Kimms, Alf

Working Paper — Digitized Version

Parameter controlled instance generation for lot sizing problems

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 412

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 412

Parameter Controlled Instance Generation for Lot Sizing Problems

A. Kimms

No. 412

Parameter Controlled Instance Generation for Lot Sizing Problems

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

This contribution defines an instance generator for capacitated, dynamic, multi–level lot sizing problems. It provides sophisticated methods for generating systematically varied parameter level combinations automatically. This helps to perform evaluations of lot sizing methods according to a factorial experimental design and to establish standard test–beds.

Keywords: Instance generator, experimental design, lot sizing

1 Introduction

An important section in high–quality research papers which present solution methods for a certain problem is the one that is concerned with the competitive edge. To allow a fair and profound comparison of new methods with existing ones, both, the new and the old ones should be applied to the same test–bed for an assessment of their performance. If you are lucky, there has a standard test–bed been established in the literature for the problem you deal with. If not, it is up to you to define a test–bed which guarantees fairness. Moreover, not only for comparing two or more methods, but also for introducing a method for a problem that has not been attacked before one should use a test–bed that gives insight under which conditions, i.e. problem parameter settings, a certain method performs good or bad.

Some general remarks about the design of test–beds can be found in [1, 5]. Rather than using a random experimental design, authors should use factorial designs to evaluate their methods. This provides the opportunity to vary certain parameter values systematically in order to study the impact of destinct parameter level combinations on the performance.

A way to achieve the goal of constructing test–beds where the parameter settings are systematically varied is to specify instance generators. The advantage of these is that different instances can easily be generated. Making the instance generator available to the public helps to establish standard test–beds, because it usually is much easier to communicate some input values for that program instead of hundreds of data sets.

For the (resource constrained) project scheduling problem (RCPSP), for example, an instance generator has recently been described in [7]. For the (capacitated, dynamic, multi–level) lot sizing problem (CLSP) and variants which are the subject of our concern, no instance generator has been published (see [6] for an extensive review of the research work on that topic). Hence, we will close that gap.

Throughout the text we make use of some short–hand notation. To draw a real–valued random variable with uniform distribution out of an interval $[a, b]$ we write $\in RRAND[a, b]$. Analogously, $\in IRAND[a, b]$ is used to draw an integer–valued random variable. Generating uniformly distributed random numbers is based on the random number generator described in [8]. If x is real–valued, $\lfloor x \rfloor$ ($\lceil x \rceil$) denotes the greatest (smallest) integer value that is less (greater) than or equal to x.

2 Parameters

An instance of a capacitated, multi–level lot sizing problem can be characterized by a set of parameters. For our purposes, we consider the parameters given in Table 1. See [6] for lot sizing models which are defined using this notation.

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.
J	Number of items.
M	Number of machines.
m_j	Machine on which item j is produced. Let $\mathcal{J}_m$ denote the set of item that share machine m.
p_j	Capacity needs for producing one unit of item j.
s_j	Non–negative setup cost for item j.
T	Number of periods.
v_j	Positive and integral lead time of item j.
y_{j0}	Unique initial setup state.

Table 1: Parameters for a Lot Sizing Problem

All parameters out of J, M, and T (which are user input respecting $M \leq J$) are generated at random. Straightforwardly, we choose

$I_{j0} \in IRAND[INITINV_{min}, INITINV_{max}]$
where $INITINV_{max} \geq INITINV_{min} \geq 0$ are user input,

$m_j \in IRAND[1, M]$
where $\mathcal{J}_m \neq \emptyset$ for all $m = 1, \ldots, M$ must hold,

$p_j \in RRAND[CAPNEED_{min}, CAPNEED_{max}]$
where $CAPNEED_{max} \geq CAPNEED_{min} > 0$ are user input,

$v_j \in IRAND[LEADTIME_{min}, LEADTIME_{max}]$
where $LEADTIME_{max} \geq LEADTIME_{min} > 0$ are user input,

$y_{j0} \in IRAND[0, 1]$
where $\sum_{j \in \mathcal{J}_m} y_{j0} \leq 1$ for all $m = 1, \ldots, M$ must hold.

The remaining parameters need a more sophisticated strategy to be generated randomly.

3 Generation of Gozinto–Structures

The input parameters for generating acyclic gozinto–structures are given in Table 2.

In the sequel we confine ourself to general gozinto–structures ($TYPE_G =$ *general*) in order to avoid too much technical detail. The generation of linear, assembly, or divergent gozinto–structures is roughly the same. Some hints for what changes is spread in the text and should be sufficient.

Symbol	Definition
$[ARC_{min}, ARC_{max}]$	Interval of gozinto–factors where $ARC_{max} \geq ARC_{min} \geq 0$. To generate single–level instances one may choose $ARC_{min} = ARC_{max} = 0$.
$COMPLEXITY$	Complexity of the gozinto–structure.
$DEPTH$	Largest low level code of items in the gozinto–structures.
$ITEMS(0)$	Number of end items.
J	Number of items, where $ITEMS(0) + DEPTH \leq J$.
$MAXITER_G$	Maximum number of iterations where $MAXITER_G \geq N_G$.
N_G	Number of different gozinto–structures to be created.
$PRIVAL(llc)$	A priority value for $llc = 1, \ldots, DEPTH$ which is used to define a discrete probability function that helps to shape the gozinto–structure where $PRIVAL(llc) > 0$ for all $llc = 1, \ldots, DEPTH$.
$TYPE_G$	Type of the gozinto–structure, i.e. $TYPE_G \in \{linear, assembly, divergent, general\}$.

Table 2: Parameters for Generating Gozinto–Structures

Once the generation of gozinto–structures is started, we first decide how many items should have what low level code. Let $ITEMS(llc)$ where $llc = 1, \ldots, DEPTH$ denote the number of items with low level code llc. We must guarantee $ITEMS(llc) \geq 1$ for all $llc = 1, \ldots, DEPTH$ to come up with a structure of the desired depth. Such a structure must exist since $ITEMS(0) + DEPTH \leq J$ holds. At the end

$$\sum_{llc=0}^{DEPTH} ITEMS(llc) = J \tag{1}$$

must hold. Low level codes are assigned at random using a discrete probability function which is defined on the basis of $PRIVAL(llc)$ for all $llc = 1, \ldots, DEPTH$. Formally, if $Z \in RRAND[0,1]$ then

$$level(Z) \stackrel{def}{=} \min \left\{ llc \in \{1, \ldots, DEPTH\} \mid \frac{\sum_{n=1}^{llc} PRIVAL(n)}{\sum_{n=1}^{DEPTH} PRIVAL(n)} \geq Z \right\} \tag{2}$$

is the low level code to choose next. The piece of code in Table 3 gives a precise definition of what is done.

$ITEMS(llc) := 1$ for all $llc = 1, \ldots, DEPTH$.
while $\sum_{llc=0}^{DEPTH} ITEMS(llc) < J$
    choose $Z \in RRAND[0,1]$.
    $ITEMS(level(Z)) := ITEMS(level(Z)) + 1$.

Table 3: Method to Construct a Gozinto–Structure, Part 1

Figure 1 shows a possible outcome for $J = 6$, $ITEMS(0) = 1$, and $DEPTH = 2$.

If linear or divergent gozinto–structures shall be constructed, we must respect $ITEMS(llc) \leq ITEMS(llc - 1)$ for all $llc = 1, \ldots, DEPTH$ and keep this condition true during execution.

In the next step, we construct a gozinto–structure where exactly $ITEMS(llc)$ items have a low level code $llc = 0, \ldots, DEPTH$. We will use the minimum number of arcs to do so. Without loss of generality we assume that items that should have a low level code llc are numbered consecutively from $1 + \sum_{n=0}^{llc-1} ITEMS(n)$ to $\sum_{n=0}^{llc} ITEMS(n)$ (see Figure 1). This, by the way, guarantees a technological ordering. The basic idea now is simple. Just introduce exactly one arc pointing from an item that should have a low level code llc where $llc = 1, \ldots, DEPTH$ to any item that should have a low level code $llc - 1$. Figure 2 shows a possible result for the example where $ARC_{min} = 1$ and $ARC_{max} = 2$. More formally, this part of the construction of a gozinto–structure can be described as in Table 4.

The resulting gozinto–structure meets its specification with a minimum number of arcs, i.e. $J - ITEMS(0)$ arcs. Additional arcs can now be introduced in order to make the structure more "complex". Let

$$\chi(x) \stackrel{def}{=} \begin{cases} 1 & \text{, if } x > 0 \\ 0 & \text{, otherwise} \end{cases} \tag{3}$$

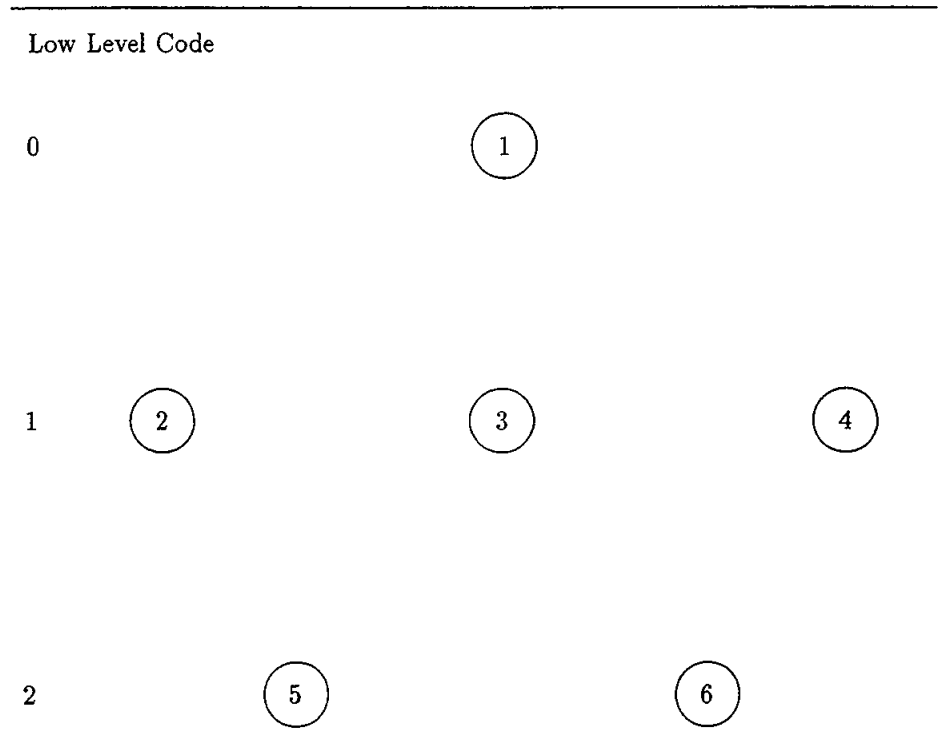

Figure 1: A Possible Assignment of Low Level Codes to Items

be an auxiliary function. A measure of complexity can be defined as

$$\mathcal{C} \stackrel{def}{=} \frac{numarcs - minarcs}{maxarcs - minarcs} \tag{4}$$

where

$$numarcs \stackrel{def}{=} \sum_{j=1}^{J} \sum_{i=1}^{j-1} \chi(a_{ji}) \tag{5}$$

is the number of arcs in the gozinto–structure,

$$minarcs \stackrel{def}{=} J - ITEMS(0) \tag{6}$$

is the minimum number of arcs in a gozinto–structure with J items given the number $ITEMS(0)$ of end items, and

$$maxarcs \stackrel{def}{=} \sum_{llc=0}^{DEPTH-1} \left(ITEMS(llc) \sum_{n=llc+1}^{DEPTH} ITEMS(n) \right) \tag{7}$$

is the maximum number of arcs in a gozinto–structure with J items given the number $ITEMS(llc)$ of items per low level for $llc = 0, \ldots, DEPTH$. The complexity $\mathcal{C}$ of a multi–level gozinto–structure is a real value where $0 \leq \mathcal{C} \leq 1$ holds. Roughly speaking, it is a measure that expresses how many arcs could be removed while keeping the same number of items per low level, and it relates this value to the maximum additional number of arcs that could exist. $\mathcal{C}$ evaluates to

$a_{ji} = 0$ for all $j, i = 1, \ldots, J$.
for $llc = 1$ to $llc = DEPTH$
    for $j = 1 + \sum_{n=0}^{llc-1} ITEMS(n)$ to $j = \sum_{n=0}^{llc} ITEMS(n)$
        choose $i \in IRAND[\ 1 + \sum_{n=0}^{llc-2} ITEMS(n),$
        $\sum_{n=0}^{llc-1} ITEMS(n)]$.
        $a_{ji} := IRAND[ARC_{min}, ARC_{max}]$.

Table 4: Method to Construct a Gozinto–Structure, Part 2

zero if the gozinto–structure under concern has the minimum number of arcs (see Figure 2 for example), and its value is one if no further arcs can be introduced without changing the number of items per low level.

Discussions on complexity measures for gozinto–structures are rather undone. Only one author, namely Collier [2], has introduced a so–called degree of commonality index

$$C' \stackrel{def}{=} \frac{\sum_{j=1+ITEMS(0)}^{J} \sum_{i=1}^{j-1} \sum_{k=1}^{ITEMS(0)} \chi\left(\chi(id_{ki}) a_{ji}\right)}{J - ITEMS(0)} \tag{8}$$

where

$$id_{ji} \stackrel{def}{=} \begin{cases} 1 & \text{, if } j = i \\ 0 & \text{, if } i \notin \bar{\mathcal{P}}_j \\ \sum\limits_{k \in \mathcal{P}_j} a_{kj} id_{ki} \left(= \sum_{k \in \mathcal{S}_i} a_{ik} id_{jk}\right) & \text{, otherwise} \end{cases} \tag{9}$$

for $j, i = 1, \ldots, J$ is the internal demand for item i that is directly or indirectly caused if one item j is produced. Here, $\mathcal{P}_j$ denotes the set of all immediate predecessors of item j, $\bar{\mathcal{P}}_j$ denotes all predecessors of item j. We do not use this definition of complexity, because we feel that the fact that the value of C' has no upper bound, which is valid for all gozinto–structures whatsoever, is less graphic than our definition. A value of C close to one indicates "many arcs" which means many predecessor relationships between items. But what does a C'–value of, say, 1.5 mean? Since we use the complexity as a user defined parameter for generating gozinto–structures, the user should have a good feeling for the impact of changing its value.

This is where the parameter $COMPLEXITY$ comes in. We now keep on adding additional arcs one by one until the gozinto–structure has the desired complexity. Figure 3 gives a possible result for the example if $COMPLEXITY = 0.5$ is chosen ($C = 0.5$, too). More precisely, we do what is described in Table 5.

while $C < COMPLEXITY$
    choose $j \in IRAND[1 + ITEMS(0), J]$
        where $\sum_{i=1}^{j-1} \chi(a_{ji}) < \sum_{n=0}^{llc_j - 1} ITEMS(n)$.
    choose $i \in IRAND[1, \sum_{n=0}^{llc_j - 1} ITEMS(n)]$ where $a_{ji} = 0$.
    $a_{ji} := IRAND[ARC_{min}, ARC_{max}]$.

Table 5: Method to Construct a Gozinto–Structure, Part 3

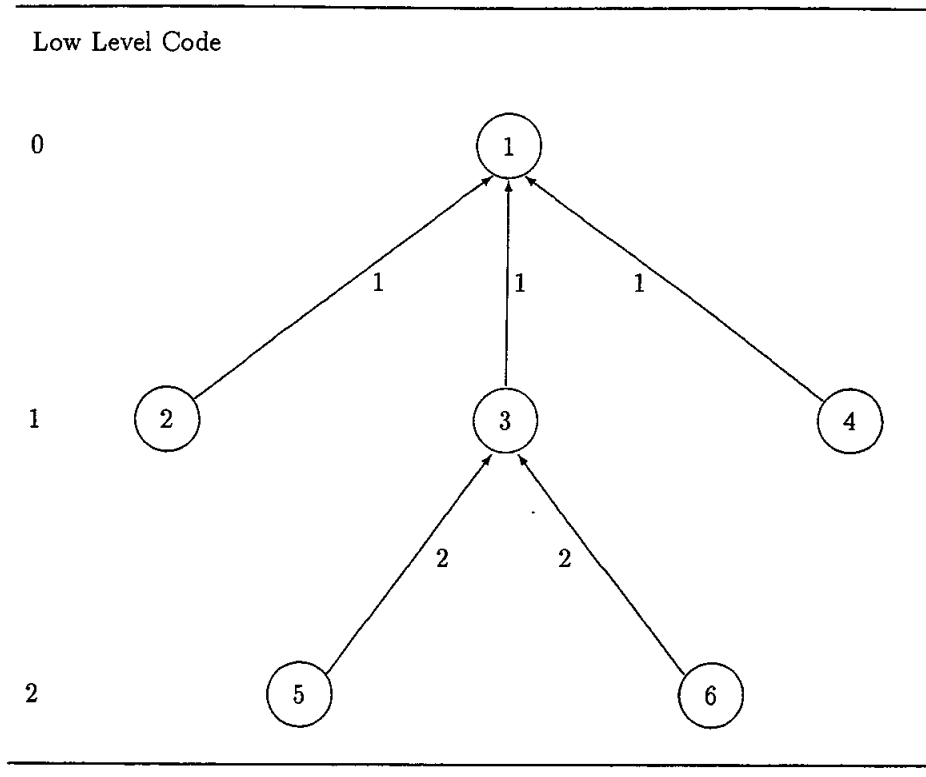

Figure 2: A Gozinto–Structure with a Minimum Number of Arcs

After this procedure terminates, we have a gozinto–structure with complexity $\mathcal{C}$ where

$$\begin{array}{c} COMPLEXITY \\ \leq \\ \mathcal{C} \\ < \\ COMPLEXITY + \frac{1}{maxarcs - minarcs} \end{array}$$

holds.

Note, linear and assembly gozinto–structures always have a complexity $\mathcal{C} = 0$. In general, divergent structures have a maximum complexity $\mathcal{C} < 1$.

Since we want to have N_G different gozinto–structures, we simply iterate the whole procedure over and over again each time starting from scratch, i.e. starting with determining $ITEMS(llc)$ for all $llc = 1, \ldots, DEPTH$. This loop is halted, if N_G different gozinto–structures are found or if $MAXITER_G$ iterations are performed. The latter condition guarantees that the generation of gozinto–structures terminates even if N_G is greater than the number of different gozinto–structures which do exist while meeting the specification. Testing whether a generated gozinto–structure is new or has already been constructed during earlier iterations seems to be an easy task at first sight. One might guess that comparing the a_{ji}–matrices is sufficient. Note, if two adjacency matrices are of a different size such a comparison is, of course, sufficient. Consider Figure 4 to see another gozinto–structure that could have been computed following the above lines. Both, Figure 3 and Figure 4 show isomorphic gozinto–structures.

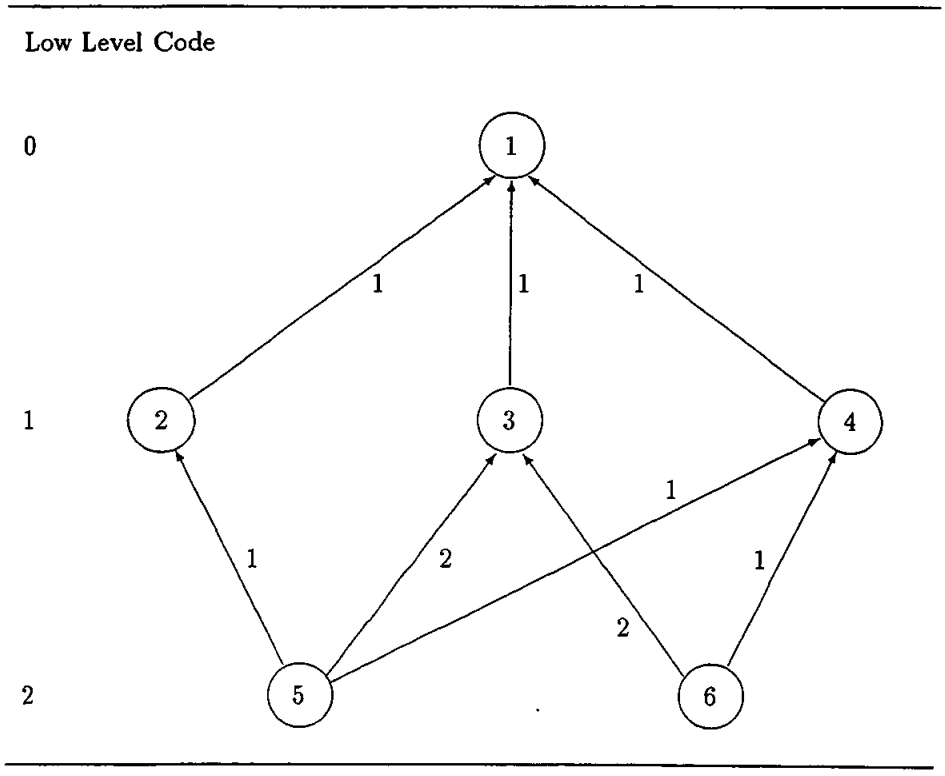

Figure 3: A Gozinto–Structure with $C = 0.5$

Since simple relabeling of the items makes both identical, we wish to identify them as the same structure.

Having a look at the a_{ji}–matrices reveals that both can hardly be identified as representing the same structure. We give the matrix for Figure 3 at the left hand side and the matrix that corresponds to Figure 4 at the right hand side:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 \end{pmatrix} \overset{?}{\Longleftrightarrow} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 1 & 0 & 0 \end{pmatrix}$$

A more sophisticated method thus needs to be employed to test whether or not two adjacency matrices represent the same gozinto–structure. If the values $ITEMS(llc)$ of the two gozinto–structures differ for at least one $llc = 0, \ldots, DEPTH$, it is obvious that the two matrices cannot represent the same gozinto–structure. Otherwise, the trick to check if two adjacency matrices represent the same gozinto–structure is to consider the corresponding incidence matrices, find pairs of so–called equivalent rows, and delete these. If this yields all rows deleted, the two incidence matrices and hence both adjacency matrices represent the same gozinto–structure. If not, they represent different structures. Before we begin to explain what equivalent rows are, let us recall the notion of

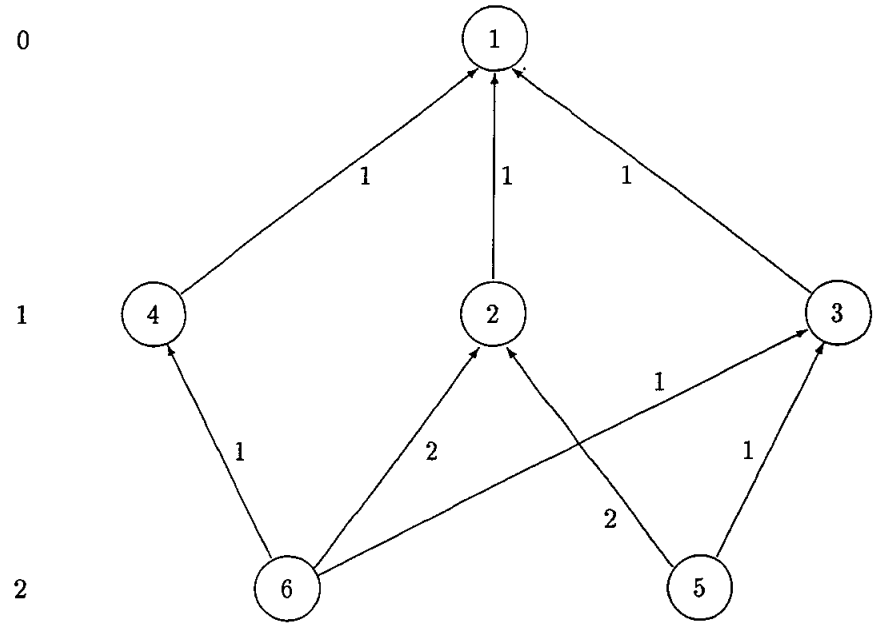

Figure 4: An Isomorphic Gozinto–Structure

an incidence matrix $(\hat{a}_{ji})_{j,i=1,\ldots,J}$ from graph theory:

$$\hat{a}_{ji} = \begin{cases} a_{ji}, & \text{, if } i \in \mathcal{S}_j \\ -a_{ij}, & \text{, if } i \in \mathcal{P}_j \end{cases} \qquad j, i = 1, \ldots, J \tag{10}$$

where $\mathcal{S}_j$ denotes the set of all immediate successors of item j.

We must use the incidence matrices, because a row j of an adjacency matrix does only show for what arcs the item with number j is the origin, but it does not provide the information for which arcs the item is a destination. It is easy to find examples which show that using the adjacency matrices instead of the incidence matrices would let the procedure to be presented indicate an isomorphism for some gozinto–structures which are not identical.

Two rows, say, j and i (which represent items j and i) are called equivalent if and only if each number that occurs, say, $n \geq 0$ times in row j, also occurs n times in row i and $llc_j = llc_i$. Note, llc_j is used to denote the low level code of item j. More formally, two rows j and i are equivalent if and only if there is a permutation π of column indices so that

$$llc_j = llc_i \text{ and } \hat{a}_{jk} = \hat{a}_{i\pi(k)} \qquad k = 1, \ldots, J \tag{11}$$

holds.

The procedure to be employed iteratively finds a row j in the one incidence matrix that is equivalent to a row i in the other and deletes both. It terminates if no further equivalent (and undeleted) rows can be found. If all rows are deleted, the two incidence matrices and so the two adjacency matrices do represent isomorphic gozinto–structures.

As an example, let us reconsider the gozinto–structures in Figures 3 and 4 again to see if the presented procedure finds out that both structures are isomorphic. The incidence matrix that corresponds to Figure 3 is given at the left hand side and the one that corresponds to Figure 4 is given on the right hand side where horizontal lines separate different low levels:

$$\left(\begin{array}{cccccc} 0 & -1 & -1 & -1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -2 & -2 \\ 1 & 0 & 0 & 0 & -1 & -1 \\ \hline 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 \end{array}\right) \overset{?}{\Longleftrightarrow} \left(\begin{array}{cccccc} 0 & -1 & -1 & -1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & -2 & -2 \\ 1 & 0 & 0 & 0 & -1 & -1 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ \hline 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 1 & 0 & 0 \end{array}\right)$$

Looking for corresponding rows, we find that row 3 of the matrix on the left hand side is equivalent to row 2 of the right hand side matrix. Hence, these rows can be deleted. So can rows 5 and 6, respectively.

$$\left(\begin{array}{cccccc} 0 & -1 & -1 & -1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & -1 & 0 \\ \times & \times & \times & \times & \times & \times \\ 1 & 0 & 0 & 0 & -1 & -1 \\ \hline \times & \times & \times & \times & \times & \times \\ 0 & 0 & 2 & 1 & 0 & 0 \end{array}\right) \overset{?}{\Longleftrightarrow} \left(\begin{array}{cccccc} 0 & -1 & -1 & -1 & 0 & 0 \\ \hline \times & \times & \times & \times & \times & \times \\ 1 & 0 & 0 & 0 & -1 & -1 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ \hline 0 & 2 & 1 & 0 & 0 & 0 \\ \times & \times & \times & \times & \times & \times \end{array}\right)$$

The reader may convince himself that moving on indeed deletes all rows and thus gives the desired result that both matrices represent isomorphic gozinto–structures.

4 Generation of External Demand

The parameters for generating external demand matrices are given in Table 6.

Symbol	Definition
$[DEM_{min}, DEM_{max}]$	Interval of demand sizes where $DEM_{max} \geq DEM_{min} \geq 0$.
$ITEMS(0)$	Number of end items.
J	Total number of items where $J \geq ITEMS(0)$.
N_D	Number of different external demand matrices to be created.
T_{idle}	Number of idle macro periods where $T_{idle} \geq 0$.
T_{macro}	Number of macro periods where $T_{macro} > T_{idle}$.
T_{micro}	Number if micro periods where $T_{micro} > 0$.
$TYPE_D$	Indicator for which items external demand may occur, i.e. $TYPE_D \in \{end\ items, all\ items\}$.

Table 6: Parameters for Generating External Demand

These parameters guide the random generation of demand matrices. While generating a d_{jt}–value for $j = 1, \ldots, J$ and $t = 1, \ldots, T$, three cases may occur:
Case 1: $t \leq T_{idle}T_{micro}$ or $\frac{t}{T_{micro}} \neq \lfloor \frac{t}{T_{micro}} \rfloor$
Case 2: $t > T_{idle}T_{micro}$ and $\frac{t}{T_{micro}} = \lfloor \frac{t}{T_{micro}} \rfloor$ and $\mathcal{S}_j \neq \emptyset$ and $TYPE_D =$ *end items*
Case 3: $t > T_{idle}T_{micro}$ and $\frac{t}{T_{micro}} = \lfloor \frac{t}{T_{micro}} \rfloor$ and ($\mathcal{S}_j = \emptyset$ or $TYPE_D =$ *all items*)

Depending on the case that holds, we define

$$d_{jt} \stackrel{def}{=} \begin{cases} 0 & Case1 \\ 0 & Case2 \\ IRAND[DEM_{min}, DEM_{max}] & Case3 \end{cases} \tag{12}$$

where $T = T_{macro}T_{micro}$. The idea behind this definition of T links model specific and real–world points of view. In the real–world external demands are not to be met at arbitrary points of time, but at some well–defined points of time such as the end of a shift, the end of a day, or the end of a week, for example. These real–world macro periods may then be subdivided into fine grain micro periods which are the subject of consideration when solving an instance. Since the user may set $T_{micro} = 1$ and hence $T = T_{macro}$ this way of generating external demand is not restrictive, but close to real–world (see also [3, 4] for a discussion of macro and micro periods when interpreting the CLSP and the DLSP). The parameter T_{idle} specifies a number of macro periods at the beginning of the planning horizon in which no external demand occurs. This allows to generate multi–level instances with low initial inventory where the depth of items must fit into the time window between period one and the due date of orders.

In total a number of N_D different external demand matrices is generated where checking if two matrices are equal can be done with a straightforward comparison.

5 Generation of Capacity Limits

To generate a matrix of capacity limits we need a gozinto–structure, an external demand matrix which defines external demand for each item in the gozinto–structure, v_j–, p_j–, and m_j–vectors, and a parameter U where $0 < U \leq 100$ which defines the percentage of capacity utilization per machine as input. A value $U > 100$ would indicate that capacity needs exceed the capacity availability. As we do not consider overtime such values are not valid.

Basically, the capacity utilization is a measure which relates capacity demand to capacity availability. Before we give more details, let us first introduce an auxiliary notation for what (external or internal) demand is to be met at what point of time. Disregarding restrictive assumptions about how many items can be produced per period, a lot–for–lot policy leads to

$$d_{jt}^{L4L} \stackrel{def}{=} \begin{cases} d_{jt} + \sum_{i \in \mathcal{S}_j} a_{ji} d_{i(t+v_j)}^{L4L} \\ \qquad \text{, if } 1 \leq t \leq T - v_j \\ d_{jt} \\ \qquad \text{, if } T - v_j < t \leq T \end{cases} \qquad \begin{matrix} j = 1, \ldots, J \\ t = 1, \ldots, T \end{matrix} \tag{13}$$

where, by the way, $d_{jt}^{L4L} = d_{jt}$ for all $j = 1, \ldots, J$ and $t = 1, \ldots, T$ in the single–level case.

Most authors define the capacity utilization of a machine m where $m = 1, \ldots, M$ as:

$$U_m \stackrel{def}{=} \frac{\sum_{j \in \mathcal{J}_m} \sum_{t=1}^{T} p_j d_{jt}^{L4L}}{\sum_{t=1}^{T} C_{mt}} 100 \tag{14}$$

A somehow more realistic definition would be

$$U_m \stackrel{def}{=} \frac{\sum_{j \in \mathcal{J}_m} p_j nr_j}{\sum_{t=1}^{T} C_{mt}} 100$$

where nr_j denotes the net requirement of item j, because in the presence of positive initial inventory the capacity utilization is overestimated, otherwise.

So, if we assume $U_1 \approx \ldots \approx U_M \approx U$ it seems to be a good idea to choose

$$C_{mt} = \left\lceil \frac{\sum_{j \in \mathcal{J}_m} \sum_{t=1}^{T} p_j d_{jt}^{L4L}}{T \cdot U} 100 \right\rceil. \tag{15}$$

But, since demand is dynamic this is not a good choice as a small single–item example shows. Suppose, $J = 1$, $M = 1$, and $T = 4$. Furthermore, assume the data given in Table 7. Let $U = 70$.

d_{jt}	$t = 1$	2	3	4	p_j
$j = 1$		35		10	1

Table 7: A Small Single–Item Example for Determining the Capacity Utilization

Using formula (15) gives $C_{1t} = 17$ for all $t = 1, \ldots, T$. Unfortunately, there is no feasible solution for an instance with these parameters, because meeting the demand in period 2 exceeds the available capacity.

The problem here is that using an overall average as defined by (15) does not take the variance of demand into account. A procedure which reflects the dynamic nature of demands is thus needed to generate a capacity limit matrix. The method that we use is given in Table 8 where

$$\bar{d}_{m\hat{t}t} \stackrel{def}{=} \frac{\sum_{j \in \mathcal{J}_m} \sum_{\tau=\hat{t}}^{t} p_j d_{j\tau}^{L4L}}{t - \hat{t} + 1} \qquad 1 \leq \hat{t} \leq t \leq T \tag{16}$$

is used to denote the average capacity demand in the interval $[\hat{t}, \ldots, t]$ of periods.

The presented method guarantees that

$$\sum_{j \in \mathcal{J}_m} \sum_{\tau=1}^{t} p_j d_{j\tau}^{L4L} \leq \frac{U}{100} \sum_{\tau=1}^{t} C_{m\tau} \qquad \begin{array}{l} m = 1, \ldots, M \\ t = 1, \ldots, T \end{array} \tag{17}$$

holds. Also,

$$\frac{\sum_{j \in \mathcal{J}_m} \sum_{t=1}^{T} d_{jt}^{L4L}}{T \cdot \frac{U}{100} + \sum_{j \in \mathcal{J}_m} \sum_{t=1}^{T} d_{jt}^{L4L}} U \leq U_m \leq U \qquad m = 1, \ldots, M \tag{18}$$

is a valid worst case bound which proves a satisfying approximation for $U_1 \approx \ldots \approx U_M \approx U$, because if

$$\sum_{j \in \mathcal{J}_m} \sum_{t=1}^{T} d_{jt}^{L4L} >> T$$

```
for m = 1 to m = M
    t := 1.
    while t ≤ T
        t̂ := t.
        d̃_m := 0.
        while d̃_m ≤ d̄_{mt̂t} and t ≤ T
            d̃_m := d̄_{mt̂t}.
            t := t + 1.
        for τ = t̂ to τ = t − 1
            C_{mτ} := ⌈d̃_m 100/U⌉.
```

Table 8: *Method to Compute Capacity Limits*

which certainly is true in most real-world situations then the left hand side evaluates to

$$\frac{\sum_{j\in\mathcal{J}_m}\sum_{t=1}^{T} d_{jt}^{L4L}}{T\cdot\frac{U}{100}+\sum_{j\in\mathcal{J}_m}\sum_{t=1}^{T} d_{jt}^{L4L}}U \approx U. \tag{19}$$

Note, if we would have chosen to compute $C_{m\tau} := \tilde{d}_m\frac{100}{U}$ in the method given in Table 8 then we would have $U_1 = \ldots = U_M = U$. However, we decided to generate integer values for demand (see (12)) as well as for capacity limits.

As an example, suppose the gozinto-structure given in Figure 5 where $J = 5$. Furthermore, assume $M = 2$ and $T = 10$. All other relevant parameters are provided in Table 9 where external demand occurs for item 1 only. Table 10 shows a protocol of running the procedure given in Table 8 with these data and $U = 70$.

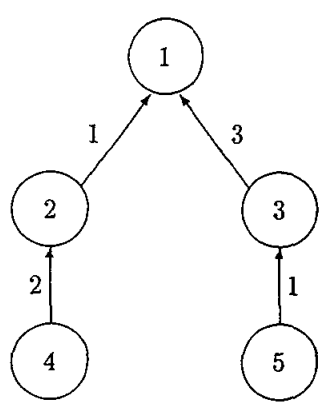

Figure 5: A Gozinto-Structure with Five Items

The entries in the columns $\tilde{d}_1$ and $\tilde{d}_2$, respectively, which are highlighted with boxes trigger the computation of $C_{1\tau}$ and $C_{2\tau}$, respectively, for $\hat{t} \le \tau \le t$. For example, since $\tilde{d}_2 = 12$ is a local maximum when computing average capacity demands beginning with period 5 stepwise up to period 9, $C_{25} = C_{26} = C_{27} =$

d_{jt}^{L4L}	$t=1$	2	3	4	5	6	7	8	9	10	m_j	p_j	v_j
$j=1$					10					20	1	1	1
$j=2$				10					20		2	1	1
$j=3$				30					60		1	1	1
$j=4$			20					40			2	1	1
$j=5$			30					60			1	1	1

Table 9: Relevant Data for the Example

$\hat{t}$	t	$\bar{d}_{1\hat{t}t}$	$\tilde{d}_1$	$\bar{d}_{2\hat{t}t}$	$\tilde{d}_2$	C_{1t}	C_{2t}
1			0		0		
	1	0	0	0	0	22	11
	2	0	0	0	0	22	11
	3	10	10	6.67	6.67	22	11
	4	15	$\boxed{15}$	7.5	$\boxed{7.5}$	22	11
	5	14		6			
5			0		0		
	5	10	$\boxed{10}$	0	0	15	18
	6	5		0	0		18
6			0				
	6	0	0			43	
	7	0	0	0	0	43	18
	8	20	20	10	10	43	18
	9	30	$\boxed{30}$	12	$\boxed{12}$	43	18
	10	28		10			
10			0		0		
	10	20	$\boxed{20}$	0	$\boxed{0}$	29	0
	11						

Table 10: A Protocol of the Method in Table 8

$C_{28} = C_{29} = \lceil 12\frac{100}{70} \rceil = 18$. Due to (14), $U_1 = 69.08$ and $U_2 = 67.16$ which is close to the desired value $U = 70$.

6 Generation of Holding and Setup Costs

The parameters for generating holding and setup costs are given in Table 11.

Holding costs are generated at random using

$$h_j \in IRAND[\ \lfloor HCOST_{min} + (J-j)\frac{HCOST_{max}-HCOST_{min}}{J} \rfloor, \quad (20)$$

$$\lceil HCOST_{min} + (J-j+1)\frac{HCOST_{max}-HCOST_{min}}{J} \rceil]$$

for $j = 1, \ldots, J$. Note, using the $RRAND$-function would be fine as well. The generation of holding costs therefore tends to assign high holding costs to items with a small low level code and low holding costs to items with a large low level code (see Subsection 3).

Symbol	Definition
$COSTRATIO$	Ratio of setup and holding costs where $COSTRATIO \geq 0$.
$[HCOST_{min}, HCOST_{max}]$	Interval of holding costs where $HCOST_{max} \geq HCOST_{min} \geq 0$.
J	Number of items.
N_C	Number of different cost vectors to be created.
$RATIODEV$	Acceptable deviation from the cost ratio where $0 \leq RATIODEV \leq 100$.

Table 11: Parameters for Generating Holding and Setup Costs

Setup costs can now be chosen with respect to

$$s_j \in IRAND[\ \lfloor COSTRATIO \cdot h_j(1 - \tfrac{RATIODEV}{100})\rfloor, \lceil COSTRATIO \cdot h_j(1 + \tfrac{RATIODEV}{100})\rceil] \tag{21}$$

for $j = 1, \ldots, J$. Again, the $RRAND$–function would be okay, too.

To find N_C different cost vectors a simple comparison helps to check for doubles.

7 Conclusion and Future Work

We have presented an instance generator for capacitated, multi–level lot sizing problems. It is trivial to remark that uncapacitated or single–level instances can also be generated by choosing the method parameters accordingly. The specifications given here have been coded in C. The source code is available on our ftp–site: ftp://ftp.wiso.uni–kiel.de/pub/operations–research

Future work should extend the instance generator for setup times, parallel machines, or sequence dependent setup costs (and times) which may be important for lot sizing and scheduling models.

Acknowledgement

This work was done with partial support from the DFG–project Dr 170/4–1.

References

[1] Barr, R.S., Golden, B.L., Kelly, J.P., Resende, M.G.C., Stewart, W.R., (1995), Designing and Reporting on Computational Experiments with Heuristic Methods, Journal of Heuristics, Vol. 1, pp. 9–32

[2] Collier, D.A., (1981), The Measurement and Operating Benefits of Component Part Commonality, Decision Sciences, Vol. 12, pp. 85–96

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

[4] FLEISCHMANN, B., (1994), The Discrete Lot–Sizing and Scheduling Problem with Sequence–Dependent Setup Costs, European Journal of Operational Research, Vol. 75, pp. 395–404

[5] HOOKER, J.N., (1995), Testing Heuristics: We Have It All Wrong, Journal of Heuristics, Vol. 1, pp. 33–42

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

[7] KOLISCH, R., SPRECHER, A., DREXL, A., (1995), Characterization and Generation of a General Class of Resource–Constrained Project Scheduling Problems, Management Science, Vol. 41, pp. 1693–1703

[8] SCHRAGE, L., (1979), A More Portable FORTRAN Random Number Generator, ACM Transactions on Mathematical Software, Vol. 5, pp. 132–138