLSP is attacked.

The text is organized as follows: Section 2 gives a MIP–model formulation of the PLSP–MM. Solving the LP–relaxation of a PLSP–MM–model optimally is a straightforward idea. Hence, Section 3 deals with this approach and discusses a network reformulation of the model. Another way to get lower bounds is to ignore some of the constraints and to solve the remaining problem optimally. This path is followed in Section 4 where a B&B–procedure is used to attack the uncapacitated, multi–level, multi–machine lot sizing and scheduling problem. Note, solving an uncapacitated lot sizing and scheduling problem is far more than just finding a value less than or equal to the optimum objective function value of a PLSP–MM–instance, or lower bound for short. It also provides a solution for the uncapacitated problem which may appear in distribution networks and supply chains for instance [26, 27, 32]. On the basis of this, Section 5 introduces a method to solve a Lagrangean relaxation of the capacity constraints. Finally, Section 6 summarizes the lower bounds obtained.

2 Multi–Level PLSP with Multiple Machines

An important variant of the PLSP is the one with multiple machines (PLSP–MM). Several resources (machines) are available and each item is produced on an item–specific machine. This is to say that there is an unambiguous mapping from items to machines. Of course, some items may share a common machine. Special cases are the single–machine problem for which models and methods are given in [21, 22], and the problem with dedicated machines where items do not share a common machine. For the latter optimal solutions can be easily computed with a lot–for–lot policy [20]. Heuristics for the PLSP–MM are described in [23].

Let us first introduce some notation. In Table 1 the decision variables are defined. Likewise, the parameters are explained in Table 2. From these parameters, we can easily derive $\bar{\mathcal{S}}_j$ the set of all successors and $\bar{\mathcal{P}}_j$ the set of all predecessors, respectively, of item j which will be needed in later sections. 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–MM

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

subject to

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

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

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

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

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 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–MM

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

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

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

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

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 Network Representations

Some researchers have introduced network representations for lot sizing problems to derive "tighter" reformulations, i.e. new models where the LP–relaxation of an instance has an optimum objective function value that is greater than the optimum objective function value of the LP–relaxations of the straightforward model formulation [5, 30, 33, 34].

3.1 A Simple Plant Location Representation

For the multi–level CLSP, a computational study in [33] reveals that a simple plant location representation adapted from [30] gives the same lower bounds as a shortest route representation adapted from [5].

In here, we thus confine our focus of attention to a simple plant location representation of the PLSP–MM. Table 3 gives a new decision variable where the computation of the gross demand is defined as:

$$d_{jt}^{L4L} \stackrel{def}{=} \begin{cases} d_{jt} + \sum_{i \in S_j} a_{ji} d_{i(t+v_j)}^{L4L} \\ \qquad \text{, if } 1 \leq t \leq T - v_j \\ d_{jt} \\ \qquad \text{, if } T - v_j < t \leq T \end{cases} \qquad \begin{array}{l} j = 1, \ldots, J \\ t = 1, \ldots, T \end{array} \qquad (10)$$

All other notation is as defined in Section 2.

Symbol	Definition
$z_{jt\tau}$	Fraction of the gross demand $d_{j\tau}^{L4L}$ for item j produced in period t where $z_{jt\tau} \in [0, 1]$.

Table 3: A New Decision Variable for the PLSP–MM Network Representation

$$\min \sum_{j=1}^{J} \sum_{t=1}^{T} (s_j x_{jt} + h_j I_{jt}) \tag{11}$$

subject to

$$I_{jt} = I_{j(t-1)} + \sum_{\tau=t}^{T} z_{jt\tau} d_{j\tau}^{L4L} - d_{jt} \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \end{matrix} \tag{12}$$

$$- \sum_{i \in \mathcal{S}_j} a_{ji} \sum_{\tau=t}^{T} z_{it\tau} d_{i\tau}^{L4L}$$

$$I_{jt} \geq \sum_{i \in \mathcal{S}_j} \sum_{\tau=t+1}^{\min\{t+v_j, T\}} a_{ji} \sum_{\hat{\tau}=\tau}^{T} z_{i\tau\hat{\tau}} d_{i\hat{\tau}}^{L4L} \qquad \begin{matrix} j = 1, \ldots, J \\ t = 0, \ldots, T-1 \end{matrix} \tag{13}$$

$$\sum_{j \in \mathcal{J}_m} y_{jt} \leq 1 \qquad \begin{matrix} m = 1, \ldots, M \\ t = 1, \ldots, T \end{matrix} \tag{14}$$

$$x_{jt} \geq y_{jt} - y_{j(t-1)} \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \end{matrix} \tag{15}$$

$$z_{jt\tau} \leq y_{j(t-1)} + y_{jt} \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \\ \tau = t, \ldots, T \end{matrix} \tag{16}$$

$$\sum_{j \in \mathcal{J}_m} p_j \sum_{\tau=t}^{T} z_{jt\tau} d_{j\tau}^{L4L} \leq C_{mt} \qquad \begin{matrix} m = 1, \ldots, M \\ t = 1, \ldots, T \end{matrix} \tag{17}$$

$$\sum_{\tau=1}^{t} z_{j\tau t} \leq 1 \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \end{matrix} \tag{18}$$

$$y_{jt} \in \{0, 1\} \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \end{matrix} \tag{19}$$

$$I_{jt}, x_{jt} \geq 0 \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \end{matrix} \tag{20}$$

$$z_{jt\tau} \geq 0 \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \\ \tau = t, \ldots, T \end{matrix} \tag{21}$$

To get this model formulation we simply replaced the decision variables q_{jt} in the PLSP–MM model given in Section 2 with $\sum_{\tau=t}^{T} z_{jt\tau} d_{j\tau}^{L4L}$. Furthermore, we added the constraints (18) which reflect the fact that the new decision variable represents a fraction of gross demand. Noteworthy to say that $\sum_{\tau=1}^{t} z_{j\tau t} = 1$ for $j = 1, \ldots, J$ and $t = 1, \ldots, T$ is not valid in the presence of initial inventory. This, by the way, allows us to simplify the constraints (6) now stated as (16).

3.2 Experimental Evaluation

In a computational study we compare the LP–relaxation of the original PLSP–model with the LP–relaxation of the simple plant location model for the PLSP. As a test–bed we use the small PLSP–instances given in [23]. This test–bed consists of a total of 1,033 instances for which a feasible solution exists. These instances were generated using a full factorial experimental design where five parameters are systematically varied: U (the capacity utilization), M (the number of machines), C (the gozinto–structure complexity as defined in [23]), $COSTRATIO$ (the ratio of setup and holding costs), and $(T_{macro}, T_{micro}, T_{idle})$ (the demand pattern). For all these instances we have $J = 5$ and $T = 10$ which is small enough to compute optimum solutions and large enough to give non–trivial instances. Our measure of performance is the so–called integrality gap. For each instance this deviation is computed as

$$deviation \stackrel{def}{=} 100\frac{LB - OPT}{OPT} \tag{22}$$

where LB is the lower bound determined by means of the LP–relaxation and OPT is the optimum objective function value of the PLSP–instance. Note, a deviation close to zero is desired.

To ease an analysis of these results we aggregate the data. Table 4 reveals the impact of the parameter M and provides the average integrality gap. Apparently, increasing the number of machines reduces the deviation from the optimum results decidedly. This matches a former result given in [20] where it was shown that, if $M = J$ which is the maximum number of machines in a multi–machine setting, the problem of optimizing the PLSP–MIP–model formulation reduces to be an LP–problem.

	$M = 1$	$M = 2$
PLSP–Model	-55.49	-40.96
Simple Plant Location Model	-47.70	-37.30

Table 4: The Impact of the Number of Machines on the Integrality Gap

Whether or not the complexity of the gozinto–structure plays a role can be read in Table 5. Though the impact of the complexity is not dramatic, it seems that gozinto–structures with a high complexity give an average gap that is slightly closer for the original PLSP–model formulation. When solving a plant location model, gozinto–structure complexity has almost no effect.

In Table 6 we see what happens to the lower bound if the demand pattern changes. While for the original PLSP–model a sparsely–filled demand matrix gives the best result, for the plant location model a demand matrix with many non–zeroes turns out to be advantageous on average. The original PLSP–model

	$C = 0.2$	$C = 0.8$
PLSP–Model	-49.48	-46.88
Simple Plant Location Model	-42.68	-42.29

Table 5: The Impact of the Gozinto–Structure Complexity on the Integrality Gap

	$(T_{macro}, T_{micro}, T_{idle}) =$		
	$(10,1,5)$	$(5,2,2)$	$(1,10,0)$
PLSP–Model	-48.52	-51.72	-44.71
Simple Plant Location Model	-40.47	-44.43	-42.71

Table 6: The Impact of the Demand Pattern on the Integrality Gap

seems to be more sensitive to different demand patterns, because the variance of the average results is greater than for the simple plant location model.

The ratio of setup and holding costs significantly affects the quality of the lower bounds as can be seen in Table 7. Low setup costs result in small integrality gaps for both, the original PLSP–model and the simple plant location model. The higher the setup costs are, the greater the gap. The explanation for this phenomenon is that in an LP–relaxation only a small fraction of the actual setup costs are charged. If setup costs are low, the optimum objective function value of a PLSP–instance mainly is a sum of holding costs. If setup costs are large, the objective function value in large parts is a sum of setup costs. Hence, LP–relaxations perform better when setup costs are low. However, even if setup costs would be zero the LP–relaxation would not necessarily give a zero integrality gap. This is, because in a solution of the LP–relaxation the setup state of a machine is no longer uniquely defined, by definition. As a consequence, more than two items may be produced per period which leads to infeasible production plans and to an underestimation of total holding costs.

	$COSTRATIO =$		
	5	150	900
PLSP–Model	-24.72	-50.72	-69.29
Simple Plant Location Model	-18.70	-44.66	-64.23

Table 7: The Impact of the Cost Structure on the Integrality Gap

The capacity utilization is assumed to be of great importance for the per-

formance of lot sizing and scheduling. For computing lower bounds via an LP–relaxation, however, Table 8 shows that the capacity utilization is not the reason for dramatic performance differences. For the original PLSP–model varying the capacity utilization gives no significant changes. For the simple plant location model there is a tendency that the quality of the lower bounds decreases if the capacity utilization increases.

	$U = 30$	$U = 50$	$U = 70$
PLSP–Model	-47.97	-48.53	-48.10
Simple Plant Location Model	-40.10	-42.59	-45.04

Table 8: The Impact of the Capacity Utilization on the Integrality Gap

Using LINGO to solve the LP–relaxations takes on average less than a minute per instance running on a Pentium P120 computer no matter what model is used.

In summary, we can state that the performance of the simple plant location model formulation slightly outperforms the original PLSP–model formulation. The overall average result for the former is a -42.49% integrality gap, while the latter yields a gap of -48.20% on average. In contrast to lot sizing without scheduling, a plant location reformulation for lot sizing and scheduling does not give a sharp lower bound by solving the LP–relaxation. Hence, we need other ways to determine lower bounds.

4 Capacity Relaxation

4.1 Motivation for Relaxing Capacity Constraints

Many hard–to–solve problems, and so the PLSP–MM, can be modelled as an easier–to–solve problem complicated by a set of constraints. Removing this set of constraints from the PLSP–model yields a model which defines an optimum objective function value that is a lower bound for the PLSP. So, we should find out which constraints make instances of the PLSP–MM–model hard to solve, remove these constraints, and develop an exact solution procedure for the remaining problem which may then be used to find lower bounds. The fundamental motivation here is that a tailor–made method is much more efficient (primarily in terms of run–time) than a standard MIP–solver.

As we have learned in the preceding section, violating the binary conditions for the setup state variables gives poor results. Thus, more promising approaches should respect integrality. To see what else makes the PLSP–MM be a hard–to–solve problem, suppose that all binary variables are fixed, say, to the values that occur in an optimum solution. What is left then is a linear program still not easy to solve (optimally). Of course, standard LP–solvers may be used,

but, as we shall see we require more efficient procedures. For example, as it is discussed in [23], lot splitting may need to occur which makes the computation of production quantities a non–trivial task. So, the question is which constraints must be dropped to give a problem in which lot splitting does not occur any more. Let us drop the capacity constraints of the PLSP–MM and assume that initial inventory is zero. The latter assumption is restrictive. But, remember that lower bounds will later on be used to test the performance of PLSP–MM–heuristics. As we expect the initial inventory to have no significant impact on the performance of these heuristics, a test–bed with no initial inventory seems to be sufficient. The resulting uncapacitated PLSP–MM is denoted as U–PLSP for short.

Under these assumptions optimum production quantities equal the sum of subsequent order sizes. More formally, in an optimum solution of an U–PLSP–instance we either have $q_{jt} = 0$ or

$$q_{jt} = CD_{jt} \stackrel{def}{=} \sum_{\tau=t}^{\hat{t}} d_{jt} + \sum_{i \in \mathcal{S}_j} \sum_{\tau=t+v_j}^{min\{\hat{t}+v_j, T\}} a_{ji} q_{i(\tau+v_j)} \tag{23}$$

for $j = 1, \ldots, J$, $t = 1, \ldots, T$, and some $\hat{t}$ for which $t \leq \hat{t} \leq T$ holds. CD_{jt} denotes the cumulative future demand for item j in period t not been met by production in periods later than t. Thus, the matrix of production quantities can also be represented by an integer matrix

$$mask_{jt} \stackrel{def}{=} \begin{cases} 0 & \text{, if } q_{jt} = 0 \\ > 0 & \text{, if } q_{jt} > 0 \end{cases} \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \end{matrix} \tag{24}$$

to which we will refer to as a production mask.

On the other hand, given a $mask_{jt}$–matrix the q_{jt}–matrix can uniquely be restored using the following rule for each pair of indices $j = 1, \ldots, J$ and $t = 1, \ldots, T$: If $mask_{jt} = 0$ then $q_{jt} = 0$. If $mask_{jt} > 0$ then q_{jt} is computed using formula (23) with

$$\hat{t} = min(\{\tau \mid t \leq \tau < T \wedge mask_{j(\tau+1)} > 0\} \cup \{T\}). \tag{25}$$

Bringing things together, to solve an U–PLSP–instance optimally, we may enumerate all y_{jt}–matrices where (4) must hold. Given a y_{jt}–matrix, a x_{jt}–matrix that causes minimum setup costs can easily be derived by setting $x_{jt} = 1$ if $y_{j(t-1)} = 0$ and $y_{jt} = 1$ to fulfill (5). Otherwise, $x_{jt} = 0$. Due to the results given in [23], optimum schedules need not be semi–active. Hence, for each setup state matrix we must then enumerate all $mask_{jt}$–matrices where

$$mask_{jt} \leq y_{j(t-1)} + y_{jt} \leq 2 \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \end{matrix} \tag{26}$$

must be valid because of (6). Having both, a setup state matrix and a production mask (which is a representation of the production quantities), we must test for feasibility using formulae (2), (3) and $I_{jt} > 0$ for $j = 1, \ldots, J$ and $t = 1, \ldots, T$.

In summary, it turns out that solving an U–PLSP–instance essentially reduces to enumerating integer matrices which, when guided by problem specific insight, promises to be much more efficient than using standard MIP–solvers.

4.2 Basic Enumeration Scheme

The basic working principle for enumerating setup states and production masks is a backward oriented depth–first search moving on from period T to period 1. Let period t be the current focus of attention. We first choose the setup state for each machine where $j_{mt} \in \mathcal{J}_m$ is the item machine m is set up for at the end of period t. Then, we decide for $mask_{j_{mt}(t+1)}$ indicating whether or not item j_{mt} is produced in period $t+1$ which would be allowed, because the setup state at the beginning of period $t+1$ (equal to the setup state at the end of period t) is properly set. Once this is done, $mask_{j_{mt}t}$ is set. Next, period $t-1$ is concerned following the same lines.

More formally, a recursive procedure $uplsp(t, \Delta t, m, p)$ defines the details. Four parameters are passed to this method: $t \in \{1, \ldots, T\}$ the period under concern, $\Delta t \in \{0, 1\}$ indicating whether the setup state in period t should ($\Delta t = 1$) or should not ($\Delta t = 0$) be set, $m \in \{1, \ldots, M\}$ the machine under concern, and $p \in \{0, 1\}$ the value to be used for computing a production mask entry. The idea is to consider the production mask in period $t + \Delta t$. The initial call is $uplsp(T, 1, 1, 1)$ where all $mask_{jt}$– and y_{jt}–values are initialized with zero for $j = 1, \ldots, J$ and $t = 1, \ldots, T$. How to evaluate a call of the form $uplsp(T, 1, \cdot, \cdot)$ is defined in Table 9.

$ITEMSET_{mT} := \mathcal{J}_m$.
while $(ITEMSET_{mT} \neq \emptyset)$
    choose $j_{mT} \in ITEMSET_{mT}$.
    $ITEMSET_{mT} := ITEMSET_{mT} \backslash \{j_{mT}\}$.
    $y_{j_{mT}T} := 1$.
    if $(m = M)$
        $uplsp(T, 0, 1, 1)$.
    else
        $uplsp(T, 1, m+1, 1)$.
    $y_{j_{mT}T} := 0$.

Table 9: Evaluating $uplsp(T, 1, \cdot, \cdot)$

If $uplsp(T, 1, \cdot, \cdot)$ is called, the parameter p is of no relevance, because choosing a production mask for period $T+1$ does not make sense. It is important to understand that once we return from a recursive call to the *uplsp*–procedure, the calling procedure loops back choosing another setup state and starting all over again until all setup states are enumerated. Moving stepwise from $m = 1$

to $m = M$, we assign a setup state to every machine at the end of period T. Afterwards, a call to $uplsp(T, 0, 1, 1)$ is made to decide for the production mask in period T. Table 10 gives more details.

$mask_{j_{mt}t} := mask_{j_{mt}t} + p$.
if $(m = M)$
    $uplsp(t-1, 1, 1, 1)$.
else
    $uplsp(t, 0, m+1, 1)$.
$mask_{j_{mt}t} := mask_{j_{mt}t} - p$.
if $(p = 1)$
    $uplsp(t, 0, m, 0)$.

Table 10: Evaluating $uplsp(t, 0, \cdot, \cdot)$ where $1 \leq t \leq T$

Note, evaluating $uplsp(t, 0, \cdot, \cdot)$ does not require to choose a setup state. The recursive call to $uplsp(t, 0, m, 0)$ implements the enumeration for the values of the parameter p. What is new in this scheme is the call to $uplsp(t-1, 1, \cdot, \cdot)$ to enumerate the setup states at the end of period $t-1$. Table 11 provides an implementation of its evaluation.

$ITEMSET_{mt} := \mathcal{J}_m$.
while $(ITEMSET_{mt} \neq \emptyset)$
    choose $j_{mt} \in ITEMSET_{mt}$.
    $ITEMSET_{mt} := ITEMSET_{mt} \backslash \{j_{mt}\}$.
    $y_{j_{mt}t} := 1$.
    $mask_{j_{mt}(t+1)} := mask_{j_{mt}(t+1)} + p$.
    if $(m = M)$
        $uplsp(t, 0, 1, 1)$.
    else
        $uplsp(t, 1, m+1, 1)$.
    $mask_{j_{mt}(t+1)} := mask_{j_{mt}(t+1)} - p$.
    $y_{j_{mt}t} := 0$.
if $(p = 1)$
    $uplsp(t, 1, m, 0)$.

Table 11: Evaluating $uplsp(t, 1, \cdot, \cdot)$ where $1 \leq t < T$

For the special case $t = 0$ the evaluation of $uplsp(t, 1, \cdot, \cdot)$ is given in Table 12. The difference to what is given in Table 11 is that for $t = 0$ we have no choice for the setup state, because y_{j0} is given as a parameter for $j = 1, \ldots, J$. Let

j_{m0} denote the unique item machine m is initially set up for (assume $j_{m0} = 0$ if machine m is initially setup for no item).

---

if $(j_{m0} \neq 0)$
    $mask_{j_{m0}1} := mask_{j_{m0}1} + p.$
if $(m = M)$
    $uplsp(0, 0, 1, 1).$
else
    $uplsp(0, 1, m + 1, 1).$
if $(j_{m0} \neq 0)$
    $mask_{j_{m0}1} := mask_{j_{m0}1} - p.$

Table 12: Evaluating $uplsp(0, 1, \cdot, \cdot)$

A call to $uplsp(0, 0, \cdot, \cdot)$ indicates a terminal node in which a setup state matrix and a production mask matrix are completely defined. If such a node is reached, we simply have to check feasibility and evaluate the corresponding production plan as described in Subsection 4.1. If this plan is feasible and it improves the current best plan, we memorize it. After a complete enumeration we thus have found an optimum solution for an U–PLSP–instance.

4.3 Branching Rules

We perform a depth–first search. So, a degree of freedom that remains for branching is the sequence in which setup states are enumerated which is represented by the line

$$\text{choose } j_{mt} \in ITEMSET_{mt}$$

in Tables 9 and 11 where $1 \leq t \leq T$.

Using some priority rule which assigns a priority value $priority_{jt}$ to each item $j \in ITEMSET_{mt}$, items may be chosen in decreasing order with respect to their priority values (ties might be broken with respect to the item index for example).

Two different kinds of priority rules are worth to be discriminated. On the one hand, we may use static rules which depend on the item indices and/or period indices only. On the other hand, we may use dynamic rules which depend on the history of the execution as well. The advantage of the former ones is that items may be sorted before the enumeration starts while the latter ones cause additional overhead for sorting items over and over again whenever the code represented by Tables 9 and 11 is called.

Some examples for static priority rules are the item index itself

$$priority_{jt} \stackrel{def}{=} \frac{1}{j}, \tag{27}$$

a setup cost based rule

$$priority_{jt} \stackrel{def}{=} \frac{1}{s_j}, \tag{28}$$

or a capacity demand oriented rule

$$priority_{jt} \stackrel{def}{=} \sum_{\tau=1}^{t} p_j d_{j\tau}^{L4L}. \tag{29}$$

A dynamic rule can be given as

$$priority_{jt} \stackrel{def}{=} \frac{h_j CD_{jt}}{s_j} \tag{30}$$

where CD_{jt} is defined as in (23) using (25) to determine $\hat{t}$. Note, CD_{jt} and hence $priority_{jt}$ cannot be computed before the enumeration starts. But, since CD_{jt} is a cumulative value, it can efficiently be computed while moving stepwise from period to period adding up those demands which are not scheduled.

We use the dynamic rule (30) to compute priority values in our implementation. Its interpretation is that for not fulfilling future demand extra holding costs are charged. If setup costs are low, this tends to give bad solutions. But, if setup costs are high, building lots tends to be a good idea.

4.4 Bounding Rules

The enumeration scheme presented in Subsection 4.2 performs a complete enumeration and uses no insight information to prune the search tree. Hence, we should develop some bounding rules to reduce the computational effort.

First, let us consider the set of items to choose among in order to fix the setup state of a machine m. Both, in Table 9 and in Table 11, we defined $ITEMSET_{mt} = \mathcal{J}_m$ for initialization. Since switching the setup state is necessary only if production takes place, this item set can usually be chosen smaller. At the end of period T (Table 9) a machine m is set up for the item that is produced last on that machine. Any item for which external demand occurs could be that item. If no external demand occurs for an item, it can only be the last one on a machine if there is no successor item sharing the same machine. Hence,

$$\begin{aligned} ITEMSET_{mT} = \; & \left\{ j \in \mathcal{J}_m \mid \textstyle\sum_{t=1}^{T} d_{jt} > 0 \right\} \\ & \cup \left\{ j \in \mathcal{J}_m \mid \{ i \in \bar{\mathcal{S}}_j \mid m_i = m_j \} = \emptyset \right\} \end{aligned} \tag{31}$$

is a valid choice. For periods t where $1 \leq t < T$ (Table 11) it is sufficient to consider those items only for which demand occurs in period t or in period $t+1$, or for which the machine is also set up in period $t+1$. The latter condition allows keeping the setup state up. More formally,

$$ITEMSET_{mt} = \{ j \in \mathcal{J}_m \mid CD_{jt} > 0 \} \cup \{ j_{m(t+1)} \} \tag{32}$$

defines the initialization of $ITEMSET_{mt}$.

The production mask matrix is completely enumerated when following the lines of the *uplsp*-procedure. But, since a positive entry $mask_{jt}$ must only be considered if there is (future) demand for item j that is not been met, we can reduce the computational effort as follows: In Table 10 we add

if ($p = 1$ and $CD_{j_{mt}t} = 0$) $p = 0$

at the very beginning to skip the consideration of the value $p = 1$. In Table 11 we simply add

if ($p = 1$) $ITEMSET_{mt} := ITEMSET_{mt}$
$\setminus \{ j \in \mathcal{J}_m \mid CD_{j(t+1)} = 0 \}$

right behind the initialization of $ITEMSET_{mt}$. As a consequence, the production masks being enumerated actually are binary[1] matrices now where positive entries in periods $1 < t \leq T$ do indeed represent that production takes place.

Another way to speed up the enumeration is to detect intermediate states which cannot lead to any feasible solution. For notational convenience, let

$$\mathcal{J}_{mt}^{+} \stackrel{def}{=} \{ j \in \mathcal{J}_m \mid CD_{jt} > 0 \} \tag{33}$$

denote the set of items which share machine m and for which there exists a positive cumulative demand in period t. On the basis of this,

$$\mathcal{J}_{mt}^{++} \stackrel{def}{=} \{ j \in \mathcal{J}_m \cap \bar{\mathcal{P}}_i \mid i \in \bigcup_{\hat{m}=1}^{M} \mathcal{J}_{\hat{m}t}^{+} \} \cup \mathcal{J}_{mt}^{+} \tag{34}$$

defines a set of items which will have to be scheduled on machine m in periods 1 to t. This is true, because we have assumed no initial inventory. Owing to a unique setup state at the end of each period, at the beginning of the procedure given in Table 10 we therefore test

$$\mid \mathcal{J}_{mt}^{++} \mid > t+1 \tag{35}$$

which when true indicates that no feasible solution can be found any more. Similarly, when entering the code given in Tables 11 and 12 we check for

$$\mid \mathcal{J}_{m(t+1)}^{++} \mid > t+1 \tag{36}$$

and initiate a backtracking step depending on its outcome. Since we face multi–level gozinto–structures, it may happen that items causing internal demand for preceding items with respect to positive lead times do not fit into the remaining time window. More precisely, if we enter the code given in Table 10 and if there is an item

$$j \in \bigcup_{\hat{m}=1}^{M} \mathcal{J}_{\hat{m}t}^{+} \text{ where } dep_j \geq t \tag{37}$$

[1] Only in period 1 production mask entries may have the value 2 which is due to the scheme in Table 12. Note, this does not mean unnecessary overhead in period 1, because the code in Table 12 does not enumerate the values for the parameter p.

a setup cost based rule

$$priority_{jt} \stackrel{def}{=} \frac{1}{s_j}, \tag{28}$$

or a capacity demand oriented rule

$$priority_{jt} \stackrel{def}{=} \sum_{\tau=1}^{t} p_j d_{j\tau}^{L4L}. \tag{29}$$

A dynamic rule can be given as

$$priority_{jt} \stackrel{def}{=} \frac{h_j CD_{jt}}{s_j} \tag{30}$$

where CD_{jt} is defined as in (23) using (25) to determine $\hat{t}$. Note, CD_{jt} and hence $priority_{jt}$ cannot be computed before the enumeration starts. But, since CD_{jt} is a cumulative value, it can efficiently be computed while moving stepwise from period to period adding up those demands which are not scheduled.

We use the dynamic rule (30) to compute priority values in our implementation. Its interpretation is that for not fulfilling future demand extra holding costs are charged. If setup costs are low, this tends to give bad solutions. But, if setup costs are high, building lots tends to be a good idea.

4.4 Bounding Rules

The enumeration scheme presented in Subsection 4.2 performs a complete enumeration and uses no insight information to prune the search tree. Hence, we should develop some bounding rules to reduce the computational effort.

First, let us consider the set of items to choose among in order to fix the setup state of a machine m. Both, in Table 9 and in Table 11, we defined $ITEMSET_{mt} = \mathcal{J}_m$ for initialization. Since switching the setup state is necessary only if production takes place, this item set can usually be chosen smaller. At the end of period T (Table 9) a machine m is set up for the item that is produced last on that machine. Any item for which external demand occurs could be that item. If no external demand occurs for an item, it can only be the last one on a machine if there is no successor item sharing the same machine. Hence,

$$\begin{aligned} ITEMSET_{mT} = \; & \left\{ j \in \mathcal{J}_m \mid \textstyle\sum_{t=1}^{T} d_{jt} > 0 \right\} \\ & \cup \left\{ j \in \mathcal{J}_m \mid \{ i \in \bar{\mathcal{S}}_j \mid m_i = m_j \} = \emptyset \right\} \end{aligned} \tag{31}$$

is a valid choice. For periods t where $1 \leq t < T$ (Table 11) it is sufficient to consider those items only for which demand occurs in period t or in period $t+1$, or for which the machine is also set up in period $t+1$. The latter condition allows keeping the setup state up. More formally,

$$ITEMSET_{mt} = \{ j \in \mathcal{J}_m \mid CD_{jt} > 0 \} \cup \left\{ j_{m(t+1)} \right\} \tag{32}$$

defines the initialization of $ITEMSET_{mt}$.

The production mask matrix is completely enumerated when following the lines of the *uplsp*–procedure. But, since a positive entry $mask_{jt}$ must only be considered if there is (future) demand for item j that is not been met, we can reduce the computational effort as follows: In Table 10 we add

$$\text{if } (p = 1 \text{ and } CD_{j_{mt}t} = 0)\ p = 0$$

at the very beginning to skip the consideration of the value $p = 1$. In Table 11 we simply add

$$\text{if } (p = 1)\ ITEMSET_{mt} := ITEMSET_{mt} \setminus \{j \in \mathcal{J}_m \mid CD_{j(t+1)} = 0\}$$

right behind the initialization of $ITEMSET_{mt}$. As a consequence, the production masks being enumerated actually are binary[1] matrices now where positive entries in periods $1 < t \leq T$ do indeed represent that production takes place.

Another way to speed up the enumeration is to detect intermediate states which cannot lead to any feasible solution. For notational convenience, let

$$\mathcal{J}_{mt}^{+} \stackrel{def}{=} \{j \in \mathcal{J}_m \mid CD_{jt} > 0\} \tag{33}$$

denote the set of items which share machine m and for which there exists a positive cumulative demand in period t. On the basis of this,

$$\mathcal{J}_{mt}^{++} \stackrel{def}{=} \{j \in \mathcal{J}_m \cap \bar{\mathcal{P}}_i \mid i \in \bigcup_{\hat{m}=1}^{M} \mathcal{J}_{\hat{m}t}^{+}\} \cup \mathcal{J}_{mt}^{+} \tag{34}$$

defines a set of items which will have to be scheduled on machine m in periods 1 to t. This is true, because we have assumed no initial inventory. Owing to a unique setup state at the end of each period, at the beginning of the procedure given in Table 10 we therefore test

$$\mid \mathcal{J}_{mt}^{++} \mid > t + 1 \tag{35}$$

which when true indicates that no feasible solution can be found any more. Similarly, when entering the code given in Tables 11 and 12 we check for

$$\mid \mathcal{J}_{m(t+1)}^{++} \mid > t + 1 \tag{36}$$

and initiate a backtracking step depending on its outcome. Since we face multi–level gozinto–structures, it may happen that items causing internal demand for preceding items with respect to positive lead times do not fit into the remaining time window. More precisely, if we enter the code given in Table 10 and if there is an item

$$j \in \bigcup_{\hat{m}=1}^{M} \mathcal{J}_{\hat{m}t}^{+} \text{ where } dep_j \geq t \tag{37}$$

[1] Only in period 1 production mask entries may have the value 2 which is due to the scheme in Table 12. Note, this does not mean unnecessary overhead in period 1, because the code in Table 12 does not enumerate the values for the parameter p.

then the current node is fathomed. Analogously, when the code in Tables 11 and 12 is entered and if there is an item

$$j \in \bigcup_{\hat{m}=1}^{M} \mathcal{J}_{\hat{m}t}^{+} \text{ where } dep_j > t \tag{38}$$

then the current node is fathomed, too.

Eventually, we can prune the search tree on the basis of cost criteria. Let $costs^{PS}$ be the sum of setup and holding costs for a partial schedule ranging from periods $t+1$ to T. When initialized with zero when we start the enumeration, i.e. $costs^{PS} = 0$, it can easily be computed as we pass through the search tree. More precisely, consider what is done in Table 11. After j_{mt} is selected, we compute

$$costs^{PS} := costs^{PS} + s_{j_{m(t+1)}} \tag{39}$$

if $j_{mt} \neq j_{m(t+1)}$, because a setup then takes place in period $t+1$. Having added p to $mask_{j_{mt}(t+1)}$, we compute

$$costs^{PS} := costs^{PS} + \sum_{j \in \mathcal{J}_m} h_j CD_{j(t+1)} \tag{40}$$

for holding the cumulative demand one (more) period in inventory. Note, if $p = 1$ and thus $mask_{j_{mt}(t+1)} > 0$, then $CD_{j_{mt}(t+1)}$ evaluates to zero. Since backtracking may occur, we must not forget to perform the reverse operations

$$costs^{PS} := costs^{PS} - s_{j_{m(t+1)}} - \sum_{j \in \mathcal{J}_m} h_j CD_{j(t+1)}. \tag{41}$$

A valid position to execute this operation is, for example, right before the value p is subtracted from $mask_{j_{mt}(t+1)}$. The same operations can be performed within the code given in Table 12. The only difference is that there is no choice for j_{m0} which is the initial setup state. But, this does not affect the determination of $costs^{PS}$. Imagine that we have a lower bound for the minimum costs incurred in periods 1 to t, say, $lowerbound_{1t}$. Furthermore, suppose that we have an upper bound for the optimum solution of the overall instance, say, *upperbound*. Then, once we have increased $costs^{PS}$, we evaluate

$$lowerbound_{1t} + costs^{PS} > upperbound \tag{42}$$

which when true indicates that the choice of j_{mt} does not give an optimum solution. Hence, we can immediately loop back to select another setup state (or, if $t = 0$, initiate backtracking). What is left now is the discussion of determining *upperbound* and $lowerbound_{1t}$. The former follows standard ideas. Starting with an initial value (which might be infinity) we update its value whenever we reach a terminal node that improves the current best upper bound. A better initial value than infinity can be computed with any of the heuristics for the

PLSP–MM which will be described in Chapter 6. For computing $lowerbound_{1t}$ we use a new though simple idea. We just solve the U–PLSP–instance that emerges from the original U–PLSP–instance when restricting our attention to the first t periods only. As a solution method we use the B&B–procedure again. Since this idea can be carried on recursively, we have to solve a sequence of U–PLSP–instances. The working principle should be made a bit more clear now: Set $lowerbound_{10} = 0$. First, we solve a U–PLSP–instance which is the instance that emerges from the original instance when we consider period 1 only. The result gives $lowerbound_{11}$. Then, we solve a U–PLSP–instance which consists of the first two periods of the original instance which yields $lowerbound_{12}$. This goes on until an instance with T periods is eventually solved. The instance with T periods is the original instance which gives the desired result. The remarkable point to note here is that when we are about to solve an instance with, say, t periods, we make use of $lowerbound_{11}, \ldots, lowerbound_{1(t-1)}$ which are previously computed. Since many instances have some periods with no external demand (see the discussion of macro and micro periods in Chapter 4), the stepwise procedure can be accelerated in most cases. If period t is a period with no external demand then solving the instance consisting of the first t periods gives the same objective function value as solving the instance consisting of the first $t-1$ periods. Hence, if $\sum_{j=1}^{J} d_{jt} = 0$ then choose $lowerbound_{1t} = lowerbound_{1(t-1)}$.

4.5 Experimental Evaluation

To study the performance of the presented B&B–procedure, we solve the small PLSP–instances again by running a C–implementation on a Pentium computer with 120 MHz.

For a detailed analysis of the deviation data, we aggregate the results to see whether or not certain parameter levels have a significant impact on the performance. To start with, let us consider the number of machines first. Table 13 gives more insight. As we can see, increasing the number of machines reduces the average performance remarkably. The reason for this is not obvious. A possible explanation might be the following: On the one hand, producing a lot to fulfill a demand may last several periods if capacities are scarce, but takes one period if the capacity constraints are relaxed. Hence, the total holding costs tend to be underestimated. On the other hand, more machines tend to result in lower total holding costs, because items which do not share a common machine may be produced in parallel and need not be sequenced. Both together might explain the observed result, because the underestimation of holding costs is relatively high if total holding costs are low.

Next, we are interested in the impact of the gozinto–structure complexity. Table 14 shows the results. We cannot state that there is any effect.

Table 15 reveals whether or not the demand pattern plays a role in performance changes. It can be seen that sparse demand matrices give poor results.

$M = 1$	$M = 2$
-9.70	-15.19

Table 13: The Impact of the Number of Machines on the Lower Bound

$C = 0.2$	$C = 0.8$
-12.34	-12.57

Table 14: The Impact of the Gozinto–Structure Complexity on the Lower Bound

Demand matrices with many non–zero entries give the best results with less than a 10% deviation on average. A reason for this phenomenon seems to be the fact that lot sizing is more important if there are many demands which can be grouped together in order to form a lot and thus to safe setup costs. These saving opportunities are detected by the B&B–method, too.

$(T_{macro}, T_{micro}, T_{idle}) =$		
$(10, 1, 5)$	$(5, 2, 2)$	$(1, 10, 0)$
-9.88	-11.20	-16.11

Table 15: The Impact of the Demand Pattern on the Lower Bound

If the cost structure is important for a good performance can be read in Table 16. Though differences are not dramatic, there is a tendency for instances with high setup costs to have a better lower bound than instances with low setup costs.

Since we relax the capacity constraints, the capacity utilization is expected to have a significant impact on the performance of the B&B–method. As Table 17 indicates, this is indeed the case. For $U = 30$ we have an average deviation of -5.29% which is a fairly good result. But, for $U = 70$ the average deviation is -21.27% which is quite poor, although it is still better than the LP–relaxation. As a result, we have that the capacity utilization is the most significant factor for the computation of a lower bound via a capacity relaxation. Remarkable to note, the average deviation for $U = 30$ is not very close to zero which indicates that even a low capacity utilization does not make the multi–level lot sizing and scheduling problem be an easy–to–solve problem.

The run–time performance of the B&B–method varies very much. For the instances with $M = 1$, it takes between 0 to 53 CPU–seconds to solve them where zero time actually means that the run–time is too small to be measured.

$COSTRATIO =$		
5	150	900
-13.42	-12.05	-11.89

Table 16: The Impact of the Cost Structure on the Lower Bound

$U = 30$	$U = 50$	$U = 70$
-5.29	-11.73	-21.27

Table 17: The Impact of the Capacity Utilization on the Lower Bound

Most of the instances with $M = 2$ can also be solved within 60 CPU–seconds. Almost all instances can be solved with a time limit of 900 CPU–seconds. Three outliers need a bit less than 3,600 CPU–seconds.

In summary, the overall average deviation from the optimum results is a -12.46% deviation for our test–bed. This result decidedly outperforms the outcome of the LP–relaxation approaches. Hence, the proposed B&B–method provides a means to compute lower bounds for small instances within reasonable time. Lower bounds for medium– to large–size instances can, however, not be determined due to the exponential growth of the computational effort.

5 Lagrangean Relaxation

In the preceding section we removed the set of capacity constraints which complicate solving a PLSP–MM–instance. For the resulting problem we then developed a tailor–made procedure to find lower bounds. A bit more sophisticated than just removing complicating constraints is the general idea of replacing constraints with a penalty term in the objective function. The trick is to define a penalty term which — for minimization problems — increases the objective function value if the removed constraints are violated. Such an approach which additionally guarantees to give lower bounds for the original problem is known as Lagrangean relaxation. A nice introduction to it is given in [7] (see also [9, 6]).

Subsequently, we will describe how to apply a Lagrangean relaxation to the PLSP–MM in order to get a lower bound. Again, the capacity constraints are considered as the complicating constraints. We can therefore use the B&B–procedure described in the preceding section. In contrast what was done above,

we now minimize the objective function

$$\sum_{j=1}^{J}\sum_{t=1}^{T}(s_j x_{jt} + h_j I_{jt}) - \sum_{m=1}^{M}\sum_{t=1}^{T}\lambda_{mt}^{(k)}\Delta capacity_{mt}^{(k)} \tag{43}$$

where

$$\Delta capacity_{mt}^{(k)} \stackrel{def}{=} C_{mt} - \sum_{j\in\mathcal{J}_m} p_j q_{jt}^{(k)} \qquad \begin{matrix} m = 1,\ldots,M \\ t = 1,\ldots,T \end{matrix} \tag{44}$$

gives a negative value, if the capacity of a machine m in period t is exhaustively required. For the moment it suffices here to know that the values $\lambda_{mt}^{(k)}$ are non-negative parameters — so-called Lagrangean multipliers.

A procedure based on Lagrangean relaxation proceeds iteratively and the upper index (k) simply counts the iterations. Roughly speaking, starting with initial values for the Lagrangean multipliers, the resulting instance is solved. Afterwards, the multipliers are modified and a new iteration starts to solve the instance again. This goes on until a stopping criterion is met.

In our context, we begin with

$$\lambda_{mt}^{(1)} = 0 \qquad \begin{matrix} m = 1,\ldots,M \\ t = 1,\ldots,T \end{matrix} \tag{45}$$

which actually means that we solve an U–PLSP–instance which is exactly the same as the one concerned in the preceding section.

After each iteration k we update the Lagrangean multipliers using

$$\lambda_{mt}^{(k+1)} = \max\left\{0, \lambda_{mt}^{(k)} - \delta^{(k)}\frac{(UB^* - LB^*)\Delta capacity_{mt}^{(k)}}{\sum_{\hat{m}=1}^{M}\sum_{\hat{t}=1}^{T}(\Delta capacity_{\hat{m}\hat{t}}^{(k)})^2}\right\} \tag{46}$$

for $m = 1,\ldots,M$ and $t = 1,\ldots,T$ as it is adviced in [15]. UB^* and LB^* are used to denote the smallest known upper bound and the largest known lower bound, respectively, for the PLSP–MM–instance under concern. Note, UB^* can be determined with one of the heuristics described in later chapters. LB^* can be chosen zero initially, and be updated after each iteration that gives a higher objective function value than LB^*. The parameter $\delta^{(k)}$ is a positive value where, again, [15] helps: Initially,

$$\delta^{(1)} = 2. \tag{47}$$

For $k > 1$, if LB^* has not increased within the last, say, $DELTAITER$ iterations then

$$\delta^{(k+1)} = \frac{\delta^{(k)}}{2}, \tag{48}$$

otherwise,

$$\delta^{(k+1)} = \delta^{(k)}. \tag{49}$$

The parameter $DELTAITER$ is chosen by the user.

The stopping rule to terminate the iteration process depends on several user specified parameters which are defined in Table 18 and which are tested after each iteration. If one of these criteria is fulfilled, the Lagrangean relaxation procedure stops giving LB^* as a lower bound. By construction, this lower bound is greater than or equal to the lower bound computed in the preceding section when we solved U–PLSP–instances.

Symbol	Definition
$MAXGAP$	Terminate if $UB^* - LB^* \leq MAXGAP$.
$MAXITER$	Terminate if $MAXITER$ iterations are performed.
$MINLAMBDA$	Terminate if $\lambda_{mt}^{(k+1)} \leq MINLAMBDA$ for all $m = 1, \ldots, M$ and $t = 1, \ldots, T$.
$TIMELIMIT$	Terminate if the total run–time is greater than $TIMELIMIT$ seconds.

Table 18: Stopping Criteria for the Lagrangean Relaxation Approach

It should be stressed that the B&B–method described in the preceding section can be used almost unchanged as a submodule. All that needs to be modified is the evaluation of production plans which now must use formula (43). The application of the cost–oriented bounding rule needs to compute

$$costs^{PS} := costs^{PS} + \sum_{j \in \mathcal{J}_m} (h_j CD_{j(t+1)} - \lambda_{mt}^{(k)} \Delta capacity_{mt}^{(k)}) \qquad (50)$$

instead of (40). The reverse operations (41) must be adapted likewise. Remember Subsection 4.1 for determining the values $q_{jt}^{(k)} (= q_{jt})$ which in turn are needed for computing the values $\Delta capacity_{mt}^{(k)}$.

5.1 Experimental Evaluation

To study the performance of the Lagrangean relaxation procedure we solve the 1,033 small PLSP–MM–instances on the Pentium P120 computer again. The method parameters are chosen as follows: $DELTAITER = 5$, $MAXGAP = 0.001$, $MAXITER = 1,000$, $MINLAMBDA = 0.0001$, and $TIMELIMIT = 3,600$.

The discussion of the results on the basis of aggregated data begins with the impact of the number of machines on the performance. Table 19 shows the numbers. As for the U–PLSP solution, the findings are that the average gap is positively correlated with the number of machines.

$M = 1$	$M = 2$
-3.82	-7.34

Table 19: The Impact of the Number of Machines on the Lower Bound

The effect of changing the complexity of the gozinto–structure is analyzed in Table 20. Although differences are not dramatic, gozinto–structures which are more complex tend to result in slightly larger deviations.

$C = 0.2$	$C = 0.8$
-5.26	-5.92

Table 20: The Impact of the Gozinto–Structure Complexity on the Lower Bound

The demand pattern is the focus of interest in Table 21. While there is a strong tendency for the U–PLSP to yield large deviations if the demand matrix is sparsely filled, the Lagrangean relaxation remedies this phenomenon and now gives best results for this case. However, there is not a clear tendency that more demand entries reduce the performance, because the results for the parameter level $(T_{macro}, T_{micro}, T_{idle}) = (10, 1, 5)$ are better than for $(T_{macro}, T_{micro}, T_{idle}) = (5, 2, 2)$.

$(T_{macro}, T_{micro}, T_{idle}) =$		
$(10, 1, 5)$	$(5, 2, 2)$	$(1, 10, 0)$
-5.59	-6.11	-5.11

Table 21: The Impact of the Demand Pattern on the Lower Bound

Table 22 reveals the impact of the cost structure on the performance. In contrast to the results in the preceding section, we now have that low setup costs give the best results on average. The average deviation is positively correlated with the ratio of setup and holding costs.

For the capacity utilization we observe again that a high utilization results in a significantly worse deviation on average than a low utilization. Table 23 makes this apparent.

The average deviation for all 1,033 instances is -5.59% which is a satisfying result.

$COSTRATIO =$		
5	150	900
-4.58	-5.18	-7.00

Table 22: The Impact of the Cost Structure on the Lower Bound

$U = 30$	$U = 50$	$U = 70$
-2.20	-4.81	-10.23

Table 23: The Impact of the Capacity Utilization on the Lower Bound

6 Summary of Evaluation

The computational study reveals that the we obtain no satisfying lower bounds by solving the LP–relaxation of the original PLSP–model formulation or a simple plant location formulation. Hence, a B&B–method is proposed to solve the uncapacitated PLSP optimally. The lower bound that we get this way is decidedly better. A Lagrangean relaxation of the capacity constraints improves these bounds. Table 24 summarizes the overall average results.

	Average Gap
LP–Relaxation of the PLSP–Model	-48.20
LP–Relaxation of the Simple Plant Location Model	-42.49
Solution of the Uncapacitated PLSP	-12.46
Lagrangean Relaxation of the Capacity Constraints	-5.59

Table 24: Summary of Lower Bounding Methods

Acknowledgement

This work was done with partial support from the DFG–project Dr 170/4–1. We are indebted to Andreas Drexl for his steady support.

References

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

[2] 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

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

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

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

[6] FISHER, M.L., (1981), The Lagrangian Relaxation Method for Solving Integer Programming Problems, Management Science, Vol. 27, pp. 1–18

[7] FISHER, M.L., (1985), An Applications Oriented Guide to Lagrangian Relaxation, Interfaces, Vol. 15, No. 2, pp. 10–21

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

[9] GEOFFRION, A.M., (1974), Lagrangian Relaxation and its Uses in Integer Programming, Mathematical Programming Study, Vol. 2, pp. 82–114

[10] GÜNTHER, H.O., (1987), Planning Lot Sizes and Capacity Requirements in a Single-Stage Production System, European Journal of Operational Research, Vol. 31, pp. 223–231

[11] HAASE, K., (1993), Capacitated Lot-Sizing with Linked Production Quantities of Adjacent Periods, Working Paper No. 334, University of Kiel

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

[13] HAASE, K., (1996), Capacitated Lot-Sizing with Sequence Dependent Setup Costs, OR Spektrum, Vol. 18, pp. 51–59

[14] HAASE, K., KIMMS, A., (1996), Lot Sizing and Scheduling with Sequence Dependent Setup Costs and Times and Efficient Rescheduling Opportunities, Working Paper No. 393, University of Kiel

[15] HELD, M., WOLFE, P., CROWDER, H.P., (1974), Validation of Subgradient Optimization, Mathematical Programming, Vol. 6, pp. 62–88

[16] HINDI, K.S., (1996), Solving the CLSP by a Tabu Search Heuristic, Journal of the Operational Research Society, Vol. 47, pp. 151–161

[17] VAN HOESEL, S., KOLEN, A., (1994), A Linear Description of the Discrete Lot–Sizing and Scheduling Problem, European Journal of Operational Research, Vol. 75, pp. 342–353

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

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

[20] KIMMS, A., (1994), Optimal Multi–Level Lot Sizing and Scheduling with Dedicated Machines, Working Paper No. 351, University of Kiel

[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., (1996), Multi–Level Lot Sizing and Scheduling — Methods for Capacitated, Dynamic, and Deterministic Models, Ph.D. dissertation, University of Kiel

[24] 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

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

[26] LEE, H.L., BILLINGTON, C., (1993), Material Management in Decentralized Supply Chains, Operations Research, Vol. 41, pp. 835–847

[27] LEE, H.L., BILLINGTON, C., CARTER, B., (1993), Hewlett–Packard Gains Control of Inventory and Service through Design for Localization, Interfaces, Vol. 23, No. 4, pp. 1–11

[28] LOTFI, V., CHEN, W.H., (1991), An Optimal Algorithm for the Multi–Item Capacitated Production Planning Problem, European Journal of Operational Research, Vol. 52, pp. 179–193

[29] MAES, J, VAN WASSENHOVE, L.N., (1988), Multi–Item Single–Level Capacitated Dynamic Lot–Sizing Heuristics: A General Review, Journal of the Operational Research Society, Vol. 39, pp. 991–1004

[30] 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

[31] 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

[32] SIMPSON, N.C., ERENGUC, S.S., (1994), Multiple Stage Production Planning Research: History and Opportunities, Working Paper, State University of New York at Buffalo

[33] STADTLER, H., (1994), Mixed Integer Programming Model Formulations for Dynamic Multi–Item Multi–Level Capacitated Lotsizing, Working Paper, Technical University of Darmstadt

[34] STADTLER, H., (1995), Reformulations of the Shortest Route Model for Dynamic Multi-Item Multi-Level Capacitated Lotsizing, Working Paper, Technical University of Darmstadt

[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] 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