ELSEVIER

European Journal of Operational Research 120 (2000) 311–326

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/orms

Simultaneous lotsizing and scheduling by combining local search with dual reoptimization

H. Meyr *

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

Received 1 October 1997; accepted 1 October 1998

Abstract

The contribution of this paper is twofold. On the one hand, the particular problem of integrating lotsizing and scheduling of several products on a single, capacitated production line is modelled and solved, taking into account sequence-dependent setup times. Thereby, continuous lotsizes, meeting deterministic dynamic demands, are to be determined and scheduled with the objective of minimizing inventory holding costs and sequence-dependent setup costs. On the other hand, a new general algorithmic approach is presented: A dual reoptimization algorithm is combined with a local search heuristic for solving a mixed integer programming problem. This idea is applied to the above lotsizing and scheduling problem by embedding a dual network flow algorithm into threshold accepting and simulated annealing, respectively. Computational tests show the effectiveness of the new solution method.

Keywords: Lotsizing; Scheduling; Integer programming; Heuristics; Simulated annealing

1. Introduction

In a flow shop environment – which is e.g. typical for the food or (semi-)process industry – a large number of products have to be scheduled on one (or a few parallel) highly utilized production line(s). In contrast to a job shop production design all items take the same route. Therefore, each line may be planned as a single unit.

Changeovers between items sharing the same line often cause significant, *sequence-dependent* setup times and setup costs. In order to avoid unnecessary changeovers, customer demand has to be pooled in production orders (lots). When sequence-dependent setup times are predominant, the capacity available for production depends on both the sequence *and* the size of the lots. In such a situation, lotsizing and scheduling have to be applied simultaneously in a single step of planning [11].

The problem discussed here is of this type: Continuous lotsizes of several products are determined and scheduled on a single machine (production line) with the objective of minimizing holding and sequence-dependent setup costs.

* E-mail: Herbert.Meyr@Wiso.Uni-Augsburg.DE

0377-2217/00/$ – see front matter
PII: S0377-2217(99)00159-9

Deterministic, dynamic demand, given over a finite planning horizon, is to be met without back-logging. The limited availability of the machine may be further reduced by changeovers causing sequence-dependent setup times.

Reviews of lotsizing models in general are given in Refs. [3,27,13] of scheduling models that deal with batching and lotsizing issues in Ref. [30]. In the recent past, more and more attention has been paid to *simultaneous lotsizing and scheduling with setup times*, but solution methods are still not satisfying.

In Refs. [8,9] and [31,10], respectively, models with sequence-independent and sequence-dependent setup times are formulated sharing the common property that a product is either produced over a full (but rather short) *micro-period* or not at all (*all-or-nothing assumption*). The models presented in Refs. [9,31] are based on the *Discrete Lotsizing and Scheduling Problem* (DLSP) originally formulated by Fleischmann [17,18]. Jordan [24] formulates the *Batch Sequencing Problem*, a class of scheduling problems minimizing holding and setup costs, and shows that some specifications of them are equivalent to the problems in Refs. [18,9,31]. Drexl and Haase [22,12] extend the *Proportional Lotsizing and Scheduling Problem* (PLSP) – a model that weakens the all-or-nothing assumption by admitting at most two products per micro-period – to sequence-independent setup times and outline a solution procedure. However, no computational results are given.

The papers [32,35] present some special formulations that are designed to solve small problem instances to optimality. In Refs. [25,21] models and solution procedures are proposed that are motivated by practical problems. The so-called CHES problems, a collection of practical problems gathered by Chesapeake Decision Sciences (no setup times, [4]), are dealt with in Ref. [25]. Gopalakrishnan et al. [21] tackle a lotsizing problem in the napkin production, where it is sufficient to determine the sequence only for the first and last product of each (rather long) *macro-period*.

In Section 2, we extend the *General Lotsizing and Scheduling Problem* (GLSP) of Fleischmann and Meyr [19] to deal with sequence-dependent setup times (GLSPST). The GLSP is more general than the DLSP and PLSP, because the number of products per (macro-)period is not restrictive any more.

In Section 3, two solution procedures to the GLSPST are presented which are based on the local search heuristics *threshold accepting* (TA) and *simulated annealing* (SA). In each candidate test a new setup sequence is generated. After fixing the setup sequence a minimum cost network flow problem (MCFP) has to be solved in order to determine the lotsizes and holding costs of the candidate.

This technique of tackling a mixed integer programming problem (MIP) is not new. Kuik et al. [28] consider the *Multi-Level Capacitated Lotsizing Problem*. They fix the setup state (not the sequence as it has to be done in simultaneous lotsizing and scheduling) using SA and solve the remaining linear problem (LP) *heuristically* by modifying the greedy algorithm of McClain et al. [29]. The exact solution of the LP is disregarded because of computation time limitations. Fleischmann and Meyr [19] use a similar approach for the GLSP without setup times. Teghem et al. [34] deal with the grouping of book covers on offset plates. They fix the binary variables of their model with SA and solve the remaining LP *to optimality*. However, they complain about exhaustive computation times, too, since each candidate LP is solved from scratch using a standard (external) LP solver.

These ways of tackling the LP subproblems – heuristically or optimally from scratch – both suffer from the trade-off between computation time and solution quality: Heuristic solution of the LP may be done quite fast, but the quality of solution is expected to be superior in case of optimally solved LP subproblems. However, if that has to be done from scratch, computation time tends to become restrictive soon.

We present a new approach to obtain the quality of optimally solved subproblems substantially faster than starting from scratch. Here, *reoptimization* profits from information which is provided by already tested candidates. Furthermore, we utilize a *dual* algorithm for reoptimization. So, a sequence of increasing lower bounds to

the minimum cost of every LP subproblem is computed. These lower bounds can be used to refuse an unacceptable candidate very early – without having to solve the LP subproblem at all.

This principle is applied to the GLSPST by embedding a dual network flow algorithm into TA and SA, respectively.

The computational tests of Section 4 demonstrate both the usefulness of the dual reoptimization approach and the availability of an effective, "high-quality" solution procedure for the GLSPST.

2. Model formulation

Products $j = 1, \ldots, J$ are scheduled over a finite planning horizon consisting of *macro-periods* $t = 1, \ldots, T$ with given length. A macro-period t is divided into a fixed number of non-overlapping *micro-periods* with variable length. S_t denotes the set of micro-periods s assigned to macro-period t. All micro-periods are sequenced in the order $s = 1, \ldots, S$. The number of micro-periods $|S_t|$ within a macro-period t has to be fixed in advance to allow MIP-modeling.

The length of a micro-period is a decision variable, expressed by the quantity produced in the micro-period. A sequence of consecutive micro-periods where the same item is produced 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 $|S_t|$, a lot may contain *idle micro-periods* with production quantity zero. If – after an idle micro-period – the same item is produced 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 and to avoid idle micro-periods.

The following data and variables are used:

Data:

S_t	set of micro-periods s belonging to macro-period t
K_t	capacity (time) available in macro-period t
a_j	capacity consumption (time) needed to produce one unit of product j
m_j	minimum lotsize of product j (units)
h_j	holding costs of product j (per unit and per macro-period)
s_{ij}	setup costs of a changeover from product i to product j
st_{ij}	setup time of a changeover from product i to product j (time)
d_{jt}	demand of product j in macro-period t (units)
I_{j0}	initial inventory of product j at the beginning of the planning horizon (units)
y_{j0}	equals 1, if the machine 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)
$x_{js} \geqslant 0$	quantity of item j produced in micro-period s (units)
$y_{js} \in \{0, 1\}$	setup state: $y_{js} = 1$, if the machine is set up for product j in micro-period s (0 otherwise)
$z_{ijs} \geqslant 0$	takes on 1, if a changeover from product i to product j takes place at the beginning of micro-period s (0 otherwise)

We formulate the GLSPST which is a straightforward extension of the GLSP without setup times [19]:

$$\text{minimize} \sum_{j,t} h_j I_{jt} + \sum_{i,j,s} s_{ij} z_{ijs} \tag{1}$$

subject to

$$I_{jt} = I_{j,t-1} + \sum_{s \in S_t} x_{js} - d_{jt} \quad \forall t, j, \tag{2}$$

$$\sum_{j,s \in S_t} a_j x_{js} + \sum_{i,j,s \in S_t} st_{ij} z_{ijs} \leqslant K_t \quad \forall t, \tag{3}$$

$$x_{js} \leqslant \frac{K_t}{a_j} y_{js} \quad \forall s, j, \tag{4}$$

$$x_{js} \geqslant m_j \left(y_{js} - y_{j,s-1}\right) \quad \forall s, j, \tag{5}$$

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

$$z_{ijs} \geqslant y_{i,s-1} + y_{js} - 1 \quad \forall s, i, j. \tag{7}$$

Inventory holding and sequence-dependent setup costs are minimized (Eq. (1)). The inventory balancing constraints (Eq. (2)) together with $I_{jt} \geqslant 0$ ensure that demand is met without backlogging. Limited capacity is further reduced by setup times (Eq. (3)). Because of Eqs. (4) and (6) production can only take place if the machine is set up for the respective product and one and only one setup state is defined in each 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_{ij} and setup costs s_{ij}. Such a changeover has to be started and finished within the same macro-period. Since macro-periods are large-time buckets (weeks or months, for example) and the setup state is conserved after idle periods, this assumption does not seem to be crucial.

Minimum lotsizes (Eq. (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 time, respectively) in an optimal solution if the setup cost matrix does not satisfy the triangle inequality (8):

$$s_{ik} + s_{kj} \geqslant s_{ij} \quad \forall i, j, k = 1, \ldots, J. \tag{8}$$

This situation occurs e.g. 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 Eq. (8) is violated. The minimum lotsize is based on technical requirements. For example, in continuous chemical production sometimes the "low quality material" of the starting phase of a lot is mixed with the "high quality material" of latter phases. In such a situation the lotsize is bounded by the amount which is necessary to ensure the desired minimum quality level of the mix as a whole. However, if the triangle inequality (8) holds, in many practical applications the minimum lotsizes may be set to zero and, thus, do not have any impact on economical lotsizes.

The connection between setup state indicators and changeover indicators is established by Eq. (7).

3. Solution procedures

3.1. Threshold accepting

Fixing the setup pattern: In the following, we briefly outline the TA framework for solving the GLSPST. A *solution* to 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. If the setup pattern is fixed, the problem of determining lotsizes that fit to the setup pattern and cause minimal holding costs is an 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 – either heuristically or to optimality. 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 Ref. [19], 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. One way to overcome local optima is to accept a candidate solution even if its cost does not

exceed the cost of the current solution by a specific *threshold*. That is the reason why such a procedure is called *threshold accepting* [15]. Successive reduction of the threshold leads to convergence of the algorithm. In order to be independent of the objective function level of a specific problem instance, we determine the threshold as a (decreasing) percentage value *Th* of the current solution.

Since GLSP is NP-complete [19] and GLSP is a specialization of GLSPST, the General Lotsizing and Scheduling Problem with Setup Times is NP-complete, too. Therefore, finding a *feasible initial* solution to start the neighborhood search is a very difficult task. To bypass this problem we start from an *infeasible initial* solution and let TA find the first feasible solution. 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. (As proposed in Ref. [19], we implement penalty costs $h_j^0 = h_j \max_i\{s_{ij}\}$ for all j.) An 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 $t = 1, \ldots, T$.

The MCFP with fixed setup pattern: In order to evaluate the minimal holding costs attainable for the already fixed setup pattern of a new candidate r, a min cost flow problem P^r of the following form has to be solved to optimality on the network $\mathscr{G}^r = (\mathscr{N}, \mathscr{A}^r)$:

$$\text{minimize} \sum_{(k,l)\in\mathscr{A}^r} c_{kl} X_{kl}^r$$

subject to

$$\sum_{(k,l)\in\mathscr{A}^r} X_{kl}^r - \sum_{(l,k)\in\mathscr{A}^r} X_{kl}^r = b_k^r \quad \forall k \in \mathscr{N},$$

$$l_{kl}^r \leqslant X_{kl}^r \leqslant u_{kl}^r \quad \forall (k,l) \in \mathscr{A}^r.$$

The set of nodes $\mathscr{N}$ consists of the following (cf. Fig. 1):

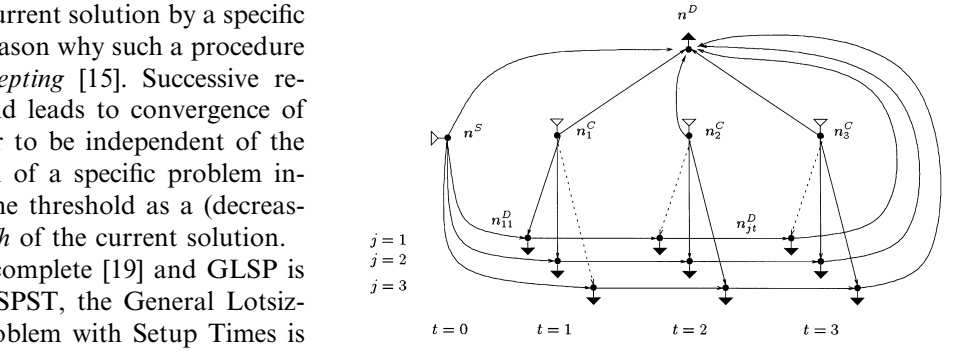

Fig. 1. Example for a graph $\mathscr{G}$ with $J = 3$ and $T = 3$.

- T capacity nodes n_t^C with supply $b_{n_t^C}^r := K_t - \sum_{i,j,s\in S_t} st_{ij} z_{ijs}^r$ representing the capacity of macro-period t; thereby, the totally available capacity K_t is reduced by the already known setup times of candidate r.
- JT demand nodes n_{jt}^D with demand $b_{n_{jt}^D}^r := -(a_j d_{jt})$ representing demand of product j in macro-period t.
- 1 dummy supply node n^S with supply $b_{n^S}^r := \sum_{j,t} a_j d_{jt}$ representing the additional (actually unlimited) capacity of the fictitious period 0.
- 1 dummy demand node n^D with demand $b_{n^D}^r := -\sum_t (K_t - \sum_{i,j,s\in S_t} st_{ij} z_{ijs}^r)$; this node is needed to balance supply and demand in the network so that $\sum_{k\in\mathscr{N}} b_k^r = 0$ holds.

A directed arc (k,l) from tail k to head l connects node $k \in \mathscr{N}$ with node $l \in \mathscr{N}$. $\mathscr{A}^r$, the set of all arcs of a new candidate r, consists of the following:

- T capacity arcs (n_t^C, n^D) representing unused capacity in macro-period t.
- At most JT production arcs (n_t^C, n_{jt}^D) representing production of product j in macro-period t; $(n_t^C, n_{jt}^D) \notin \mathscr{A}^r$ if the setup pattern of candidate r forbids production of product j in macro-period t, i.e. $\sum_{i,s\in S_t} z_{ijs}^r = 0$ (dashed arcs in Fig. 1).
- $J(T-1)$ inventory arcs $(n_{jt}^D, n_{j,t+1}^D)$ $(t = 1, \ldots, T-1)$.
- J ending inventory arcs (n_{jT}^D, n^D).
- J fictitious arcs $(n^S, n_{j,1}^D)$ representing production in the fictitious period 0.
- 1 fictitious arc (n^S, n^D) representing unused fictitious supply in period 0.

The variable X_{kl}^r denotes the flow on arc $(k,l) \in \mathscr{A}^r$, measured in units of time. The flow on

arc (k,l) is rated by the cost c_{kl} and bounded by a lower bound l^r_{kl} and an upper bound u^r_{kl} where

$$c_{kl} := \frac{1}{a_j} \cdot \begin{cases} h^0_j & \text{if } (k,l) = (n^S, n^D_{j,1}) \quad \text{(penalty costs)}, \\ h_j & \text{if } (k,l) = (n^D_{jt}, n^D_{j,t+1}) \quad (t = 1,\ldots,T-1), \\ h_j & \text{if } (k,l) = (n^D_{jT}, n^D), \\ 0 & \text{otherwise}, \end{cases}$$

$$l^r_{kl} := \begin{cases} a_j m_j \sum\limits_{i \neq j, s \in S_t} z^r_{ijs} & \text{if} \quad (k,l) = (n^C_t, n^D_{jt}) \in \mathscr{A}^r \\ & \text{(minimum lotsize)}, \\ 0 & \text{otherwise}, \end{cases}$$

$u^r_{kl} :=$ maximal possible flow on arc(k,l)

$$\text{(e.g.} \quad u^r_{kl} := K_t \quad \text{if } (k,l) = (n^C_t, n^D)\text{.}$$

If no production takes place in the fictitious period ($\sum_j X^r_{(n^S, n^D_{j,1})} = 0$), candidate r is a feasible solution to the GLSPST.

The general idea: As already mentioned the idea of fixing the binary variables of an MIP by local search and solving the remaining LP subproblem is quite common in the literature. Because of the large number of candidates to be tested Kuik et al. [28] and Fleischmann and Meyr [19] suppose to solve the subproblem *heuristically* employing fast specialized algorithms. A slight improvement of solution quality seems achievable if at least the best accepted candidate is solved to optimality ex post. Without doubt still better results would be possible if the LP subproblems of all (acceptable) candidates were *optimally* solved. However, Teghem et al. [34] demonstrate that computation times become prohibitive soon, if each LP subproblem is solved *individually, by starting from scratch*.

In the following a new way is presented to obtain the solution quality of optimally solved subproblems substantially faster than starting from scratch.

The GLSPST and the threshold accepting framework introduced above will serve as an example to illustrate the features and proceedings of this new approach. Applying this method to the GLSPST we are able to

- evaluate the correct (and optimal) holding costs for each accepted solution:
  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. Therefore, a *reoptimization* algorithm is used to solve the MCFP.
- recognize and refuse too expensive candidates at an early stage:
  Threshold accepting refuses a candidate if its objective function value exceeds the objective function value of the current solution by a certain threshold. But this is also true 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, an increasing sequence of lower bounds to the MCFP is generated and a candidate is refused as soon as the first of these lower bounds exceeds the threshold.

This new way of combining local search with *dual reoptimization* aims to achieve a better solution quality than solving the MCFP heuristically. On the other hand, moderate computation times can be expected since too expensive candidates are rejected early.

Embedding dual reoptimization into threshold accepting: Assume a current (feasible or infeasible) solution h to the GLSPST is given and the problem P^h is solved to optimality. A new candidate r is accepted as new solution $h+1$ if its holding costs $hc^r = \sum_{j,t} h_j I^r_{jt} = \sum_{(k,l)\in\mathscr{A}^r} c_{kl} X^r_{kl}$ plus setup costs $sc^r = \sum_{i,j,s} s_{ij} z^r_{ijs}$ are lower than $hc^h + sc^h$, the costs of the current solution, plus the current threshold value $Th(hc^h + sc^h)$. Thus, we refuse the candidate if

$$hc^r \geqslant \lambda \quad \text{with} \quad \lambda := (hc^h + sc^h)(1 + Th) - sc^r. \tag{9}$$

In order to refuse the new candidate r, it is not necessary to know hc^r exactly, if there is a lower bound κ to hc^r that is greater or equal to λ because of

$$hc^r \geqslant \kappa \geqslant \lambda. \tag{10}$$

We use Eq. (10) to improve the threshold accepting implementation presented so far.

Further assume that – along with the primal problem P^h – the corresponding dual problem D^h is solved to optimality. D^h is given by

$$\text{maximize} \sum_{(k,l)\in\mathscr{A}^h} (l^h_{kl}v^h_{kl} - u^h_{kl}w^h_{kl}) + \sum_{k\in\mathscr{N}} b^h_k p^h_k$$

subject to

$$v^h_{kl} - w^h_{kl} \leqslant \tilde{c}^h_{kl} \quad \forall (k,l) \in \mathscr{A}^h,$$

$$v^h_{kl}, w^h_{kl} \geqslant 0 \quad \forall (k,l) \in \mathscr{A}^h,$$

with dual (arc-)variables v^h_{kl}, w^h_{kl} for each arc $(k,l) \in \mathscr{A}^h$, dual (node-)variables p^h_k (dual prices, node potentials) for each node $k \in \mathscr{N}$ and dual costs $\tilde{c}^h_{kl} := c_{kl} - p^h_k + p^h_l$ for each $(k,l) \in \mathscr{A}^h$.

The neighborhood operations determining the new candidate r cause changes in the input data of only a few arcs $\Delta\mathscr{A}^r$ and nodes $\Delta\mathscr{N}^r$, i.e. in l^r_{kl} or u^r_{kl} for arcs $(k,l) \in \Delta\mathscr{A}^r$ and in b^r_k for nodes $k \in \Delta\mathscr{N}^r$. The other input data of the current solution h are still valid for all arcs $(k,l) \in \mathscr{A}^h \setminus \Delta\mathscr{A}^r$ and nodes $k \in \mathscr{N} \setminus \Delta\mathscr{N}^r$.

We can easily construct a dual feasible solution to D^r by setting:

$$v^r_{kl} := \tilde{c}^h_{kl}; \quad w^r_{kl} := 0 \qquad \forall (k,l) \in \Delta\mathscr{A}^r \quad \text{if } \tilde{c}^h_{kl} \geqslant 0,$$
$$v^r_{kl} := 0; \quad w^r_{kl} := -\tilde{c}^h_{kl} \quad \forall (k,l) \in \Delta\mathscr{A}^r \quad \text{otherwise.}$$

All other variables remain unchanged.

Since $\Delta\mathscr{A}^r$ and $\Delta\mathscr{N}^r$ are small, the change in holding costs Δhc^r is computed very fast by

$$\Delta hc^r := \sum_{(k,l)\in\Delta\mathscr{A}^r} [(l^r_{kl}v^r_{kl} - u^r_{kl}w^r_{kl}) - (l^h_{kl}v^h_{kl} - u^h_{kl}w^h_{kl})] + \sum_{k\in\Delta\mathscr{N}^r} (b^r_k - b^h_k)p^h_k. \tag{11}$$

With $\kappa := hc^h + \Delta hc^r$, there is a lower bound to hc^r since the cost of a feasible solution of the dual maximization problem D^r is a lower bound to the optimal solution of the corresponding primal problem P^r (note that both problems are feasible if $\sum_{i,j\neq i,s\in S_t} a_j m_j z^r_{ijs} \leqslant K_t - \sum_{i,j,s\in S_t} st_{ij} z^r_{ijs}$ for all t).

If $\kappa \geqslant \lambda$, the candidate r is to be refused. If $\kappa < \lambda$, the candidate has to be scanned as shown in Fig. 2. We initialize the primal and dual basis solutions that correspond to κ. Then, a dual network flow algorithm is used to execute a basis change that leads to a new $\kappa^{\text{new}} \geqslant \kappa$ (see Appendix A). If $\kappa^{\text{new}} \geqslant \lambda$, the candidate is refused. Otherwise, the new basis is checked for optimality. If optimality is proven, the candidate r is accepted as the new current solution $h+1$ since $\kappa^{\text{new}} = hc^r = hc^{h+1} < \lambda$. If the new basis is not optimal, this procedure is repeated until the candidate is refused (since $\kappa^{\text{new}} \geqslant \lambda$) or accepted (because optimality is proven).

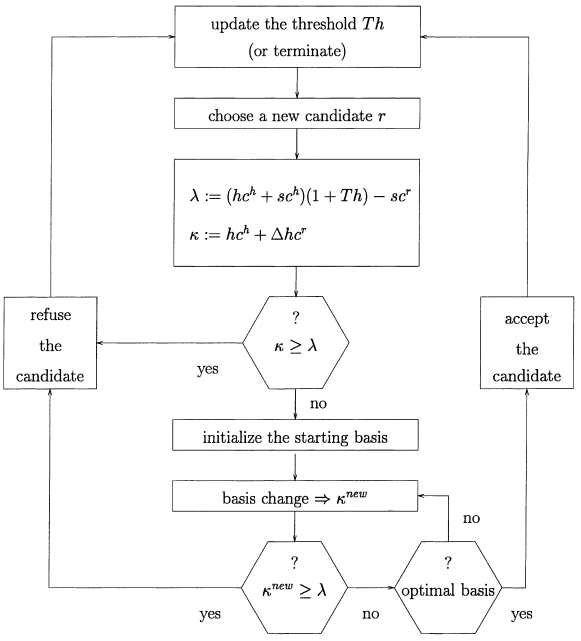

Fig. 2. Combination of threshold accepting with a dual reoptimization procedure for a current solution h and a new candidate r.

The main characteristics of this new algorithm are the following:

- Dual prices of the current solution h are used to sort out expensive candidates very fast (Eq. (11)).
- An unacceptable candidate r is refused early because the dual maximization problem D^r needs not to be solved to optimality.
- Each acceptable candidate is evaluated with its minimal holding cost and the respective lotsizes by use of a very efficient network flow algorithm.

Initialization of a starting basis for reoptimization: Several dual or primal–dual algorithms can be used for the basis changes. We employ the dual algorithm of Ali et al. [2] because it works in reverse analogy to the generally accepted primal algorithms (cf. Refs. [7,1], for example) so that very efficient data structures of the primal algorithms can be used for implementation. For future work it would be worthwhile to compare this implementation with other dual algorithms like the RELAX-code of Bertsekas [5,6], for example.

We use the following notation:

$$\mathscr{L}^r := \{(k,l) \in \mathscr{A}^r : X^r_{kl} = l^r_{kl}\}$$
the set of arcs at the lower bound,

$$\mathscr{U}^r := \{(k,l) \in \mathscr{A}^r : X^r_{kl} = u^r_{kl}\}$$
the set of arcs at the upper bound,

$$\mathscr{I}^r := \{(k,l) \in \mathscr{A}^r : X^r_{kl} < l^r_{kl} \vee X^r_{kl} > u^r_{kl}\}$$
the set of arcs with an (primal) infeasible flow.

The dual algorithm of Ali et al. [2] needs a (dual) starting basis that satisfies the following properties:

(i) There is a set of basis arcs (basis variables) $\mathscr{B}^r \subset \mathscr{A}^r$ that spans a tree in the graph $\mathscr{G}^r$ and $\tilde{c}^r_{kl} = 0$ for all $(k,l) \in \mathscr{B}^r$. The basis arcs are either primal feasible ($l^r_{kl} \leqslant X^r_{kl} \leqslant u^r_{kl}, (k,l) \in \mathscr{B}^r$) or primal infeasible ($(k,l) \in \mathscr{I}^r \cap \mathscr{B}^r$).

(ii) All non-basis arcs $\mathscr{NB}^r = \mathscr{A}^r \setminus \mathscr{B}^r$ are primal feasible and either at their lower bound or at their upper bound ($(k,l) \in \mathscr{L}^r \cup \mathscr{U}^r$ for all $(k,l) \in \mathscr{NB}^r$).

(iii) The optimality criterion is satisfied:

$$\tilde{c}^r_{kl} \geqslant 0 \quad \forall (k,l) \in \mathscr{L}^r,$$
$$\tilde{c}^r_{kl} \leqslant 0 \quad \forall (k,l) \in \mathscr{U}^r.$$

Assume, there is a current solution h and P^h and D^h are solved to optimality. So $\mathscr{B}^h$ and $\mathscr{NB}^h$ are given that fulfill (i–iii) and $\mathscr{I}^h = \emptyset$. A starting basis (in general non-optimal) for the problems P^r, D^r that satisfies (i–iii) is constructed by the following procedure:

Set $\mathscr{I}^r := \emptyset$.

1. If $(n^C_t, n^D_{jt}) \in \Delta\mathscr{A}^r$, since the number of lots of product j changes in a macro-period t:
   - If $(n^C_t, n^D_{jt}) \in \mathscr{B}^h$ and $X^h_{n^C_t, n^D_{jt}} < l^r_{n^C_t, n^D_{jt}}$ or $X^h_{n^C_t, n^D_{jt}} > u^r_{n^C_t, n^D_{jt}}$, set $\mathscr{I}^r := \left\{(n^C_t, n^D_{jt})\right\}$.
   - If $(n^C_t, n^D_{jt}) \in \mathscr{NB}^h$, there is a unique path (simple path, cf. Ref. [1]) in $\mathscr{B}^h$ between node n^C_t and node n^D_{jt}. This path and the arc (n^C_t, n^D_{jt}) define a cycle. Set

$$X^r_{n^C_t, n^D_{jt}} := \begin{cases} l^r_{n^C_t, n^D_{jt}} & \text{if } \tilde{c}^h_{kl} \geqslant 0, \\ u^r_{n^C_t, n^D_{jt}} & \text{if } \tilde{c}^h_{kl} < 0, \end{cases}$$

   and

$$\Delta := X^r_{n^C_t, n^D_{jt}} - X^h_{n^C_t, n^D_{jt}}.$$

   Set

$$X^r_{kl} := \begin{cases} X^h_{kl} + \Delta & \forall (k,l) \text{ in the cycle with the same direction as } (n^C_t, n^D_{jt}), \\ X^h_{kl} - \Delta & \forall (k,l) \text{ in the cycle with the opposite direction as } (n^C_t, n^D_{jt}). \end{cases}$$

   If some arcs in the cycle violate their bounds, update $\mathscr{I}^r$.
2. If $n^C_t, n^D \in \Delta\mathscr{N}^r$, since the new setup pattern changes setup time consumption in some macro-period t:

Increase/decrease the flow X^h_{kl} (or the new flow if X^r_{kl} was updated in step 1) for all arcs $(k,l) \in \mathscr{B}^h$ along the path between n^C_t and n^D respective to the change in capacity. If some arc (k,l) on this path now violates its upper or lower bounds, update $\mathscr{I}^r$.

Basis change: $X^r_{\cdot\cdot}$ and $\mathscr{I}^r$ are updated as explained before, $l^r_{\cdot\cdot}, u^r_{\cdot\cdot}, b^r_{\cdot}, \mathscr{A}^r, \Delta\mathscr{A}^r$ and $\Delta\mathscr{N}^r$ are updated by the data changes caused by the neighborhood operation. The other variables ($p^r_{\cdot}, \tilde{c}^r_{\cdot\cdot}, \mathscr{B}^r, \mathscr{NB}^r$) are initialized with the values of the current solution h. As shown, there is a starting basis that satisfies (i–iii).

A basis change is made by determining one arc (m,n) leaving the basis and a second arc (o,p) entering the basis so that the new basis is a spanning tree, again satisfying (i–iii). The basis changes to determine the optimal solutions of P^r and D^r are done with the algorithm of Ali et al. [2]. A technical description of the basis change and hints for implementation are given in Appendix A.

As discussed earlier, the TA algorithm starts with an initial infeasible solution where no production is allowed and demand is satisfied by the fictitious period (dummy supply node n^S), exclusively. Note that an optimal starting basis for the respective network flow problems P^0 and D^0 may be easily constructed. No production arcs are in the arc set $\mathscr{A}^0$ and all other arcs except for the ending inventory arcs are in the basis $\mathscr{B}^0$.

In the following, this solution procedure embedding dual reoptimization into threshold accepting is called TADR.

3.2. A simulated annealing approach

TADR is generally applicable to MIP-models where an MCFP is left after fixing the integer variables. Furthermore, this approach could be extended to MIP-models where the embedded subproblems are (non-network flow) linear problems by reoptimization with a dual simplex algorithm. However, this may lead to unacceptable running times if the subproblems grow too large.

On the other hand, local search methods other than TA may profit from a combination with dual reoptimization procedures if an early refusing of candidates is possible without solving the remaining MCFP (LP-problem) to optimality for all candidates. The *great deluge algorithm* and *record-to-record travel* of Dueck [14] suit very well.

The embedding of dual reoptimization into *simulated annealing* (cf. Ref. [16], for example) is not obvious. Like threshold accepting, SA accepts a candidate r as a new solution $h+1$ if the objective function value o^r of the candidate is better than the objective function value o^h of the current solution h. To overcome local optima, a candidate r resulting in a worse objective function value is accepted with a certain acceptance probability Π^r, too. For demonstration purposes we use the acceptance probability $\Pi^r := \exp((o^h - o^r)/T^r)$, suggested in Ref. [26], which depends on the change in the objective function value *and* the number of candidates tested so far, because the *temperature* T^r (and also the acceptance probability) decreases with the number of candidates increasing.

Therefore, candidate r has to be refused if it is worse than the current solution and if a certain value $\rho \in (0,1)$, drawn at random from a uniform distribution, exceeds the acceptance probability, i.e. if $\rho \geqslant \Pi^r$ is fulfilled. In other words, candidate r can be refused if

$$o^r \geqslant o^h - T^r \log(\rho)$$

holds. Again, this is also true if the objective function value o^r of candidate r is replaced by a lower bound.

We can easily apply this principle to the GLS-PST by simply substituting λ in Eqs. (9) and (10) and Fig. 2 with

$$\lambda^{\mathrm{SA}} := hc^h + sc^h - sc^r - T^r \log(\rho).$$

The solution procedure resulting from this combination of simulated annealing with dual reoptimization will be called SADR.

4. Computational results

Computational tests are executed using the operating system *Linux* and the gcc-compiler on a personal computer with a Pentium Pro 200 central processing unit (CPU).

The (percentage) 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 3000 steps, a run of TADR is stopped.

When testing SADR, the annealing schedule $T^r := 1000 \cdot (0.8)^q$ for $1000(q-1) < r \leqslant 1000q$ and $q = 1, \ldots, Q = 18$ performed best. Thereby, the acceptance probability Π^r and the temperature T^r are kept constant for different stages (plateaus) q of the search. Each run of SADR ends after 18 000 candidate tests.

Various numerical tests proved that the best ratio between solution quality and computation time is achieved if not only a single run of TADR or SADR is executed, but if the best solution of

ITER independent runs is chosen. Thereby, ITER is set to 25 for TADR and to 17 for SADR.

Thus, for a single problem instance p the percentage deviation of the best objective function value of ITER independent runs from the objective function value of the best known solution o_p^{best} is measured. For a set of test problems P given, the percentage deviation is then averaged over all problem instances $p \in P$. To overcome statistical interference, a random sample (containing all problem instances of P) of length ten is used to estimate the percentage deviation pd and to give an insight into the solution quality of the algorithm *heur*:

$$pd := \frac{1}{10}\sum_{e=1}^{10}\left[\frac{1}{|P|}\sum_{p\in P}\frac{\min_{f=1}^{\text{ITER}}\left\{o_{efp}^{heur}\right\} - o_p^{\text{best}}}{o_p^{\text{best}}}\cdot 100\right],$$

where o_{efp}^{heur} is the objective function value of problem instance p resulting from run f in random sample e.

4.1. Problems without setup times

To evaluate the performance of the dual reoptimization procedure, we first deal with problems without setup times. The test problems of Fleischmann and Meyr [19] are used to compare TADR and SADR with MOD, the local search procedure for the GLSP without setup times which provides the highest quality. MOD employs threshold accepting for fixing the setup pattern and solves the remaining network flow subproblems, heuristically [19]. Furthermore, results of BACLSD, a deterministic heuristic designed by Haase [23] that is also able to solve these instances, are presented.

For a comprehensive description of the problem data, we refer to Ref. [19]. Table 1 shows results of four problem classes established by Haase [23], each containing ten optimally solvable problem instances with the same number of products (J), number of macro-periods (T) and capacity utilization (U). Results are averaged over all instances of a class where pd denotes the percentage deviation from the optimal solution. The average number of optimally solved instances (out of 40) is given, together with the respective minimum and maximum number within all ten random samples. Note that BACLSD is a deterministic algorithm. Hence, only one run is executed.

Further, the percentage deviations from the best known solutions are shown as the average

Table 1
Problems without setup times

	$J/T/U$		#	BACLSD	MOD	TADR	SADR
				(1 s., 1 r.)	(10 samples, best of 25/25/17 runs)		
Haase	4/6/80	$pd\bigstar$	10	2.70	0.26	0.05	0.00
	4/6/90	$pd\bigstar$	10	5.13	0.76	0.24	0.07
	4/5/90	$pd\bigstar$	10	7.89	0.72	0.15	0.00
	5/5/90	$pd\bigstar$	10	6.68	0.97	0.28	0.11
	All	$pd\bigstar$	40	5.60	0.68	0.18	0.05
	Aver. optimal		40	3.0	22.0	33.9	38.3
	(min,max) opt.		–	–	(22,25)	(32,37)	(36,39)
PR		$pd\diamondsuit$	4	10.00	2.70	1.86	1.32
TV		$pd\diamondsuit$	32	13.55	4.95	3.89	3.77
CPU-seconds			76	–	33.65	13.53	35.88

Percentage deviation from optimal ($pd\bigstar$) or best known ($pd\diamondsuit$) solution.
Aver. optimal: Average number of optimal solutions found.
(min,max) opt.: Minimal/maximal numbers of optimal solutions found.
Average values of # problem instances.

over four practical problems of the food industry, the so-called PR-problems ($J = 9, T = 8$ and $J = 3/4, T = 26$, respectively), and over 32 TV-problems as discussed in Ref. [19] ($J = 8, T = 8, U$ varies between 61% and 93%).

Finally, CPU-seconds are averaged over all 76 problem instances to give a quick insight into computation time performance of the new heuristics. BACLSD is left open since it is coded with Turbo Pascal 6.0 (Borland). However, the average running time may be estimated to be about less than one second.

Comparison of TADR with MOD proves the effectiveness of the dual reoptimization procedure when applied to threshold accepting. The solution quality is clearly improved due to the optimal solution of the MCFP. Surprisingly, optimal instead of heuristic solution of the network flow subproblems does not lead to an increase of running time. On the contrary, the dual procedure with early refusing of unacceptable candidates enables a substantial reduction of computation time.

Combining dual reoptimization with simulated annealing is very promising, too. Fixing the setup pattern by simulated annealing (using SADR) instead of threshold accepting (TADR) the quality of solution can be improved, again, at the cost of higher computation times. On the average more than 95% of the Haase problems are solved to optimality by SADR.

4.2. Optimally solvable problems with setup times

Smith-Daniels and Smith-Daniels [32] describe lotsizing and sequencing problems from the process industries with $J = 4$ and $T = 5$. Thereby, two items are produced and packaged in two different package sizes. We modify the modeling assumptions of Smith-Daniels and Smith-Daniels, slightly, so that they fit to GLSPST and use the MIP-solver MOPS [33] to solve these problems to optimality. So we get an impression of the solution quality of the new heuristics when applied to (small) problems with sequence-dependent setup times.

Thus, – in contrast to Ref. [32] – no backlogging is allowed and the setup state is conserved after idle periods. Setup times only occur between different items. Changeovers between different items are twice as expensive as changeovers between the same items in different package sizes. The magnitude of the input data is taken from the problems of Smith-Daniels and Smith-Daniels.

Twelve problem instances are constructed (cf. Ref. [32]) differing in

1. the load factor between families, which is either balanced (b) (50:50) or unbalanced (u) (20:80),
2. the coefficient of variation of demand of each single product, which is either low (l) (0.05) or high (h) (0.5),
3. the coefficient of variation of the total demand of all products (load variability), which is either low (0.05) or high (0.5),
4. the level of changeover costs and changeover times, which is either low (400–800, 15 min) or high (4000–8000, 30 min).

Note that utilization is at the high level of 97%, even if setup times are ignored.

We compare the dual reoptimization procedures with MODST which is a straightforward extension of MOD in order to respect sequence-dependent setup times. For that purpose, capacity K_t has to be reduced by setup times $\sum_{i,j,s\in S_t} st_{ij} z_{ijs}$ before solving the network flow subproblems heuristically. This is not problematic since setup times are known after fixing the setup pattern by threshold accepting.

Table 2 shows the percentage deviation from the optimal objective function value for each of these problem instances and the average number of problems (out of 12) solved to optimality. Computation times are presented the same way as in Section 4.1.

The solution quality of the dual reoptimization procedures is very high. TADR and SADR solve almost all problems to optimality. MODST performs by far worse since the MCFP is solved heuristically, only. Obviously, six of the twelve problems cannot be solved to optimality by MODST. Again, TADR shows the best computation time performance.

Generally, the problems with a balanced load factor and low load variability (b·l·) seem to be of an easily solvable structure.

Table 2
Small problems with setup times

1–4		#	MODST	TADR	SADR
blll	*pd*★	1	0.00	0.00	0.00
bhll	*pd*★	1	0.00	0.00	0.00
bhhl	*pd*★	1	0.58	0.27	0.00
ulll	*pd*★	1	4.91	0.00	0.00
uhll	*pd*★	1	0.64	0.00	0.00
uhhl	*pd*★	1	0.02	0.06	0.00
bllh	*pd*★	1	0.00	0.00	0.00
bhlh	*pd*★	1	0.00	0.00	0.00
bhhh	*pd*★	1	1.86	0.01	0.04
ullh	*pd*★	1	4.98	0.00	0.00
uhlh	*pd*★	1	5.22	0.00	0.00
uhhh	*pd*★	1	0.00	0.00	0.00
All	*pd*★	12	1.52	0.03	0.00
Aver. optimal		12	5.9	11.5	11.6
(min,max) opt.			(5,6)	(10,12)	(11,12)
CPU–seconds		12	25.5	17.5	30.5

pd★: Percentage deviation from optimal solution.
Aver. optimal: Average number of optimal solutions found.
(min,max) opt.: Minimal/maximal numbers of optimal solutions found.
Average values of # problem instances; 10 samples, best of 25/25/17 runs.

4.3. Practical problems with setup times

We use some practical problems of the consumer goods industries to test the behavior of MODST, TADR and SADR when the number of products or macro-periods is increased and sequence-dependent setup costs and sequence-independent setup times are present.

There are 44 test problems with $T = 4$ and 2–16 products. The problem instances are pooled in four problem classes as shown in Table 3. The average number of products per problem instance is 9.3 products. The utilization without setup times (net utilization) varies between 69% and 94%. Utilization including setup times (gross utilization) varies between 70% and 97% if some typical feasible solutions are taken as a basis. So setup times cover about 1–3% of total capacity available.

Furthermore, 42 problem instances with $T = 8$ are tested (cf. Table 4). The number of products varies between 5 and 18 with an average of 11.8 products per instance. Net utilization is in the range of 79–91% while gross utilization of typical solutions varies between 80% and 94%. All problem data are available from the author.

Again, the percentage deviation from the best known solution is measured. With a whole of ten random samples the average, minimum and maximum numbers of problems (out of $|P| = 44$ and $|P| = 42$, respectively) are shown where the best known solution o_p^{best} of a problem instance $p \in P$ is achieved. Average computation times (CPU-seconds) are now presented for each problem class, individually.

Looking at both, the problems with four and eight macro-periods, dual reoptimization performs clearly better than the heuristic solution of the network flow subproblem which is done by MODST. Again, SADR provides the best solution quality. This behavior becomes more and more

Table 3
Practical problems with setup times and $T = 4$ macro-periods

J		#	MODST	(CPU)	TADR	(CPU)	SADR	(CPU)
2–5	*pd*◇	9	0.09	(14.4)	0.05	(7.3)	0.05	(15.9)
6–10	*pd*◇	19	0.45	(31.4)	0.11	(11.6)	0.07	(19.0)
11–15	*pd*◇	15	0.48	(41.8)	0.27	(14.2)	0.19	(20.2)
16	*pd*◇	1	0.38	(55.0)	0.17	(17.8)	0.09	(23.4)
All	*pd*◇	44	0.38	(32.0)	0.15	(11.7)	0.11	(18.9)
Aver. best		44	12.4		19.3		24.6	
(min,max) best			(10,14)		(16,23)		(21,27)	

pd◇: Percentage deviation from best known solution.
Aver. best: Average number of best known solutions found.
(min/max) best: Minimal/maximal number of best known solutions found.
Average values of # problem instances; 10 samples, best of 25/25/17 runs.

Table 4
Practical problems with setup times and $T = 8$ macro-periods

J		#	MODST	(CPU)	TADR	(CPU)	SADR	(CPU)
5	$pd\diamondsuit$	2	1.53	(36.6)	0.26	(15.1)	0.17	(26.4)
6–10	$pd\diamondsuit$	10	1.66	(66.8)	0.64	(21.3)	0.50	(26.9)
11–15	$pd\diamondsuit$	26	1.62	(95.4)	0.94	(30.5)	0.51	(29.0)
16–18	$pd\diamondsuit$	4	2.75	(131.3)	2.02	(41.1)	1.44	(32.6)
All	$pd\diamondsuit$	42	1.73	(89.2)	0.94	(28.6)	0.58	(28.7)
Aver. best		42	0.3		3.1		7.3	
(min,max) best			(0,1)		(1,5)		(5,10)	

$pd\diamondsuit$: Percentage deviation from best known solution.
Aver. best: Average number of best known solutions found.
(min/max) best: Minimal/maximal number of best known solutions found.
Average values of # problem instances; 10 samples, best of 25/25/17 runs.

obvious the "larger" the problems are, i.e. the more products and/or macro-periods are involved.

Amazingly, TADR seems to be more sensitive to an increasing number of products and periods than SADR. This is true for the quality of solution and the computation time as well. Considering the largest problems with 8 macro-periods and 11–18 products SADR performs best with respect to *both criteria*. This is probably due to the flexible stopping rule that is applied in the threshold accepting implementation. SADR, however, terminates always after 306 000 candidate tests.

5. Summary

We introduced the GLSPST, a model for simultaneous lotsizing and scheduling of several products on a single, capacitated production line when sequence-dependent setup times are present. Deterministic, dynamic demand is to be met without back-logging with the objective of minimizing inventory holding and sequence-dependent setup costs.

This problem is tackled by fixing the setup sequence, i.e. the binary variables of the model, applying local search. For each candidate setup sequence to be tested a network flow problem has to be solved in order to compute the holding costs of the candidate.

Greedy heuristics are known to provide an easy and fast way for solving those network flow subproblems, but the solutions are of minor quality. Solving each network flow problem to optimality consumes too much computation time, if it has to be done for each candidate individually and from scratch.

We presented a new mathematical solution method that improves these two possible approaches by using *dual network flow reoptimization*. In doing this, the network flow subproblems can be solved to optimality in a very effective and fast way. Already available information of the current solution is used to evaluate a new candidate by means of *reoptimization*. This proved to be very efficient since the current solution and the new candidate only differ by slight changes in the underlying data. *Dual* prices enable an early refusing of unacceptable candidates. This is especially the case before the reoptimization procedure itself starts. Thereby, new candidates are drawn at random (actually, a rather unsophisticated and inefficient way, but useful to overcome local optima); "unserious" candidates, however, may be identified quickly by the dual procedure.

Computational tests have shown that the new procedure is an effective tool to solve lotsizing and scheduling problems with sequence-dependent setup costs and setup times heuristically. Dual reoptimization in combination with threshold accepting or simulated annealing improves the heuristic solution of the network flow subproblems clearly with respect to solution quality and computation time. Thereby, simulated annealing is superior to threshold accepting if large-scale problems are considered.

Dual reoptimization seems to be generally applicable to MIP-models where a linear planning problem is left after fixing the binary variables by local search methods like threshold accepting and simulated annealing. Therefore, it is a challenge for future work to apply dual reoptimization to further problems with an appropriate structure. Because of the practical relevance one of these problems should incorporate the extension of the GLSPST to parallel production lines.

Acknowledgements

The author is grateful to Prof. Dr. Bernhard Fleischmann and the three unknown referees for their helpful support.

Appendix A. Basis change

For the sake of clarity, we describe a change from an old basis of candidate r to a new basis of candidate r by omitting the index r in all data and variables concerned. A basis change is made by determining one arc (m,n) leaving the basis and a second arc (o,p) entering the basis so that the new basis is a spanning tree, again satisfying (i–iii):

1. Choose an arc $(m,n) \in \mathscr{I}$ that leaves the basis $\mathscr{B}$. The basis splits up into two subtrees $\mathscr{T}_m$ and $\mathscr{T}_n$ where $\mathscr{T}_m$ ($\mathscr{T}_n$) denotes the subtree with node $m(n)$ (see Fig. 3). Set

$$\mathscr{C}^+_{mn} := \{(k,l) \in \mathscr{NB}: k \in \mathscr{T}_m, l \in \mathscr{T}_n\}$$
$$\mathscr{C}^-_{mn} := \{(k,l) \in \mathscr{NB}: l \in \mathscr{T}_m, k \in \mathscr{T}_n\}.$$

Thereby, (m,n) is an arc in $\mathscr{I}$ where the number of nodes in the smaller subtree is minimal:

$$(m,n) \in \left\{ (\overline{k},\overline{l}) \in \mathscr{I}: \min\left\{|\mathscr{T}_{\overline{k}}|, |\mathscr{T}_{\overline{l}}|\right\} = \min_{(k,l)\in\mathscr{I}}\{|\mathscr{T}_k|, |\mathscr{T}_l|\}\right\},$$

where $|\mathscr{T}_k|$ = cardinality of subtree $\mathscr{T}_k$ = number of nodes in subtree $\mathscr{T}_k$.

2. Compute Δ_{mn}, the change of the primal flow:

$$\Delta_{mn} := \begin{cases} u_{mn} - X_{mn}, & \text{if } X_{mn} > u_{mn}, \\ l_{mn} - X_{mn} & \text{if } l_{mn} > X_{mn}. \end{cases}$$

3. Determine the arc $(o,p) \in \mathscr{NB}$ that enters the new basis so that all non-basis arcs remain dual feasible:

If $X_{mn} > u_{mn}$: choose arc (o,p) with

$$|\tilde{c}_{op}| = \min\left\{ \min_{(k,l)\in\mathscr{C}^+_{mn}}\left\{\tilde{c}_{kl}: (k,l) \in \mathscr{L}\right\}; \min_{(k,l)\in\mathscr{C}^-_{mn}}\left\{ -\tilde{c}_{kl}: (k,l) \in \mathscr{U}\right\}\right\}$$

and set $\Delta\tilde{c}_{op} := -|\tilde{c}_{op}|$.

If $X_{mn} < l_{mn}$: choose arc (o,p) with

$$|\tilde{c}_{op}| = \min\left\{ \min_{(k,l)\in\mathscr{C}^+_{mn}}\left\{ -\tilde{c}_{kl}: (k,l) \in \mathscr{U}\right\}; \min_{(k,l)\in\mathscr{C}^-_{mn}}\left\{\tilde{c}_{kl}: (k,l) \in \mathscr{L}\right\}\right\}$$

and set $\Delta\tilde{c}_{op} := +|\tilde{c}_{op}|$.

If $\Delta\tilde{c}_{op} = 0$, the problem P is infeasible; this may not occur if the setup pattern is set correctly (cf. Section 3.1).

4. Determine Δhc, the change of the objective function value:

$$\Delta hc := \left[\left(\sum_{(k,l)\in\mathscr{L}\cap\mathscr{C}^+_{mn}} l_{kl} + \sum_{(k,l)\in\mathscr{U}\cap\mathscr{C}^+_{mn}} u_{kl} \right) - \left(\sum_{(l,k)\in\mathscr{L}\cap\mathscr{C}^-_{mn}} l_{lk} + \sum_{(l,k)\in\mathscr{U}\cap\mathscr{C}^-_{mn}} u_{lk} \right) + F \right] \cdot \Delta\tilde{c}_{op},$$

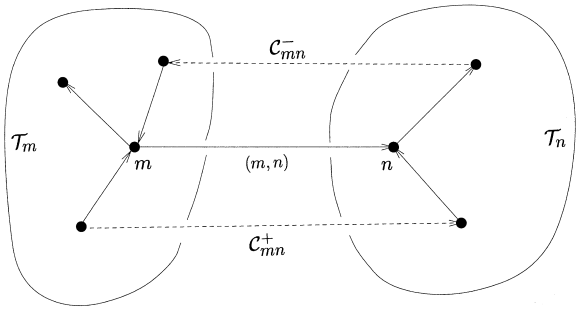

Fig. 3. Example for a basis change with a leaving arc (m,n).

where

$$F := \begin{cases} -\sum_{k \in \mathscr{T}_m} b_k & \text{if } |\mathscr{T}_m| \leqslant |\mathscr{T}_n|, \\ +\sum_{l \in \mathscr{T}_n} b_l & \text{otherwise.} \end{cases}$$

If Δhc is high enough (cf. Section 3.1), terminate and refuse candidate r.

5. Update the dual prices:

$$p_k^{\text{new}} := p_k + H \quad \forall k \in \mathscr{K}$$

where

$$H := \begin{cases} -\Delta\tilde{c}_{op} \\ +\Delta\tilde{c}_{op} \end{cases} \quad \text{and}$$

$$\mathscr{K} := \begin{cases} \mathscr{T}_m & \text{if } |\mathscr{T}_m| \leqslant |\mathscr{T}_n|, \\ \mathscr{T}_n & \text{otherwise.} \end{cases}$$

6. Update the primal flow:

$$X_{kl}^{new} := X_{kl} \begin{cases} +\Delta_{mn} & \forall (k,l) \text{ in the basis cycle with the same direction as } (m,n), \\ -\Delta_{mn} & \forall (k,l) \text{ in the basis cycle with the opposite direction as } (m,n). \end{cases}$$

If bounds are violated, update $\mathscr{I}$.

7. Basis update and update of all information; if $\mathscr{I} = \emptyset$, optimality is proven; then terminate and accept candidate r as new solution $h+1$.

Very efficient data structures, originally designed for the primal simplex algorithm, are available for storage and update of the basis. Ali et al. [1] give a comprehensive survey of these methods which mainly originate from the work of Glover and Klingman [20].

As supposed in Ref. [2], a threaded index is used to store a basis with the dummy demand n^D as the (fixed) root of the tree. Besides the predecessor information, the number of nodes in the maximal subtree rooted at node k has to be stored for each node k (cardinality of a subtree with root k). Thereby, a fast selection of the leaving arc (m,n) is possible. Additional storage of the last node in the maximal subtree rooted at node k enables an efficient identification of all nodes in this subtree. The preorder distance of each node in the thread together with information about the respective incoming/outgoing arcs is used to determine the cutsets $\mathscr{C}_{mn}^+$ and $\mathscr{C}_{mn}^-$ by visiting all nodes of the smaller subtree.

The update of the thread is done as supposed in Ref. [20]. For further implementation issues the reader is referred to Bradley et al. [7].

References

[1] A.I. Ali, R.V. Helgason, J.L. Kennington, H.S. Lall, Primal simplex network codes: State-of-the-art implementation technology, Networks 8 (1978) 315–339.

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

[3] H.C. Bahl, L.P. Ritzman, J.N.D. Gupta, Determinig lot sizes and resource requirements: A review, Operations Research 35 (3) (1987) 329–345.

[4] T. Baker, J.A. Muckstadt Jr, The CHES problems, Technical Paper, Chesapeake Decision Sciences, Inc., Providence, NJ, 1989.

[5] D.P. Bertsekas, Linear Network Optimization, 2nd ed., MIT Press, Cambridge, MA, 1992.

[6] D.P. Bertsekas, P. Tseng, RELAX–IV: A faster version of the relax code for solving minimum cost flow problems, Technical Paper, Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, MA, 1994.

[7] G.H. Bradley, G.G. Brown, G.W. Graves, Design and implementation of large scale primal transshipment algorithms, Management Science 24 (1) (1977) 1–34.

[8] G.M. Campbell, Using short-term dedication for scheduling multiple products on parallel machines, Production and Operations Management 1 (3) (1992) 295–307.

[9] D. Cattrysse, M. Salomon, R. Kuik, L.N. Van Wassenhove, A dual ascent and column generation heuristic for the discrete lotsizing and scheduling problem with setup times, Management Science 39 (4) (1993) 477–486.

[10] R.D. 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.

[11] A. Drexl, B. Fleischmann, H.O. Günther, H. Stadtler, H. Tempelmeier, Konzeptionelle Grundlagen kapazitätsorientierter PPS-systeme, Zeitschrift für betriebswirtschaftliche Forschung 46 (12) (1994) 1022–1045.

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

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

[14] G. Dueck, New optimization heuristics, Journal of Computational Physics 104 (1993) 86–92.

[15] G. Dueck, T. Scheuer, Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing, Journal of Computational Physics 90 (1990) 161–175.
[16] R.W. Eglese, Simulated annealing: A tool for operational research, European Journal of Operational Research 46 (1990) 271–281.
[17] B. Fleischmann, The discrete lot-sizing and scheduling problem, European Journal of Operational Research 44 (1990) 337–348.
[18] B. Fleischmann, The discrete lot-sizing and scheduling problem with sequence-dependent setup costs, European Journal of Operational Research 75 (1994) 395–404.
[19] B. Fleischmann, H. Meyr, The general lotsizing and scheduling problem, OR Spektrum 19 (1) (1997) 11–21.
[20] F. Glover, D. Klingman, J. Stutz, Augmented threaded index method for network optimization, INFOR 12 (3) (1974) 293–298.
[21] M. Gopalakrishnan, D.M. Miller, C.P. Schmidt, A framework for modelling setup carryover in the capacitated lot sizing problem, International Journal of Production Research 33 (7) (1995) 1973–1988.
[22] K. Haase, Lotsizing and Scheduling for Production Planning, vol. 408, Lecture Notes in Economics and Mathematical Systems, Springer, Berlin, 1994.
[23] K. Haase, Capacitated lot-sizing with sequence dependent setup costs, OR Spektrum 18 (1996) 51–59.
[24] C. Jordan, Batching and Scheduling, vol. 437, Lecture Notes in Economics and Mathematical Systems, Springer, Berlin, 1996.
[25] S. Kang, K. Malik, L. Thomas, Lotsizing and scheduling on parallel machines with sequence-dependent setup costs, Technical Paper, Department of Business Administration, The Catholic University of Korea, Seoul, 1997, pp. 422–743.
[26] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[27] R. Kuik, M. Salomon, L.N. Van Wassenhove, Batching decisions: Structure and models, European Journal of Operational Research 75 (1994) 243–263.
[28] R. Kuik, M. Salomon, L.N. Van Wassenhove, J. Maes, Linear programming, simulated annealing and tabu search heuristics for lotsizing in bottleneck assembly systems, IIE Transactions 25 (1) (1993) 62–72.
[29] J.O. McClain, L.J. Thomas, E.N. Weiss, Efficient solutions to a linear programming model for production scheduling with capacity constraints and no initial stock, IIE Transactions 21 (2) (1989) 144–152.
[30] C.N. Potts, L.N. Van Wassenhove, Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity, Journal of the Operational Research Society 43 (5) (1992) 395–406.
[31] 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.
[32] V.L. Smith-Daniels, D.E. Smith-Daniels, A mixed integer programming model for lot sizing and sequencing packaging lines in the process industries, IIE Transactions 18 (1986) 278–285.
[33] U.H. Suhl, MOPS – mathematical optimization system, European Journal of Operational Research 72 (1994) 312–322.
[34] J. Teghem, M. Pirlot, C. Antoniadis, Embedding of linear programming in a simulated annealing algorithm for solving a mixed integer production planning problem, Journal of Computational and Applied Mathematics 64 (1995) 91–102.
[35] L.A. Wolsey, MIP modelling of changeovers in production planning and scheduling problems, European Journal of Operational Research 99 (1997) 154–165.