European Journal of Operational Research 139 (2002) 277–292

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Production, Manufacturing and Logistics

Simultaneous lotsizing and scheduling on parallel machines

Herbert Meyr [*]

Lehrstuhl für Produktion und Logistik, Universität Augsburg, Universitätsstrasse 16, 86 135 Augsburg, Germany

Received 28 February 2001; accepted 2 May 2001

Abstract

This paper addresses the simultaneous lotsizing and scheduling of several products on non-identical parallel production lines (heterogeneous machines). The limited capacity of the production lines may be further reduced by sequence dependent setup times. Deterministic, dynamic demand of standard products has to be met without backlogging with the objective of minimizing sequence dependent setup, holding and production costs.

The problem is heuristically solved by combining the local search metastrategies *threshold accepting* (TA) and *simulated annealing* (SA), respectively, with dual reoptimization. Such a solution approach has already proved to be successful for the single machine case. The solution quality and computational performance of the new heuristics are tested by means of real-world problems gathered from industry. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Metaheuristics; Integer programming; Simultaneous lotsizing and scheduling; Network flows; Production

1. Introduction

When producing standard products, e.g. consumer goods, the following situation is very common: a large number of final items has to be produced on several parallel production lines, i.e. on automatic flow lines each of them consisting of many interdependent work stations. Since material flows automatically between all work stations of a production line, each production line can be considered as a single planning unit ("machine"). Deterministic dynamic demand (usually demand estimates because production orders are not available on time) is to be met without back-logging. The production lines offer – at least partially – the same services and thus can be used alternatively. However, they do not have to be technically identical. Since commonly such production lines are highly utilized, they represent potential bottlenecks.

The final items can be assigned to a few setup families. Changeovers between two items of the same family are not a problem and thus can be disregarded. Changeovers between two items of different setup families, however, may incur significant setup costs and setup times that are sequence dependent, in general.

In such a situation, decisions have to be taken about the sizes of production lots, about the assignment of these lots to single production lines, and about the line-specific sequences of the lots.

[*] Tel.: +49-821-598-4041; fax: +49-821-598-4215.
E-mail address: herbert.meyr@wiso.uni-augsburg.de (H. Meyr).

Thereby, all relevant costs – sequence dependent setup costs, inventory holding costs and variable production costs – should be minimized. The demand of each product (setup family) has to be satisfied in a timely manner and the limited capacity of each line is to be respected.

Due to the sequence dependent setup times, the capacity that is finally available for production on a specific line is known only when the size *and* the sequence of the respective lots have been determined. Therefore, the *lotsizing*, *line assignment* and *lot scheduling* have to be done *simultaneously*. Note that this planning problem relates to single stage scheduling on heterogeneous parallel machines, but is more difficult due to the lotsizing aspects. In order to emphasize the practical importance of *simultaneous* lotsizing and scheduling for business applications of the type described above, the term "production line" will be retained in the following instead of using the more general and probably better known term "machine".

Literature reviews on lotsizing in general and simultaneous lotsizing and scheduling especially are given in [8,16]. Over the last years, significant progress has been made in solving *single* line problems with (sequence dependent) setup times.

Salomon et al. [19] extend the *Discrete Lotsizing and Scheduling Problem* with sequence dependent setup costs, formerly proposed by Fleischmann [10], for *Sequence Dependent setup times* (DLSPSD). The DLSPSD is a *small time bucket model* with the special property that a respective product is either produced over a full (but rather short) *micro-period* or not at all (*all-or-nothing assumption*). Due to this time structure, setup times have to be multiples of these fixed time buckets as well. Salomon et al. solve small and medium sized problems to optimality. In [7,20], the restrictive all-or-nothing assumption is weakened. At most two products are admitted per micro-period. Both papers, however, do not present computational results.

The *General Lotsizing and Scheduling Problem with sequence dependent Setup Times* (GLSPST, [17]) is a generalization of the above models because the number of lots/products per period can unrestrictedly be varied. To achieve this, the planning horizon is divided into *large time buckets* – in the following also denoted as *macro-periods*. In [17], practical problems of the consumer goods industry can successfully be solved by novelly combining local search strategies with dual reoptimization.

Jordan and Drexl [13] present the *Batch Sequencing Problem* (BSP), a scheduling model minimizing inventory holding and setup costs. Unfortunately, *lot-splitting* is not possible, but would be necessary if demand is specified by demand estimates instead of customer orders. However, if demand is a priori split into sufficiently small buckets, an equivalence between the BSP and the DLSPSD can be shown.

Simultaneous lotsizing and scheduling of *parallel* production lines is not often discussed in the literature.

De Matta and Guignard [5,6] present two small time bucket models for non-identical production lines both being solved by Lagrangean relaxation. Only one product can be produced per micro-period. In [5], the all-or-nothing assumption is valid and setup times are formulated as a production loss during setup periods (thus being limited by the length of a micro-period). In [6], the all-or-nothing assumption is relaxed, but setup times are disregarded, too. Jordan [12] extends the BSP for identical production lines. However, only preliminary results are shown.

As an incentive for future (academic) algorithmic developments, Baker and Muckstadt [3] present the *CHES problems*, a collection of practical problems that have been gathered by Chesapeake Decision Sciences. The problem definition also includes an interesting model formulation. The CHES problems comprise parallel production lines and sequence dependent setup costs, but *no* setup times. Notably (and quite uncommonly in the literature), sales revenues are maximized.

Kang et al. [14] designed the *sequence splitting model* especially for solving the CHES problems: the entire schedule is split into a predefined number of subsequences, thus decomposing the overall planning problem into tractable subproblems. Column Generation and Branch and Bound are the basic elements of the solution heuristics of Kang et al. Detailed results of their experiments are shown in Section 4. Even though the authors affirm that setup times can be incorporated into their model, no computational results are given.

Since these heuristics for parallel production lines either comprise a rather inflexible time structure (based on small time buckets) or disregard (sequence dependent) setup times, there still remains a need for new, powerful and more general solution procedures.

In the remainder of the paper, such a solution procedure will be proposed by extending the GLSPST solution approach for parallel production lines. The GLSPST is selected because it is the broadest formulation among the single line models. As will be shown, dual reoptimization can analogously be applied if several production lines are given. However, the efficiency of such a solution method has to be investigated.

In Section 2, the GLSPST is extended by parallel production lines, thus resulting in the *General Lotsizing and Scheduling Problem for Parallel Production Lines* (GLSPPL). In Section 3, two solution procedures for the GLSPPL are presented both combining local search strategies (*threshold accepting* and *simulated annealing*) with dual reoptimization. These solution procedures prove to be quite flexible and will be broadened to meet the CHES problems [3], too.

The computational behaviour of this solution approach is tested in Section 4. With the help of this approach, practical problems of the consumer goods industry and the CHES problems are investigated.

2. Model formulation

Consider products $j = 1, \dots, J$ to be scheduled on $l = 1, \dots, L$ parallel production lines over a finite planning horizon consisting of *macro-periods* $t = 1, \dots, T$ with given length. A macro-period, for example a week or a month, is divided into a fixed number of non-overlapping *micro-periods* with variable length. Since the production lines can be scheduled independently, this is done for each line separately. S_{lt} denotes the set of micro-periods s belonging to macro-period t and production line l. All micro-periods are put in the order $s = 1, \dots, S^l$.

The length of a micro-period is a decision variable, expressed by the quantity produced in the micro-period on a respective line. A sequence of consecutive micro-periods, where the same item is produced on the same line, defines a *lot* and the quantity produced during these micro-periods defines the *size of the lot*. Therefore, a lot may continue over several micro- and macro-periods and is independent of the discrete time structure of the macro-periods. Note that micro-periods constitute both the product sequence *and* the lotsizes.

As a consequence of the fixed number S_{lt}, a lot may contain *idle micro-periods* with production quantity zero. If – after an idle micro-period – the same item is produced on the same line again, the setup state is conserved, i.e. no further setup is necessary. However, the *solution procedures* presented in this paper are able to work with a variable number of micro-periods per macro-period/line and are able to avoid idle micro-periods.

We use the following notation to formulate the problem:

Data

S_{lt}	set of micro-periods s belonging to macro-period t and line l
K_{lt}	capacity (time) of production line l available in macro-period t
a_{lj}	capacity consumption (time) needed to produce one unit of product j on line l
m_{lj}	minimum lotsize of product j (units) if produced on line l
h_j	holding costs of product j (per unit and per macro-period)
c_{lj}	production costs of product j (per unit) on line l
s_{lij}	setup costs of a changeover from product i to product j on line l
st_{lij}	setup time of a changeover from product i to product j on production line l (time)
d_{jt}	demand for product j in macro-period t (units)
I_{j0}	initial inventory of product j at the beginning of the planning horizon (units)

y_{lj0}	equals 1, if line l is set up for product j at the beginning of the planning horizon (0 otherwise)

Variables

$I_{jt} \geqslant 0$	inventory of product j at the end of macro-period t (units)
$x_{ljs} \geqslant 0$	quantity of item j produced in micro-period s on line l (units)
$y_{ljs} \in \{0,1\}$	setup state: $y_{ljs} = 1$, if line l is set up for product j in micro-period s (0 otherwise)
$z_{lijs} \geqslant 0$	takes on 1, if a changeover from product i to product j takes place on line l at the beginning of micro-period s (0 otherwise)

We formulate the GLSPPL which is a straightforward extension of the GLSP without and with setup times [11,17]:

GLSPPL:

$$\text{minimize} \quad \sum_{t,j} h_j I_{jt} + \sum_{l,i,j,s} s_{lij} z_{lijs} + \sum_{l,j,s} c_{lj} x_{ljs} \tag{1}$$

subject to

$$I_{jt} = I_{j,t-1} + \sum_{l,s \in S_{lt}} x_{ljs} - d_{jt} \quad \forall t,j, \tag{2}$$

$$\sum_{j,s \in S_{lt}} a_{lj} x_{ljs} \leqslant K_{lt} - \sum_{i,j,s \in S_{lt}} st_{lij} z_{lijs} \quad \forall l,t, \tag{3}$$

$$x_{ljs} \leqslant \frac{K_{lt}}{a_{lj}} y_{ljs} \quad \forall l,j,s, \tag{4}$$

$$x_{ljs} \geqslant m_{lj}(y_{ljs} - y_{lj,s-1}) \quad \forall l,j,s, \tag{5}$$

$$\sum_j y_{ljs} = 1 \quad \forall l,s, \tag{6}$$

$$z_{lijs} \geqslant y_{li,s-1} + y_{ljs} - 1 \quad \forall l,i,j,s. \tag{7}$$

Inventory holding costs, sequence dependent setup costs and line specific production costs are minimized (1). Note, if production costs c_{lj} are identical for all lines ($c_{lj} \equiv c_j \; \forall l,j$), the total production costs $\sum_{j,t} c_j d_{jt}$ are irrelevant for optimization and can be disregarded.

The inventory balancing constraints (2) together with $I_{jt} \geqslant 0$ ensure that demand is met without back-logging. Limited capacity is further reduced by setup times (3). Because of (4) and (6) production can only take place if the line is set up for the respective product and one and only one setup state is defined per line and micro-period. In order to change the setup state from product i to *another* product j, a *changeover* has to be executed entailing a setup time st_{lij} and setup costs s_{lij}. Such a changeover has to be started and finished within the same macro-period. Since macro-periods are large time buckets and the setup state is conserved after idle periods, this assumption does not seem to be crucial.

Minimum lotsizes (5) are introduced in order to avoid setup changes without product changes, which could lead to a wrong evaluation of the setup costs (and setup times, respectively) in an optimal solution if the setup cost matrix (of line l) does not satisfy the triangle inequality (8):

$$s_{lik} + s_{lkj} \geqslant s_{lij} \quad \forall i,j,k = 1,\ldots,J. \tag{8}$$

This situation occurs, for example, in chemical industries where certain product sequences i,j require cleaning at the changeover in order to avoid contamination. If the cleaning can be replaced by the insertion of a "rinsing" product k, then (8) is violated.

The connection between setup state indicators and changeover indicators is established by (7). Note that only the inventory balancing constraints (2) link the parallel production lines together.

3. Solution procedures

The GLSPST comprises the special case of the GLSPPL where only a single production line is given. The GLSPST has successfully been solved by combining local search metastrategies with dual reoptimization [17]. Therefore, it seems worth checking whether the principle of dual reoptimization is applicable to the GLSPPL, too. Before doing so, we briefly summarize the solution procedures for the GLSPST in order to keep the paper self-contained. For a more detailed presentation the reader is referred to [17].

3.1. Solution procedures for the GLSPST

Since only a single production line is regarded, the index l denoting the production line(s) is skipped within this section.

3.1.1. Fixing the setup pattern by local search

A *solution* for the GLSPST is characterized by the *setup pattern* y_{js} (implying z_{ijs}) and the production quantities x_{js} that are assigned to this setup pattern. A *lot* consists of a sequence of production quantities of the same product. The *cost of a solution* is the sum of setup costs caused by the setup pattern and holding costs caused by the respective lots (remember that production costs are irrelevant in the single line case). If the setup pattern is fixed, i.e., if the *sequence* of the lots is known, the problem of determining *lotsizes* that fit to the setup pattern and cause *minimal* holding costs is a minimum cost network flow problem (MCFP).

A *neighbor* of a current solution of the GLSPST is another solution whose setup pattern is slightly changed and whose lotsizes are determined by a specific procedure that solves the new MCFP. These changes in the setup pattern may result from *insertion* of a new lot between two lots of the current solution, *deletion* of a lot of the current solution or an *exchange* of two lots of the current solution. These operations are called *neighborhood operations*.

Starting from an initial (current) solution, a *candidate* for a new neighbored solution is selected by applying one of these neighborhood operations. The neighborhood operation may be chosen randomly or in a deterministic way as described in [11], for example. The respective product(s) for insertion, deletion or exchange and the respective micro-period(s) are drawn at random. A candidate is accepted as a new current solution if its costs are lower than the costs of the current solution plus some positive constant *Th*.

Depending on the choice of *Th*, the two local search procedures *threshold accepting* (TA, [9]) and *simulated annealing* (SA, [15]) can be implemented. In the first case, *Th* is directly chosen from a sequence of decreasing threshold values. In the second case, *Th* is determined by $Th := \tau \cdot |\log(\rho)|$, where τ is the (decreasing) temperature of the annealing process and ρ is a random number drawn from a uniform distribution.

The GLSPST is NP-complete (and thus the GLSPPL, too). Therefore, finding a *feasible initial* solution to start the neighborhood search is a very difficult task. To bypass this problem an *infeasible initial* solution is used as a starting point for the local search. For that purpose, the MCFP is slightly modified, so that actually *infeasible* candidates can also be accepted: a *fictitious* macro-period 0 – without capacity constraints – is introduced. Production of a quantity x_j^0 of item j within this period is punished with a penalty cost $h_j^0 x_j^0$ which expresses the degree of infeasibility and has to be high enough to prefer feasible solutions to infeasible ones. The initial (infeasible) setup pattern is then defined by assigning the complete production for all products to the *fictitious* period 0 thus suppressing production in all *real* macro-periods $t = 1, \dots, T$.

3.1.2. Dual reoptimization

One runs into trouble with computation times if trying to solve each MCFP to optimality *by starting from scratch*. On the other hand, solving the MCFP heuristically is – for sake of solution quality – not particularly desirable, too.

Dual reoptimization bypasses these two problems, because it is able to recognize and refuse too expensive candidates very quickly, but also to evaluate the minimal holding costs of each acceptable candidate.

The above local search strategies refuse a candidate if its objective function value exceeds the objective function value of the current solution by *Th*. Without loss of generality the candidate can also be refused if the cost of the candidate is replaced by a lower bound. Since the objective function value of a feasible solution of the dual problem is a lower bound to the optimal solution of the corresponding primal problem, a *dual* network flow algorithm is used to solve the MCFP. Thereby, a sequence of increasing lower bounds to the MCFP is generated and a candidate is refused as soon as the first of these lower bounds exceeds *Th*. If none of the lower bounds exceeds *Th*, the dual network flow algorithm terminates in an optimal solution of the MCFP and the candidate has to be accepted.

For each candidate to be tested, a lot of information is available in advance since it differs from the current solution only by slight changes of the problem data. Solving the candidate's MCFP by starting from scratch would ignore this information. Therefore, time-savingly exploiting this information, the current solution is just *reoptimized* by the dual network flow algorithm.

As compared to solving the MCFP heuristically by a greedy algorithm (stepping backward in time and determining the lotsizes per macro-period in order of descending relative holding costs; cf. [11, Section 4.4; 17, Section 4.2]), the dual reoptimization algorithms yield better solution quality in shorter computation time.

3.2. Solution procedures for the GLSPPL

The solution procedures for the GLSPST essentially comprise the two tasks "fixing the setup pattern" by local search and "solving the remaining problem" (*the MCFP*) by dual reoptimization. In order to adapt the GLSPST solution procedures for the GLSPPL, these two tasks have to be thought over.

3.2.1. Fixing the setup pattern

The GLSPPL is harder to solve than the GLSPST because several production lines can satisfy a respective product's demand – maybe even more than one line producing the same product at the same time. Nevertheless, the setup patterns of the different lines can be fixed independently. For that purpose, the neighborhood operations of the local search framework in Section 3.1.1 are not applied until the affected production line has been drawn at random from a uniform distribution. Note that a candidate is refused anyway, if its setup patterns violate

$$\sum_j a_{lj} m_{lj} \left(\sum_{s \in S_{lt}, i \neq j} z_{lijs} \right) \leqslant K_{lt} - \sum_{i,j,s \in S_{lt}} st_{lij} z_{lijs} \qquad (9)$$

for any production line l or macro-period $t = 1, \ldots, T$. If (9) is violated, the (net) capacity (after subtracting already known setup times) is not high enough to satisfy the minimum lotsizes for a given setup pattern.

When the setup sequences of the lots have been changed, the computation of the corresponding lotsizes and the detailed scheduling (without sequencing) once more is left open to the second task "solving the remaining problem".

To construct an initial solution for the GLSPPL, again the whole demand is assigned to the fictitious period $t = 0$. Thereby the penalty costs are computed via $h_j^0 := h_j + \max_{l,i} \{s_{lij}\}$. (If production costs c_{lj} are significantly high, one should increase the penalty costs by $\max_l \{c_{lj}\}$.)

No further modifications concerning the local search are necessary when upgrading the GLSPST algorithms to several production lines.

3.2.2. Solving the remaining problem

The main difficulties are the identification and solution of the problem remaining after the setup patterns of the production lines have been fixed. As already mentioned, an ordinary MCFP has to be solved if there is only a single production line.

This is not valid for the GLSPPL any more as soon as several non-identical lines are given. In this case, the remaining problem can be formulated as a *Generalized Network Flow Problem* (GNFP, cf. [1,18], for example) of the following type:

GNFP:

$$\text{minimize} \quad \sum_{m,n:(m,n) \in \mathscr{A}} c_{mn} X_{mn} \qquad (10)$$

subject to:

$$\sum_{m:(m,n) \in \mathscr{A}} \mu_{mn} X_{mn} - \sum_{m:(n,m) \in \mathscr{A}} X_{nm} = 0 \quad \forall n \in \mathscr{N}, \qquad (11)$$

$$l_{mn} \leqslant X_{mn} \leqslant u_{mn} \quad \forall (m,n) \in \mathscr{A}. \qquad (12)$$

The network $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ consists of a set of nodes $\mathscr{N}$ and a set of arcs $\mathscr{A}$. The problem is the identification of a flow X_{mn} for each arc $(m,n) \in \mathscr{A}$ so that the total costs are minimized (10). Thereby, c_{mn} denotes the per unit cost of arc (m,n). The flow X_{mn} on arc (m,n) is bounded by some lower bound l_{mn} and some upper bound u_{mn} (12). For each node

$n \in \mathscr{N}$, the total inflow $\sum_{m:(m,n)\in\mathscr{A}} \mu_{mn}X_{mn}$ has to meet the total outflow $\sum_{m:(n,m)\in\mathscr{A}} X_{nm}$ (11).

Contrary to an ordinary MCFP, the network $\mathscr{G}$ is a so-called *network with losses and gains*. This is caused by the (*arc*) *multipliers* μ_{mn} decreasing the flow on arc (m,n) if $\mu_{mn} < 1$ and increasing the flow on arc (m,n) if $\mu_{mn} > 1$. If $\mu_{mn} = 1$ for all $(m,n) \in \mathscr{A}$, GNFP reduces to an ordinary MCFP again.

Fig. 1 and Table 1 show how to interpret the GNFP. In Fig. 1, an example for a network $\mathscr{G} = (\mathscr{N}, \mathscr{A})$ with $L = 2$ production lines, $J = 2$ products and $T = 2$ macro-periods (plus the additional fictitious period 0) is given.

The following nodes are used:

- $L \cdot T$ capacity nodes n^{C}_{lt} and L dummy nodes n^{Du}_{l} are mandatory for modeling the limited capacity of production line l in macro-period t.
- $J \cdot T$ demand nodes n^{D}_{jt} are – among other things – necessary to represent the demand of product j in macro-period t.
- Ending inventory may be built up due to minimum lotsizes. For this reason J ending inventory nodes n^{E}_{j} have been introduced.
- J dummy nodes n^{S}_{j} are, for example, useful in modeling the unlimited capacity of the fictitious period $t = 0$ and in assigning the penalty costs to the products j.

Depending on the arc, the flow is measured in the two different dimensions "units of product quantity" [UP] and "units of time" [UT]. The flow X_{mn} on a *production arc* $(m,n) = (n^{\mathrm{C}}_{lt}, n^{\mathrm{D}}_{jt})$, for example, is measured in the dimension [UT]. It represents the share of the (net) capacity of line l in macro-period t which is used for producing j. As soon as this flow is entering node n^{D}_{jt}, it is transformed by multiplying by $\mu_{mn} = (1/a_{lj})[UP/UT]$. The output of node n^{D}_{jt} is then measured in units of product quantity [UP].

Altogether, eight different types of arcs link the nodes of the graph $\mathscr{G}$. The respective lower bounds l_{mn}, upper bounds u_{mn}, costs c_{mn} and gains/losses μ_{mn} are shown in Table 1.

Demand is met because the lower and upper bounds of the *demand arcs* are set to d_{jt}. *Inventory arcs* carry the inventory of a product j in macro-period t over to the next macro-period $t + 1$. Ending inventory is allowed, but punished by holding costs h_j which are assigned to the *ending inventory arcs*. In order to satisfy the flow balancing constraints (11), this ending inventory flows back to the dummy nodes n^S_j utilizing the *reflux arcs*.

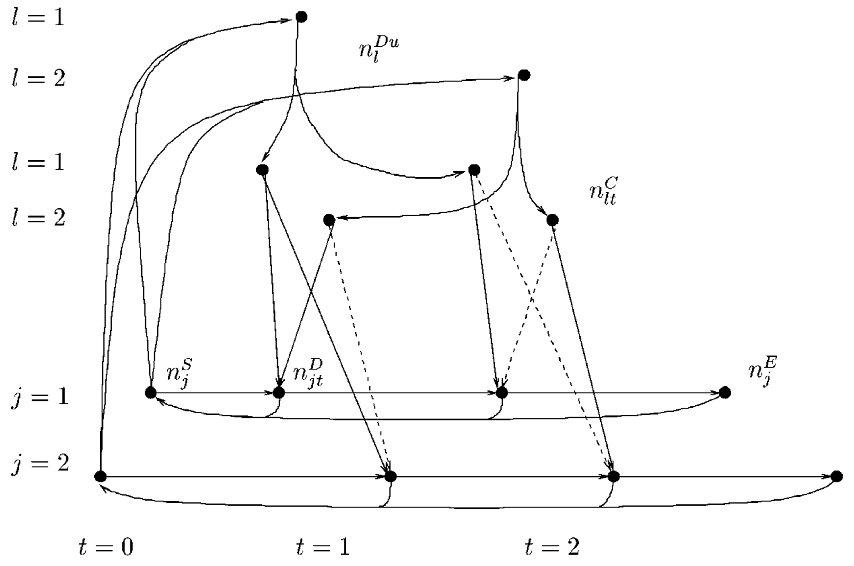

Fig. 1. Example for a graph $\mathscr{G}$ with $L = 2$, $J = 2$ and $T = 2$.

Table 1
Arcs (m, n) of the network $\mathscr{G}$ and the corresponding data: lower bounds l_{mn}, upper bounds u_{mn}, costs c_{mn} and gains/losses μ_{mn}. Additionally, the initial flow X_{mn}^{Start} (exclusive production in the fictitious period) is shown

(m, n)	l_{mn}	u_{mn}	c_{mn}	μ_{mn}	Arc type	X_{mn}^{Start}
$(n_{lt}^{\text{C}}, n_{jt}^{\text{D}})$	$m_{lj} a_{lj} \hat{z}_{ljt}$[a]	$K_{lt} \cdot \min\{1; \hat{z}_{ljt}\}$	$\frac{c_{lj}}{a_{lj}}$	$\frac{1}{a_{lj}}$	Production	0
$(n_{jt}^{\text{D}}, n_j^{\text{S}})$	d_{jt}	d_{jt}	0	1	Demand	d_{jt}
$(n_{jt}^{\text{D}}, n_{j,t+1}^{\text{D}})$ for $t = 1, \ldots, T-1$	0	∞	h_j	1	Inventory	$\sum_{\tau=t+1}^{T} d_{j\tau}$
$(n_{jT}^{\text{D}}, n_j^{\text{E}})$	0	∞	h_j	1	Ending inventory	0
$(n_j^{\text{E}}, n_j^{\text{S}})$	0	∞	0	1	Reflux	0
$(n_j^{\text{S}}, n_{j1}^{\text{D}})$	0	∞	h_j^0	1	Feasibility	$\sum_t d_{jt}$
$(n_l^{Du}, n_{lt}^{\text{C}})$	0	$K_{lt} - \sum_{i,j,s \in S_t} st_{lij} z_{lijs}$	0	1	Capacity	0
$(n_j^{\text{S}}, n_l^{\text{D}u})$	0	∞	0	a_{lj}	Linking	0

[a] $\hat{z}_{ljt} := \sum_{s \in S_{lt}, i \neq j} z_{lijs}$ (number of lots of product j starting in macro-period t).

The *feasibility arcs* model the unlimited capacity of the fictitious period $t = 0$ and establish the penalty costs h_j^0. If the flow of all *feasibility arcs* amounts to 0, a feasible solution for the GLSPPL is found. The (net) capacity that can be used for production on line l in macro-period t is given by the *capacity arcs*.

The *linking arcs* have merely been introduced to balance the total flow in the network (11). When the flow is entering a node $n_l^{\text{D}u}$, the transformation of goods that has taken place on the *capacity arcs* is compensated by the multipliers $\mu_{mn} = a_{lj}[UT/\ UP]$.

Table 1 additionally shows the flow X_{mn}^{Start} which is optimal for the initial (infeasible) solution (when production is suppressed in all "real" macro-periods $t = 1, \ldots, T$).

Please note that the above GNFP is well defined as long as (9) holds. Also note that identical production lines ($a_{lj} \equiv a_j\ \forall l, j$) imply an ordinary MCFP. This is true because a unitary flow in the network – measured in units of time – can be established after some minor data transformation. In this case, all multipliers μ_{mn} may be set to 1 and thus are no longer crucial.

A GNFP is harder, i.e. slower, to solve than an ordinary MCFP. Arc multipliers $\mu_{mn} \neq 1$ necessitate the use of less convenient data structures. As opposite to the MCFP, the mathematical operations *multiplication* and *division* are needed to solve a GNFP. Nevertheless, *generalized network flow algorithms* promise to be more efficient than solution procedures for pure Linear Programs.

In order to enable the early refuse of unacceptable candidates, a *dual* or *primal-dual* generalized network flow algorithm is needed. For that purpose the *relaxation algorithm* of Bertsekas and Tseng [4] has been implemented. So the primal flow balancing constraints (11) are relaxed, but dual feasibility is always ensured. Each iteration of the algorithm leads either to a reduction of primal infeasibility (because of a flow update) or to an increase of the dual objective function value. When primal feasibility has been achieved, both the primal and the corresponding dual GNFP are solved to optimality.

3.2.3. Accelerating the solution procedures

Section 4 will show that the generalized network flow algorithm sometimes causes undesirably high computation times. In order to accelerate the solution process, two modifications are useful:

- Solving the GNFP with the relaxation algorithm of Bertsekas and Tseng [4], some of the candidates need an excessive number of iterations. This is due to a high number of flow updates and/or a very small increase of the dual objective function value per iteration. To mitigate the influence of such candidates, a *maximum number of iterations per candidate MaxIt* is introduced. A candidate is refused regardless of its objective function value if *MaxIt* is exceeded.
- When testing a new candidate, the current solution of the GLSPPL is merely *reoptimized*. An initial lower bound to the optimal objective

function value of the candidate's GNFP can be computed very quickly by exploiting the already known optimal dual prices of the current solution's GNFP (cf. [17]). If this initial lower bound is rather high, the candidate is likely to be refused during reoptimization. Therefore, a candidate is refused if its setup costs and its initial lower bound together exceed the objective function value of the current solution. In the following, this procedure will be called (*early*) *denial of reoptimization*.

Both of these procedures refuse a candidate regardless of the local search threshold Th. This is expected to decrease the solution quality for sake of shorter computation times. The extend of this phenomenon will be investigated in Section 4.1.

Finally note that – despite of these modifications – only those candidates will be accepted whose corresponding GNFP has been solved to optimality.

3.3. Solving the CHES problems

These solution procedures are quite flexible and can easily be expanded. They can – after some minor modifications – even solve the CHES problems introduced by Baker and Muckstadt [3].

The CHES problems comprise five real world problems gathered by *Chesapeake Decision Sciences*. Baker and Muckstadt [3] formulate a mathematical model describing these problems and present a "good" solution for each of these problems.

The problem characteristics are as follows: except for the problems CHES3 (single line) and CHES4 (identical lines), non-identical production lines are given. The setup state is conserved after idle periods. There are sequence dependent setup costs, but no setup times. Except for the problem CHES1, the triangle inequalities (8) are violated.

The CHES problems coincide with the GLSPPL formulation of Section 2. In addition to the GLSPPL, the following requirements have to be met:

1. Products j are sold in the market realizing prices p^{M}_{jt} per unit of product j being sold in macro-period t. So demand d_{jt} is a lower bound on total sales of product j in macro-period t and can be exceeded.
2. If j is the last product of the schedule of production line l, a fixed ending cost c^{E}_{lj} is incurred.
3. For each lot of product j in the schedule of line l, a *maximum lotsize* max_{lj} is to be respected.
4. A lot may not extend over several macro-periods. Thus, a new setup is mandatory at the beginning of each macro-period.
5. A lot of product j may not be followed by another lot of the same product. Since maximum lotsizes are assumed, this is a serious restriction.
6. Instead of the initial setup state, a *starting operation* is given. Therefore, the *type* of the first lot of each production line is known in advance. The *size* of such a lot, however, is a decision variable, again limited by an upper and lower bound.

Minor modifications are necessary to adapt the GLSPPL solution procedures to these new requirements:

cf. (1) To model market sales, set $c_{mn} := -p^{\mathrm{M}}_{jt}$ and $u_{mn} := \infty$ for all demand arcs $(m,n) = (n^{\mathrm{D}}_{jt}, n^{\mathrm{S}}_{j})$.

cf. (2) After fixing the setup pattern of a production line l, the last product j of the corresponding line-schedule is already determined. Therefore, ending costs c^{E}_{lj} can easily be respected.

cf. (3)–(5) Maximum lotsizes are modeled by introducing upper bounds

$$u_{mn} := a_{lj} \cdot max_{lj} \cdot \sum_{s \in S_{lt}, i \neq j} z_{lijs} \tag{13}$$

for each production arc $(m,n) = (n^{\mathrm{C}}_{lt}, n^{\mathrm{D}}_{jt})$. Simultaneously, penalty costs

$$s_{ljj} := \infty \quad \forall l, j \tag{14}$$

prohibit two consecutive lots of the same product. Note that the same setting enforces a new lot to be started at the beginning of each macro-period. (Because in the GLSPPL a setup carryover has anyway been modeled by setup costs $s_{ljj} = 0$ for all l, j.)

cf. (6) To ensure that product j is the starting operation on line l, a fictitious product 0 has to be introduced. This fictitious product defines the initial setup state of line l: $y_{l00} := 1$ and $y_{li0} := 0$ for $i = 1, \dots, J$. With

$$s_{l0i} := \begin{cases} 0 & \text{if} \quad i = j, \\ \infty & \text{else,} \end{cases} \qquad (15)$$

$$\text{and } s_{li0} := \infty \text{ for } i = 1, \dots, J$$

the desired setup is finally enforced.

4. Computational results

In [17], the two solution procedures TADR and SADR to the single line problem GLSPST are presented. Both of them apply dual reoptimization and will serve as benchmark algorithms when evaluating the multi-line procedures introduced in Sections 3.2 and 3.3.

TADR employs TA to control the local search. The threshold values *Th* of TADR are taken from the decreasing sequence 0.15, 0.03, 0.025, 0.02, 0.015, 0.014, 0.013, ..., 0.002, 0.001, 0. The maximum number of candidate tests before changing the threshold value is set to 1000. The threshold is also lowered when 250 tests have not improved the current objective value. If the current solution has not changed within *TaEnd* := 3000 steps, a run of TADR is stopped.

SADR is based on SA. The temperature τ of the rth candidate test computes as

$$\tau := \alpha^{q-1} \cdot \hat{\tau} \quad \text{for } (q-1)M < r \leqslant qM. \qquad (16)$$

In this annealing schedule, the temperature is kept constant for different *plateaus* $q = 1, \dots, Q$ of the search. $Q = 18$ performed best in the parameter tests underlying [17]. The *length of plateau M* is set to 1000. Furthermore, the *initial temperature* $\hat{\tau} = 1000$ and the *cooling rate* $\alpha = 0.8$ are used.

In the following, four different solution procedures for the GLSPPL are evaluated. TAPL and TAPLS combine dual reoptimization with threshold accepting. *TAPL* tries to imitate TADR as close as possible. Therefore, the same control parameters of the local search are used. *Early denial of reoptimization* is forbidden (cf. Section 3.2.3). The *maximum number of iterations per candidate* is given by *MaxIt* = 1000, thus practically having no effect.

TAPLS should shorten computation time when compared to TAPL. As a consequence *early denial of reoptimization* is allowed and *TaEnd* and *MaxIt* are lowered to 500 and 75, respectively.

SAPL and SAPLS solve the GLSPPL by combining dual reoptimization with simulated annealing. *SAPL* employs the parameter configuration $Q = 40$, $M = 1000$, $\hat{\tau} = 50$ and $\alpha = 0.95$. *Early denial of reoptimization* is permitted, again, and *MaxIt* is set to 1000. *SAPLS* shows a better running time performance because Q and *MaxIt* are reduced ($Q = 10$, $MaxIt = 75$).

All computational tests concerning these solution procedures have been executed on a personal computer with a PentiumPro200 CPU. The operating system *Linux* and the *gcc* compiler are used in the experiments of Section 4.1 (to be comparable with [17]), whereas *Windows/NT* and *Borland/C++* are preferred in the subsequent sections.

4.1. Single line problems

In Section 3.2.3, it was already mentioned that the multi-line procedures TAPL and SAPL cause rather high computation times. Therefore, two modifications were proposed, which ought to reduce running times. On the other hand, they were expected to decrease solution quality. These statements are going to be quantified now. In order to get some basic insights, it will be sufficient to concentrate on the threshold accepting heuristics.

Some single line problems of the consumer goods industry presented in [17] serve as the data basis. Considering only a single production line, TADR and TAPL are directly comparable. Since both employ the same threshold accepting control parameters, the impacts of changing from an ordinary MCFP to a GNFP with losses and gains can be illustrated.

The data set comprise 44 practical problems with $T = 4$ macro-periods and 2–16 products and 42 problem instances with $T = 8$ macro-periods and 5–18 products. Gross utilization (including setup times) varies between 70% and 97%. Sequence independent setup times cover about 1–3% of total capacity available. Setup costs, however, are sequence dependent.

Table 2 shows the results of the computational experiments. The problems with $T = 4$ and $T = 8$ macro-periods, respectively, are pooled in four problem classes with up to $5, 10, 15, \ldots$ products. The number of problem instances within a class is denoted by #. For each problem class the average computation time (CPU seconds) of a single run of the respective solution procedure is presented. For every problem instance 250 runs of TADR and 25 runs of TAPL/TAPLS have been executed.

Additionally, the average percentage deviation of TAPL/TAPLS from TADR is shown. It is measured by

$$dev := \frac{z(TAPL/TAPLS) - z(TADR)}{z(TADR)} \cdot 100 \qquad (17)$$

with $z(\cdot)$ denoting the respective solution procedure's objective function value (averaged over all 250 or 25 runs of a problem instance). *dev* is further aggregated over all problems of a problem class.

As one would expect, there is no significant difference in solution quality between TADR and TAPL (0.3%). This is because the same local search control parameters have been used. TAPLS, however, shows a decrease in solution quality of 2.2%. The modifications presented in Section 3.2.3 indeed lead to worse results. Nevertheless, the overall impact seems not to be crucial.

On the other hand, this reduction of solution quality comes along with a decrease in computation time. Running times of TAPLS are three times less on an average than running times of TAPL.

Compared to TADR, the computation times of the multi-line procedures TAPL and TAPLS appear dramatically high. This gap increases the longer the planning horizon is and the more products are involved. When looking at TADR and TAPL one can see the only reason for these results: the MCFP that remains after fixing the setup pattern is solved by far faster applying the GLSPST solution procedure and its specialized MCFP algorithm [2]. The GNFP algorithm of Bertsekas and Tseng implemented in TAPL performs comparably only for very small problem instances.

Nevertheless, TAPLS solves these industrial single line problems in high, but still acceptable computation times. Whether this is also true for the multi-line case will be investigated in the following sections.

4.2. Benefits of a computational line assignment

The practical problems of the last section originally comprise four production lines. Two of them are identical. Thus, the respective products can be scheduled alternatively on both lines. The assignment of demand to a single line has so far been done manually by practitioners. In this section, the manual assignment is skipped and the multi-line algorithms TAPLS/SAPLS are used to schedule both lines simultaneously. By comparing

Table 2
Average computation time *CPU* (seconds) and average percentage deviation *dev* of TAPL/TAPLS from TADR for # problem instances of a problem class with *T* macro-periods and *J* products

T	*J*	#	*CPU*			*dev*	
			TADR	TAPL	TAPLS	TAPL	TAPLS
4	2–5	9	0.3	4.8	1.5	1.02	3.17
4	6–10	19	0.5	21.4	7.9	0.50	2.11
4	11–15	15	0.6	45.6	16.8	0.16	2.23
4	16	1	0.7	144.5	49.4	−0.50	0.47
8	5	2	0.6	37.9	16.7	1.68	3.20
8	6–10	10	0.9	193.2	63.6	0.50	2.50
8	11–15	26	1.2	445.1	159.0	−0.03	1.86
8	16–18	4	1.6	1017.6	287.0	−0.02	0.98
All problems		86	0.8	227.2	77.0	0.31	2.15

the emerging results with the results of the last section, the advantage of a "computational" line assignment over the "manual" assignment can be detected.

In the multi-line case, 12 problem instances $T01, \dots, T12$ with $T = 8$ macro-periods, $L = 2$ production lines and $J = 15$–19 products are considered. Each instance results from combining the respective pair of single line problems (where the manual line assignment had already taken place). Since both lines are identical, production costs are disregarded.

Twenty-five runs of TAPLS/SAPLS and 250 runs of TADR/SADR have been executed to test each of the 12 multi-line and 24 single line problems. Table 3 shows the percentage deviation *dev* of TAPLS/SAPLS from TADR/SADR for all problems $T01, \dots, T12$, separately. The average objective function value of the multi-line problem is compared to the summed objective function values of the corresponding pair of single line problems as (analogously) defined by expression (17).

The computational line assignment improves the manual one by 3–4%. Since the accelerated multi-line versions of threshold accepting and simulated annealing are used, these results are particularly notable. We have seen in the last section that the solution quality of the multi-line procedures can further be increased by tolerating higher computation times.

TAPLS and SAPLS show approximately the same solution quality. To demonstrate this, the percentage deviation of TAPLS from SAPLS is also presented in Table 3. Thus, the difference in solution quality between threshold accepting (−4.2%) and simulated annealing (−2.7%) is due to the higher quality of SADR when directly compared to TADR (cf. [17]).

Additionally, the average computation times (CPU seconds) of both multi-line heuristics are depicted in Table 3. With at most 10 minutes, running times of TAPLS appear rather high, but are again on an acceptable level. Computation time clearly increases with the number of products J increasing. The problems $T02$ and $T06$ are somewhat off-beat because in these problem instances one line is shut down during 4 of the 8 macro-periods.

Running times of SAPLS exceed running times of TAPLS by more than 50% on an average. Since both heuristics show comparable solution quality, threshold accepting outperforms simulated annealing in the two line case.

Table 3
Average percentage deviation *dev* of TAPLS from TADR ($\mathrm{TA}_{\mathrm{DR}}^{\mathrm{PLS}}$), SAPLS from SADR ($\mathrm{SA}_{\mathrm{DR}}^{\mathrm{PLS}}$) and TAPLS from SAPLS ($^{\mathrm{TA}}_{\mathrm{SA}}\mathrm{PLS}$); average computation times CPU (seconds) of a single run of TAPLS and SAPLS for the problems $T01, \dots, T12$

	J	#	*dev*			CPU	
			$\mathrm{TA}_{\mathrm{DR}}^{\mathrm{PLS}}$	$\mathrm{SA}_{\mathrm{DR}}^{\mathrm{PLS}}$	$^{\mathrm{TA}}_{\mathrm{SA}}\mathrm{PLS}$	TAPLS	SAPLS
$T01$	15	1	−6.79	−5.45	−0.48	268	471
$T02$[a]	16	1	−0.36	1.71	0.08	(163)	(225)
$T03$	16	1	−3.57	−0.62	−1.59	289	560
$T04$	17	1	1.43	3.50	−0.98	326	710
$T05$	17	1	−2.63	0.81	−0.53	369	589
$T06$[a]	18	1	−6.02	−8.32	3.58	(349)	(376)
$T07$	18	1	−9.24	−6.20	−2.13	422	616
$T08$	19	1	−10.14	−9.31	0.59	465	751
$T09$	19	1	0.50	2.29	0.06	489	860
$T10$	19	1	−5.91	−4.49	−1.39	512	863
$T11$	19	1	−11.53	−10.60	0.47	548	868
$T12$	19	1	3.90	4.20	2.42	598	696
All problems		12	−4.20	−2.71	0.01	400	632

[a] One production line is shut down during 4 of the 8 macro-periods.

4.3. The CHES problems

It is desirable to compare the GLSPPL heuristics to other solution procedures tackling the same or at least a related problem. To make the latter one possible, the GLSPPL heuristics have been expanded in Section 3.3 so that the CHES problems can be solved, too. For each of the CHES instances the "good" solution of Baker and Muckstadt [3] is already known. The same problems have also (heuristically) been solved by Kang et al. [14]. These authors additionally provide lower bounds.

The five CHES instances have already been described in Section 3.3. For each instance the number of macro-periods T, products J and production lines L is shown in Table 4. To test the local search procedures, 400 runs of TAPL and SAPL have been executed per problem instance. If a changeover from product i to product j is forbidden on a production line l, penalty costs $s_{lij} = 5000$ substitute for $s_{lij} = \infty$.

Within Sections 4.1 and 4.2, the number of micro-periods per macro-period $|S_{lt}|$ did not have to be restricted. In [14] however, the number of lots per macro-period is bounded by the number of split-sequences `Lt` times the maximum number of products per split-sequence `maxBr`. Therefore, in all TAPL and SAPL experiments of this section $|S_{lt}|$ was also bounded by J.

For each problem instance, the lower bound and the *best* solution found by Baker and Muckstadt ([3], Ches), Kang et al. ([14], Kang), TAPL and SAPL are shown in Table 4. The best TAPL and SAPL solutions result from the above mentioned 400 runs of the respective local search procedure, the best Kang solution has been determined by a systematic variation of `Lt` and `maxBr`.

Note that negative costs are due to market sales. The benchmark solutions of Baker and Muckstadt can be improved by all other solution procedures. Among these, SAPL shows the best results. It is able to achieve or improve the Kang solutions in all five problem instances. Additionally, the percentage deviation *dev* of these best solutions from the lower bound is depicted in Table 5. Except for the Ches solutions, this gap is less than 5%, throughout.

The computation times (CPU seconds) of the best TAPL and SAPL solutions are also shown in Table 5. Note that the computational study of Kang et al. was done on a Pentium 75 personal

Table 4
Lower bound and best solution found by Baker and Muckstadt (Ches), Kang et al. (Kang), TAPL and SAPL

	$T/J/L$	Lower bound	Best objective function value			
			Ches	Kang	TAPL	SAPL
CHES1	1/10/10	121.84	130.1	121.84	121.84	121.84
CHES2	1/21/8	−2860.7	−2814.8	−2825.54	−2821.75	−2825.54
CHES3	3/11/1	−1289485.2	−1244855.3	−1285204.26	−1285603.95	−1287037.80
CHES4	1/11/2	−646898.7	−627610.5	−646857.85	−646849.19	−646857.85
CHES5	3/12/2	−7413.0	−6829.8	−7102.32	−7098.11	−7104.11

Table 5
Percentage deviation of the best solution of Baker and Muckstadt (Ches), Kang et al. (Kang), TAPL and SAPL from the lower bound and the respective computation time (seconds) for the problems CHES1, ..., CHES5

	Percentage deviation from lower bound				Computation time (seconds)	
	Ches	Kang	TAPL	SAPL	TAPL	SAPL
CHES1	6.78	0.00	0.00	0.00	26	75
CHES2	1.60	1.23	1.36	1.23	19	76
CHES3	3.46	0.33	0.30	0.19	28	30
CHES4	2.98	0.01	0.01	0.01	< 1	2
CHES5	7.87	4.19	4.25	4.17	23	74

computer using the operating system *Windows95* and the Microsoft C/C++ compiler. Therefore, running times are not comparable.

When comparing TAPL to SAPL, there is a price to pay for the better solution quality of SAPL: the computation times of the simulated annealing procedure clearly increase.

Beyond the CHES problems, Kang et al. [14] investigate nine further artificial problems *Da*, $\ldots, Di$ with $J = 6$ products and $T = 9$ macro-periods. These problems are related to CHES5 and differ in the number of production lines L (a single line and two identical lines, respectively), the utilization U, the minimum lotsize m_{lj} and the maximum lotsize max_{lj} (cf. Table 6).

In Table 7, again, the objective function values of the best solutions and the respective computation times (of these single runs) are depicted. Since lower bounds are not available this time, the percentage deviation *dev* of the best TAPL/SAPL solution from the best solution of Kang et al. is shown.

There is a significant difference in solution quality between the TAPL/SAPL solutions and the solutions of Kang et al. to be noticed now. This is probably due to the fact that – as opposite to the CHES problems – solution quality was not a prime concern of Kang et al. when testing these problems (private correspondence). They mainly wanted to obtain insights on the problem and their algorithm for varying test scenarios.

Regarding the problem characteristics, a confirmation of the findings of Kang et al. can be made: Kang et al. expected better solutions to be obtained on two production lines than on one line. However, they were only able to prove this statement for a machine utilization of 95% and 70%, respectively, but not for a utilization of 99%. When looking at TAPL and SAPL for the problems *Db* and *Dh*, one can see that Kang et al. found a good solution for *Db*, but a rather bad solution for *Dh*. So their guess is generally true – no matter what utilization is given.

Table 6
Number of production lines L, utilization U, minimal lotsizes m_{lj} and maximal lotsizes max_{lj} for the problems $Da, \ldots, Di$

	Da	*Db*	*Dc*	*Dd*	*De*	*Df*	*Dg*	*Dh*	*Di*
L	1	1	1	1	1	1	2	2	2
U	95	99	70	95	99	95	95	99	70
m_{lj}	20	20	20	20	20	0	20	20	20
max_{lj}	200	200	200	100	100	1000	200	200	200

Table 7
Best objective function value and the respective computation time (seconds) of Kang et al. (Kang), TAPL and SAPL; percentage deviation *dev* of best TAPL/SAPL solution from the best solution of Kang et al. for the problems $Da, \ldots, Di$

	Best objective function value			CPU (seconds)		*dev*	
	Kang	TAPL	SAPL	TAPL	SAPL	TAPL	SAPL
Da	856.81	846.97	844.62	25	128	−1.2	−1.4
Db	865.29	859.97	869.30	39	158	−0.6	0.5
Dc	816.61	766.58	760.08	12	73	−6.1	−6.9
Dd	1263.01	1182.11	1174.01	31	138	−6.4	−7.1
De	1360.87	1248.26	1260.33	39	163	−8.3	−7.4
Df	832.95	812.66	812.66	26	188	−2.4	−2.4
Dg	776.80	726.00	710.43	60	289	−6.5	−8.5
Dh	940.49	841.05	859.00	64	295	−10.6	−8.7
Di	594.11	592.90	582.88	42	224	−0.2	−1.9

5. Summary and conclusions

The GLSPPL, a model for simultaneous lotsizing and scheduling of several products on non-identical parallel production lines, has been introduced. In this model deterministic dynamic demand is to be met without back-logging with the objective of minimizing production costs, inventory holding costs and sequence dependent setup costs. Sequence dependent setup times may further reduce the limited capacity of each production line.

The problem is solved by combining the local search procedures threshold accepting and simulated annealing with dual reoptimization. A large number of setup sequences is generated and tested by means of local search. For each candidate sequence being determined and fixed this way, a generalized network flow problem has to be tackled by dual reoptimization in order to evaluate the respective production and inventory holding costs. For that purpose, the relax algorithm of Bertsekas and Tseng [4] has been implemented.

If only a single production line is given, an ordinary minimum cost network flow problem (MCFP) remains after fixing the setup sequence [17]. As Section 4.1 has shown, this MCFP can be solved by far faster when using specialized (dual) network flow algorithms. Since the running times of the standard GLSPPL solution procedures are quite high, some modifications for accelerating the computation time have been proposed. Unfortunately, these modifications also lead to a (minor) decrease of solution quality.

Nevertheless, the GLSPPL solution procedures have proven to solve practical problems of consumer goods industry successfully. The computational assignment of demand to production lines improves the manual assignment (suggested by practitioners) noteworthy.

The GLSPPL solution procedures are very flexible. They have easily been adapted to deal with the CHES problems, a related type of lotsizing and scheduling problems with zero setup times. Also their solution quality is competitive to other (specialized) solution procedures. This has been shown when comparing the GLSPPL solutions with the solutions suggested by Baker and Muckstadt [3] and Kang et al. [14], respectively. Note that the algorithm of Kang et al. is especially designed to solve the CHES problems.

Altogether, the GLSPPL procedures are practically applicable and profitable, but their computational behaviour is not really satisfying. Therefore, reducing running times should be a major concern of future research.

Acknowledgements

The author is grateful to Prof. Dr. Bernhard Fleischmann, Prof. Sungmin Kang, Prof. David L. Woodruff and the two unknown referees for their helpful comments.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] A.I. Ali, R. Padman, H. Thiagarajan, Dual algorithms for pure network problems, Operations Research 37 (1) (1989) 159–171.

[3] T. Baker, J.A. Muckstadt, Jr., The CHES problems, Technical Paper, Chesapeake Decision Sciences, Inc., 200 South Street, New Providence, NJ, 1989.

[4] D.P. Bertsekas, P. Tseng, Relaxation methods for minimum cost ordinary and generalized network flow problems, Operations Research 36 (1) (1988) 93–114.

[5] R. De Matta, M. Guignard, Studying the effects of production loss due to setup in dynamic production scheduling, European Journal of Operational Research 72 (1994) 62–73.

[6] R. De Matta, M. Guignard, The performance of rolling production schedules in a process industry, IIE Transactions 27 (1995) 564–573.

[7] A. Drexl, K. Haase, Proportional lotsizing and scheduling, International Journal of Production Economics 40 (1995) 73–87.

[8] A. Drexl, A. Kimms, Lot sizing and scheduling – survey and extensions, European Journal of Operational Research 99 (1997) 221–235.

[9] G. Dueck, T. Scheuer, Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing, Journal of Computational Physics 90 (1990) 161–175.

[10] B. Fleischmann, The discrete lot-sizing and scheduling problem with sequence-dependent setup costs, European Journal of Operational Research 75 (1994) 395–404.

[11] B. Fleischmann, H. Meyr, The general lotsizing and scheduling problem, OR Spektrum 19 (1) (1997) 11–21.

[12] C. Jordan, in: Batching and Scheduling, Lecture Notes in Economics and Mathematical Systems, vol. 437, Springer, Berlin, 1996.

[13] C. Jordan, A. Drexl, Discrete lotsizing and scheduling by batch sequencing, Management Science 44 (5) (1998) 698–713.

[14] S. Kang, K. Malik, L. Thomas, Lotsizing and scheduling on parallel machines with sequence-dependent setup costs, Management Science 45 (2) (1999) 273–289.

[15] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[16] R. Kuik, M. Salomon, L.N. Van Wassenhove, Batching decisions: Structure and models, European Journal of Operational Research 75 (1994) 243–263.

[17] H. Meyr, Simultaneous lotsizing and scheduling by combining local search with dual reoptimization, European Journal of Operational Research 120 (2) (2000) 311–326.

[18] K.G. Murty, Network Programming, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[19] M. Salomon, L. Solomon, L.N. Van Wassenhove, J. Dumas, S. Dauzère-Pérès, Solving the discrete lotsizing and scheduling problem with sequence dependent set-up costs and set-up times using the Traveling Salesman Problem with time windows, European Journal of Operational Research 100 (1997) 494–513.

[20] L.A. Wolsey, MIP modelling of changeovers in production planning and scheduling problems, European Journal of Operational Research 99 (1997) 154–165.