Innovative Applications of O.R

A Decomposition Approach for the General Lotsizing and Scheduling Problem for Parallel Production Lines

Herbert Meyr, Matthias Mann

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
SciVerse ScienceDirect
www.elsevier.com/locate/ejor

PII: S0377-2217(13)00269-5
DOI: http://dx.doi.org/10.1016/j.ejor.2013.03.036
Reference: EOR 11600

To appear in: *European Journal of Operational Research*

Received Date: 5 March 2012
Revised Date: 12 March 2013
Accepted Date: 23 March 2013

A Decomposition Approach for the General Lotsizing and Scheduling Problem for Parallel Production Lines

Herbert Meyr[a,*], Matthias Mann[b]

[a] *Universität Hohenheim, Lehrstuhl für Supply Chain Management (580C), 70 593 Stuttgart, Germany, Tel.: +49 (-711) 459–24590, Fax: –24599*
[b] *Wirtschaftsuniversität Wien, 1090 Wien, Austria*

Abstract

This paper presents a novel solution heuristic to the General Lotsizing and Scheduling Problem for Parallel production Lines (GLSPPL). The GLSPPL addresses the problem of simultaneously deciding about the sizes and schedules of production lots on parallel, heterogeneous production lines with respect to scarce capacity, sequence-dependent setup times and deterministic, dynamic demand of multiple products. Its objective is to minimize inventory holding, sequence-dependent setup and production costs. The new heuristic iteratively decomposes the multi-line problem into a series of single-line problems, which are easier to solve. Different approaches for decomposition and for the iteration between a modified multi-line master problem and the single-line subproblems are proposed. They are compared with an existing solution method for the GLSPPL by means of medium-sized and large practical problem instances from different types of industries. The new methods prove to be superior with respect to both solution quality and computation time.

Keywords: Scheduling; Heuristics; Simultaneous Lotsizing and Scheduling; Production

1. Introduction

Management concepts sometimes repeat. Currently seems to roll another "wave of lean management", preaching that inventories are waste. Inventories would blur a clear view on mistakes which are made in production planning. Thus, inventories should be reduced. All drivers that necessitate inventories should be eliminated by means of investments. This makes sense in many industries, but not in all. Some industries utilize expensive, automated machines. These cannot easily be replaced by newer ones allowing automated changeovers and thus generating almost no setup times or setup costs. They apply a machinery, which has grown over time. It consists of many machines which show a similar functionality, but have been purchased successively over one or several decades. There are, e.g., some older machines, which are less efficient, and a few newer ones, which are more efficient.

An example is the consumer goods industry. There are often only a few production stages (like Make and Pack) with only one of them being a bottleneck. However

[*]Corresponding author
Email addresses: H.Meyr@uni-hohenheim.de (Herbert Meyr), MMann@pi-ag.com (Matthias Mann)

– due to the above reasons – several heterogeneous, parallel production lines can be found on this bottleneck stage. Parallel means that these lines show a similar functionality, i.e., they can be used alternatively to produce (more or less) the same products. Heterogeneous means that they, nevertheless, do not need to be identical. This concerns production speeds, production coefficients, production costs, setup costs and setup times. Both setup costs and times may even be sequence-dependent. The individual work stations of a production line are usually connected by an automated transport system with a fixed cycle time. Thus, each production line can be considered as a single planning unit.

We focus on this kind of industries, more concretely on short- to medium-term production planning and scheduling in this kind of industries. Typically, many final items can be assigned to a few setup families. They have to be produced on stock on basis of demand forecasts. These demand forecasts are assumed to be deterministically known for several planning periods (e.g., weeks or months) of a finite planning horizon (e.g., a quarter or even a year). Thus, demands can dynamically vary per setup family and time period. Backlogging is not allowed. Changeovers between items of the same family are negligible. However, changeovers between items of different setup families may incur high, sequence-dependent changeover costs and changeover times. There is only a single bottleneck stage within the flow shop production system. However, this stage consists of several parallel, heterogeneous production lines with scarce capacity. The production lines share the same functionality, i.e., they can alternatively be used to fulfill the setup families' demands. Producing earlier than in the demanded periods incurs inventory holding costs. These inventory holding costs are in a crucial conflict with the setup costs. But also the heterogeneous production costs of the different lines may play an important role.

Thus, the planning problem is to find a schedule specifying the sizes, sequences and timing of the production lots on the different production lines. The overall costs are to be minimized. The demands of the different setup families are to be satisfied. And the limited capacities of the production lines are to be respected. Note that there is a strong interdependence between the lotsizing and the sequencing (or – in general – scheduling) part of the problem. To calculate feasible lotsizes, the capacity has to be respected. Because of the sequence-dependency of setup times, the sequence of the lots influences the available net capacity. However, deciding about the sizes of the lots also means deciding about the number of the setups. The number of setups again influences the available net capacities. Because of this crucial interrelationship, both the sizing and the scheduling of the production lots should be done simultaneously.

In the meantime, there is a rich body of literature on this so-called "Simultaneous Lotsizing and Scheduling" (SLS). General reviews on SLS are given by [15, 45]. In contrast, the reviews of [31, 28, 40, 9] rather concentrate on the lotsizing part of the problem and not so much on the integration of both aspects. Because of its flexibility and because of its quite general applicability for practical problems (see Sect. 5.3), in the following we focus on the General Lotsizing and Scheduling Problem for Parallel production Lines (GLSPPL, [38]). A solution heuristic for the GLSPPL has also been proposed in [38]. However, its scalability for large problem instances is not really satisfying. As Sect. 2 will show, there was further research on extensions of the GLSPPL and on related SLS problems. However, there still seems to be a lack of and a need for better scalable solution methods for the original GLSPPL.

The aim of this paper is to introduce such new solution heuristics for the original version of the GLSPPL. The new heuristics will be better scalable to larger problems of practical relevance. The basic idea is to iteratively decompose the multi-line GLSPPL into a series of single-line problems. These single-line problems can quite efficiently be solved by a heuristic presented in [37]. Different approaches of decomposing the multi-line problem and of iterating between a modified multi-line master problem and the single-line subproblems are proposed. They are compared by means of practical problem instances of different types of industries. Examples are consumer goods production, health care industry, acrylic glass production and label printing.

The next section summarizes the literature on SLS models and methods for heterogeneous production lines and sequence-dependent setup times. Section 3 presents the original GLSPPL as proposed in [38]. In Section 4, the new solution methods are introduced. After describing the basic idea of the iterative procedure, a decomposition approach based on priority rules and another approach based on an aggregation of the multi-line problem are explained. Section 5 shows the results of the computational tests. Both medium-sized and large practical problem instances are considered.

2. Literature review

The GLSP for a single line was first introduced by [22]. It has been shown to be NP–complete for non-zero minimum lotsizes. The GLSP was expanded for sequence-dependent setup times (GLSPST) by [37]. There, also a new solution approach called "dual reoptimization" was introduced. The basic idea is to vary the setup sequences for the single-line GLSPST by using the local search meta-heuristic threshold accepting (see [16]). Candidate sequences are drawn at random from a neighborhood of the current sequence. Candidate sequences, which improve the current sequence, are always accepted. In order to escape from local optima, candidates, which are not worse than a certain threshold, are also accepted. Lowering the threshold ensures convergence of the algorithm. All other candidates are refused. The setup sequence only defines the setup costs of the candidate. However, in order to also calculate its minimum holding costs, an ordinary network flow problem (NFP) has to be solved to optimality. The idea of dual reoptimization is to accelerate this process. This is achieved by using a dual NFP algorithm and by reoptimizing the current solution instead of solving the candidate from scratch. Using the dual NFP algorithm, it is even possible to refuse inappropriate candidates without having to solve the NFP to optimality. This property will later on be referred to as "early refusal of unacceptable candidates". Because candidate sequences are drawn at random, different independent runs of the above local search may lead to different (locally, but hopefully often also globally optimal) solutions. Thus, [37] always executes 25 independent runs and selects their best solution as the final result. The corresponding heuristic has been denoted as TADR. It proved to be very efficient for the single-line GLSPST.

The single-line GLSPST has been expanded to the parallel-line GLSPPL by [36, 38] in 1999 and 2002, respectively. The principle of combining threshold accepting with dual reoptimization has also been adapted to this kind of problem in a heuristic procedure called TAPLS. However, in the case of heterogeneous parallel production lines, a generalized network flow problem with losses and gains (GNFP) has to be solved instead of an ordinary NFP. This proved to be less efficient than in the single-line case. Only medium-sized practical problems with 2 production lines, up to 8 periods and up to 19 products were solved in a reasonable amount of time. A potential reason for this inefficiency are numerical instabilities of the GNFP solver used. Nevertheless, TAPLS was competitive to the solution heuristic of Kang et al. [30]. Kang's method has been designed to heuristically solve a similarly comprehensive SLS model, which was inspired by [5]. However, Kang's heuristic does not consider setup times.

At this point in time, the only other models (according to the authors' knowledge) considering heterogeneous production lines and sequence-dependent setup times have been proposed by De Matta and Guignard [14] in 1994 and by Clark and Clark [10] in 2000. The former model resembles the quite restrictive "all-or-nothing assumption" of the Discrete Lotsizing and Scheduling Problem (DLSP, [21]). "All-or-nothing" means that either a given period's capacity has to be completely exhausted by (setup time and) production for a single product or it is not produced at all within this period. The authors exploit this special property within Lagrangian based solution heuristics to solve a real world problem in the tile industry. The latter model is more general concerning the periods' capacities. It allows backlogging, but setup costs are not considered. The solution heuristics presented are especially designed for rolling horizon planning. Thus they are only aiming at finding exact schedules for the first period(s) of the planning horizon. Later periods are just approximated. According to the authors [10, p. 2306], the computational results suggest that *"for medium to large problems, impracticable amounts of time will be spent just identifying a feasible solution"*.

Because of an intense research on SLS in the last decade, quite a few models and solution heuristics for problems similar to the GLSPPL type have been proposed in the meantime. "Similar" means that these are also respecting sequence-dependent setup times and heterogeneous lines on a single bottleneck stage.

Ghosh Dastidar and Nagi [25] use a CSLP-type formulation for a scheduling problem in an injection molding facility. Backlogging is allowed and setup resources can be scarce. The authors use a two-phase decomposition of so-called "work centers" to heuristically solve the problem. The single-line version of the Continuous Setup Lotsizing Problem (CSLP) originally goes back to [32]. CSLP-type models generally allow to produce at a maximum one product per period. As opposite to the DLSP, here the product needs not to completely exhaust the period's capacity.

Almeder and Almado-Lobo [3] also extend the GLSPPL for a scarce setup resource. They compare this formulation with a less general one, yet showing a better computational behavior than the GLSPPL. Solution heuristics are not proposed. Almada-Lobo et al. [1, 2] present mixed integer programming (MIP) formulations which are designed to especially meet the requirements of glass container production. The former paper also employs a CSLP-type formulation. It applies Lagrangian relaxation to solve a short-term SLS problem for parallel molding machines. Complementary, the latter one tackles the medium-term, multi-location, multi-objective SLS problem of scheduling parallel furnaces. Here, variable neighborhood search (VNS) is used to heuristically solve the problem.

Józefowska and Zimniak [29] extend the GLSPPL for multiple objectives and backlogging in order to support decision making at a plastic pipe producing plant in Poland. On the one hand, the solution space is restricted by means of expert knowledge. On the other hand, a multi-objective genetic algorithm is applied to find

Pareto-optimal solutions. Amorim et al. [4] also extend the GLSPPL for multiple objectives and use a genetic algorithm to evaluate the Pareto-optimal boundary. However, they especially focus on perishability issues in yogurt production/packaging. They thus integrate the block planning concept of [27] with the GLSPPL.

Motivated by a real case producing refractory bricks and inspired by previous work of [34, 13] in job shop environments, Mateus et al. [35] propose an iterative approach for the integration of capacitated lotsizing and sequence-dependent setup scheduling. Moving forward in time period by period, an aggregate lotsizing problem is solved to optimality first. This is followed by a one-period, parallel machine scheduling problem that is heuristically solved using a GRASP algorithm. To integrate both levels of planning, production lots have to be heuristically decomposed into three different types of jobs (either fulfilling the period's demand, holding stocks, or backlogging inventory).

Since most of the above mentioned papers consider applications in industry, it can be concluded that SLS for a single bottleneck stage seems to be of high practical relevance. Nevertheless, recent research also focuses on multi-stage SLS, which is important in case of shifting bottlenecks. Some of these models basically allow to treat the parallel-line models considered here as a single-stage special case. However, because of the significantly increased complexity, it cannot be expected that solution methods designed for multi-stage SLS models can compete with heuristics especially designed for the single-stage case. Thus, we only very briefly point to these multi-stage models (allowing heterogeneous, parallel lines per stage and sequence-dependent setup times). However, we do not discuss them in further detail: Meyr [39] extends the GLSPPL to the GLSP for Multiple production Stages (GLSPMS). [42] compare reformulations and valid inequalities for this model, whereas [41] propose a solution heuristic based on VNS and fix & optimize. Lang [33, Chap. 7.2] extends the GLSPMS for product substitution. Finally, a series of papers [19, 18, 43, 17] deals with a two-stage soft drink production system (liquid preparation and bottling). This production system is represented by two- and single-stage SLS models and solution methods, respectively.

Summing up, the GLSPPL is well-accepted in the scientific literature and of high practical relevance. Models and methods for specialized and extended problem variants do exist. Nevertheless, there is still a need for scalable solution heuristics for the basic GLSPPL. Even though there exist reformulations of the GLSPPL, which are better suited for applying a standard MIP solver (see, e.g., [3]), we now present the original model formulation of the GLSPPL according to [38] because it seems more flexible to model further extensions required by new practical applications.

3. Model formulation

In the GLSPPL several products $j = 1, \dots, J$ are to be scheduled on $l = 1, \dots, L$ parallel production lines. The finite planning horizon T consists of discrete *macro periods* $t = 1, \dots, T$ (for example, months or weeks) with a given length. For each line l, each macro period t is additionally represented by a set S_{lt} of non-overlapping *micro periods* s where the number $\mid S_{lt} \mid$ of micro periods has to be fixed in advance. Only a single product can be produced within a single micro period. As opposite to macro periods, the lengths of the micro periods are decision variables and expressed by the (unknown) quantity x_{ljs} produced within a respective micro period s on the corresponding line l times a (known) production coefficient a_{lj}. All micro periods of a line l are arranged in the sequence $s = 1, \ldots, \sum_t | S_{lt} |$.

If on a certain line the same item is produced within one or several consecutive micro periods, these micro periods constitute a *lot*. The total quantity produced during these micro periods defines the *size of the lot*. Thus, a lot may even span over several macro periods. Since the starting times of macro periods and the sequence of micro periods are known, the sequence of lots is also known and a complete production schedule for all lines can be derived.

The following notation is used to formulate the problem:

Data:

S_{lt} set of micro periods s belonging to macro period t and production line l

K_{lt} capacity (time) of 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 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:

$I_{jt} \geq 0$ inventory of product j at the end of macro period t (units)

$x_{ljs} \geq 0$ 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)

$z_{lijs} \geq 0$ 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)

Using this notation the *General Lotsizing and Scheduling Problem for Parallel production Lines* (GLSPPL) can be stated as:

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

subject to

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

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

$$x_{ljs} \leq \frac{K_{lt}}{a_{lj}} y_{ljs} \qquad \forall l,\ j,\ s,\ t \text{ with } s \in S_{lt} \tag{4}$$

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

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

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

The objective is to minimize inventory holding costs, the sequence-dependent setup costs and line specific production costs of potentially heterogeneous production lines (1).

Because of (2) and $I_{jt} \geq 0$ inventory holding costs are accounted for and demand is met without back–logging. Sequence-dependent setup times may further reduce the limited capacity of the production lines (3). Constraints (6) and (4) ensure that a unique setup state y_{ljs} is defined per line and micro period and that production can only take place if a line is indeed set up for the respective product. If the setup state of line l has to change from product i to *another* product j in a micro period s, a *changeover* has to be executed and the corresponding changeover indicator z_{lijs} has to be set to 1 (7). This allows modeling of sequence-dependent setup costs and setup times, whereas constraints (5) allow the modeling of minimum lotsizes, which may sometimes be necessary for technical reasons.

Note that micro periods with zero length (so-called "idle periods") can occur. Further note that the above formulation of the GLSPPL is only appropriate for practical applications where the setup state is "conserved". That means, after periods without production, the same product can be produced again without incurring further setup efforts ($st_{ljj} = s_{ljj} = 0 \; \forall \; l, j$). In practical applications, where the setup state "gets lost" when a production line is out of use for a certain amount of time, the above formulation has to be adapted. A loss of setup state can, for example, be modeled by introducing a "neutral" setup state $j = 0$ and substituting the inequalities (3) with $\sum_{j \geq 0, s \in S_{lt}} a_{lj} x_{ljs} = K_{lt} - \sum_{i \geq 0, j \geq 0, s \in S_{lt}} st_{lij} z_{lijs} \; \forall \; l, t$ (see [22] for further details).

4. Solution methods

As demonstrated in [38], the GLSPPL can be solved by combining the local search heuristic threshold accepting (TA) with dual reoptimization. In order to early refuse unacceptable candidates the embedded GNFPs are solved by the "relaxation algorithm" of [7], which is specialized for this kind of problems. However, Meyr [38] already noted that the relaxation algorithm sometimes needs "an excessive number of iterations" and thus proposed some means to accelerate his heuristic solution procedures. One representative of these heuristics, which has been called "TAPLS", will serve as a benchmark in the following.

[8] and [44] made proposals to improve the numerical performance of the relaxation algorithm [7]. However, instead of reimplementing TAPLS with respect to these insights, we preferred to embed an ordinary dual Linear Programming (LP) solver into the same threshold accepting framework. The reasons for this are threefold: in the meantime, almost all commercial and non-commercial LP solvers offer a dual simplex variant and give access to objective function values of basis solutions, what is a prerequisite for combining TA with dual reoptimization (but was not state of the art ten years ago). The programmers of these solvers already spent a lot of effort in numerical tractability, so taking care about numerical problems is shifted to the external solvers. Second, solving an LP instead of a GNFP facilitates future extensions of the embedded problem, e.g., for time-indexed holding costs, backlogging, or overtime. Of course, it has to be expected that computation times are worse than they are when using a specialized GNFP algorithm. However, here we deliberately face the consequences. We successfully tested the LP solvers of the COIN-OR LP

project (CLP, [11]), the Gnu Linear Programming Kit (GLPK, [26]), and the FICO Xpress Optimization Suite [20] for combing them with dual reoptimization. Despite of the (especially for large problem instances) performance advantages of Xpress, we chose to apply GLPK because of its more liberal licensing policy and thus simpler transferability.

Therefore, a solution heuristic called "TA-GLPK" will serve as a further benchmark algorithm for the tests of Sect. 5. It is a reimplementation of TAPLS [38], which uses GLPK for the dual reoptimization of the embedded LPs instead of the relaxation algorithm [7] to solve the embedded GNFPs. The interested reader may note that the $parm.meth = GLP\_DUAL$ option of GLPK advices the GLPK LP solver to apply a dual simplex algorithm and that initializing the $parm.obj\_ul$ option of GLPK with the current threshold allows an early refusal of unacceptable candidates.

Nevertheless, it still has to be expected that these solution procedures will reach their limits if large problem instances are to be solved. Thus, the following sections demonstrate another solution approach, which aims at decomposing the overall GLSPPL problem into smaller subproblems, solving the decomposed subproblems individually and deriving solutions for the original problem from them. Basically, decomposition approaches are nothing new. Well-known examples are Benders' decomposition [6] or the Danzig–Wolfe decomposition [12], to name only a few. However, we want to introduce another approach, especially designed for the GLSPPL. It relaxes the inventory balancing constraints (2) – the only ones linking the parallel production lines together – in order to take advantage of the fact that with TADR a very fast and successful solution heuristic for the single-line special case GLSPST already exists.

The following Sect. 4.1 presents the framework of the overall approach. Based on this, two new solution heuristics, TA-Prio and TA-Agg, will be introduced in Sects. 4.2 and 4.3. The first one decomposes the multi-line problem GLSPPL into several individual single-line problems by means of rather simple priority rules. The latter one is an aggregation-based decomposition approach, first aggregating the original problem into a smaller, easier tractable counterpart and then disaggregating its solution again to derive a decomposition of the original problem.

4.1. A framework for decomposition

Figure 1 shows an overview of the decomposition approach. The original GLSPPL problem to be solved is denoted as PL/O. As part of the initialization, the original data of PL/O are copied to a second version PL/M. While PL/O will remain unchanged, its copy PL/M will be worked on and possibly be modified during the following proceeding. The decomposition approach is an iterative procedure. It stops when either a feasible solution has been reached in an iteration i or when a pre-defined iteration limit max has been reached.

Step 1 of the iteration is to decompose PL/M into L independent single-line problems $SL_1, \ldots, SL_L$. This is done by deriving line-specific demands d_{ljt} and line-specific initial inventories I_{lj0} from the line-independent demand data d_{jt} and initial inventories I_{j0} of PL/O and from the capacities K_{lt} of the current PL/M. Many different ways are conceivable to reach a *consistent* decomposition, fulfilling $\sum_l d_{ljt} = d_{jt}$ for all j, t and $\sum_l I_{lj0} = I_{j0}$ for all j. Sections 4.2 and 4.3 will present two of them. The art is to find decompositions which furthermore are *feasible* or even *perfect*. Feasible means that solutions, which are feasible with respect to the

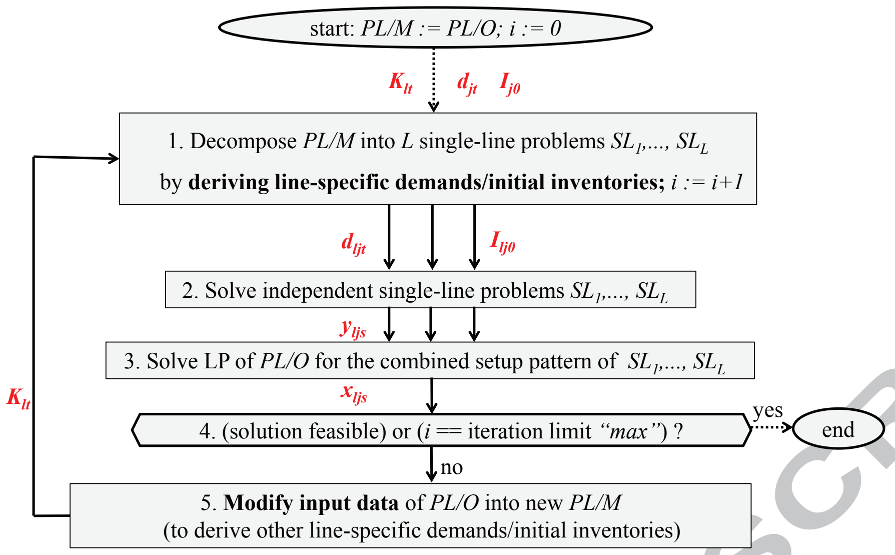

start: PL/M := PL/O; i := 0
K_{lt} d_{jt} I_{j0}
1. Decompose PL/M into L single-line problems SL_1,..., SL_L by deriving line-specific demands/initial inventories; i := i+1
d_{ljt} I_{lj0}
2. Solve independent single-line problems SL_1,..., SL_L
y_{ljs}
3. Solve LP of PL/O for the combined setup pattern of SL_1,..., SL_L
x_{ljs}
4. (solution feasible) or (i == iteration limit "max") ?
yes
end
no
5. Modify input data of PL/O into new PL/M (to derive other line-specific demands/initial inventories)
K_{lt}

Figure 1: A framework for decomposition

capacities K_{lt} of the original problem PL/O, can be found for all single-line problems SL_l. Perfect means that the optimal solutions of the $l = 1, \ldots, L$ independent SL_l together would even constitute the optimal solution of PL/O.

Each of the single-line problems SL_l is of a GLSPST type [37]. They can be solved with solution methods like TADR that are especially tailored to the GLSPST. However, since the GLSPST is a special case of the GLSPPL for $L = 1$, they also could be solved with GLSPPL heuristics like TAPLS or TA-GLPK. In the experiments of Sect. 5 and in step 2 of Fig. 1, TADR is applied because it has proven to be a fast and reliable solution method for the GLSPST [37]. Thus, the results of step 2 are setup patterns y_{ljs} for each single-line problem SL_l. Note that these setup patterns even exist if the decomposition was infeasible because TADR allows violations of the capacity constraints, but punishes them by means of penalty costs [37].

In step 3 the original problem PL/O is solved for the setup patterns y_{ljs} of step 2 being fixed. This means that an LP has to be solved what can, for example, be done by using GLPK. This way of computing production quantities x_{ljs} (and I_{jt}) is preferable to using the individual production plans of the SL_l directly. The reason is that – although the setup patterns are fixed – it might make sense to shift production quantities between production lines with respect to the original capacities of PL/O and the overall objective of the original problem PL/O (considering *all* production lines *simultaneously*).

The production plan x_{ljs}, being input to step 4 of Fig. 1, might be feasible for PL/O or not. If it is, the procedure terminates with a feasible solution. If it is not, but the iteration limit *max* has been reached ($i == max$), the procedure terminates without having found a solution.

Otherwise, the input data of PL/M are slightly modified, hoping to find a better decomposition this way and thus to end in a feasible solution in a further iteration $i+1$ of the overall procedure. This is illustrated by step 5 of Fig. 1. As can be seen, the original capacities of PL/O are transformed to get new, modified capacities K_{lt} for PL/M. The idea is to reserve capacity for the bottleneck lines of the last iteration i and thus to come up with a new decomposition in step 1 of the next iteration $i+1$, which is then feasible for step 2. To explain this effect in more detail: If the last decomposition was not able to allow finding feasible solutions in steps 2 and 3, the line-specific demand $\sum_j d_{ljt}$ of iteration i was higher than the original capacity of PL/O in some bottleneck period(s) and on some bottleneck line(s). If line-specific demand were shifted to earlier[1] non-bottleneck periods or to other non-bottleneck lines with unused capacity, TADR might find feasible solutions in step 2 of the next iteration $i+1$. This shift of demand has to be executed in the decomposition step 1 of iteration $i+1$. To motivate the decomposition algorithm to put such a shift into practice, the modified capacity K_{lt} of PL/M pretends that less capacity is available than it was in/on bottleneck periods/lines of the infeasible solution of the last iteration i. Thus "reserving capacity" actually means "reducing capacity", i.e., pretending that there is less capacity in order to force the decomposition of the next iteration to cleverly react by shifting line-specific demand. Note that such anticipation mechanisms are quite common in hierarchical production and supply chain planning (see, e.g., [24, p. 97]).

Two methods of modifying the capacity have been implemented and are tested in Sect. 5. The first one applies a fixed capacity reduction and is denoted as "fix" in the following. For this purpose, a capacity reduction factor $0 < redfac < 1$ has to be fixed in advance. The new capacity of the modified PL/M is then computed according to $K_{lt} := redfac^i \cdot K_{lt}\ \forall l, t$. Since the capacity is reduced in all periods and on all lines, implicitly also the bottlenecks of the last iteration are considered. Of course, this "flat" reduction of capacity – fix for all periods and all production lines – appears to be rather dumb because it does not focus specifically on the characteristics of the last iteration's solutions. Furthermore, it is not obvious how $redfac$ should be chosen.

Thus, the second, more intelligent method "var" reduces the capacity variably, i.e., dependent on the solution x_{ljs} of the last iteration i. It does not need a parameter to be fixed a priori. Its mode of operation is illustrated by Alg. 1. The idea is to first calculate the shortages of the production plan of iteration i forward in time (lines 3–3 of Alg. 1) and then to reduce capacity for these missing quantities backward in time (l. 17–29). If a product j is short in macro period t which has not been produced in this period at all ($\sum_{l,s\in S_{lt}} x_{ljs} = 0$), additional time $MaxSt_{jt}$ for a future setup has to be reserved, too (l. 22). Since the new setup sequence is not yet known, the worst case is assumed and the maximum setup time for changing to product j is retained (l. 10–16).

Note that this kind of feedback of the last iteration i is crucial for decomposition approaches. Established mathematical approaches like Danzig–Wolfe decomposition typically decompose the original problem into a master problem and one or several easy to solve subordinate problem(s). The optimal solutions of these subordinate problems are supposed to deliver helpful feedback for the next iteration's master problem. Unfortunately, this is not really promising in our case because already the GLSPST is difficult to solve so that it cannot be expected to find optimal solutions for the single-line problems. Nevertheless, the procedure "var" at least tries to

[1]Note that backlogging was not allowed in Sect. 3.

Algorithm 1: Variable ("var") calculation of reduced capacity

```
main input: $d_{ljt}$, $I_{lj0}$, $x_{ljs}$, original $K_{lt}$
output    : modified $K_{lt}$
1  forall ( do
       //
2  end
3  *[f]forward calculation of shortages)$l, j, t = 1, \dots, T$ $I_{ljt} := I_{lj,t-1} + \sum_{s \in S_t} x_{ljs} - d_{ljt}$;
4  if $I_{ljt} < 0$ then
5      $SQ_{ljt} := -I_{ljt}$ //*
6      [r]Short Quantity $I_{ljt} := 0$;
7  else
8      $SQ_{ljt} := 0$;
9  end
10 forall $j, t$ do
11     $MaxSt_{jt} := 0$;
12     if ( then
           //
13     end
14     *[f]if $j$ is not produced at all in $t$)$\sum_{l, s \in S_{lt}} x_{ljs} = 0$ and $\sum_l SQ_{ljt} > 0$ $MaxSt_{jt} :=$
         $\max_{l,i} \{st_{lij}\}$ //*
15     [f]initialization of Maximum Setup time
16 end
17 forall $l$ do
18     $CR := 0$;
19     forall ( do
           //
20     end
21     *[f]backward calculation of reduced capacity)$t = T, \dots, 1$ $CR :=$
         $CR + \sum_j (a_{lj} \cdot SQ_{ljt} + MaxSt_{jt})$ //*
22     [r]Capacity Reduction if $CR > K_{lt}$ then
23         $CR := CR - K_{lt}$;
24         $K_{lt} := 0$;
25     else
26         $K_{lt} := K_{lt} - CR$;
27         $CR := 0$;
28     end
29 end
```

mimic these ideas by using information about drawbacks of current solutions of the subordinate problems SL_l to send feedback to the next iteration's "master problem" PL/M. On the other hand, the master problem coordinates several subordinate problems by its way of decomposing d_{jt} and I_{j0} into d_{ljt} and I_{lj0}.

Of course, also other ways of sending feedback to step 1 of Fig. 1 via PL/M are conceivable. For example, time-indexed production coefficients a_{ljt} might be introduced and increased instead of decreasing capacities K_{lt}. This would allow to adapt the capacity consumption of each product j individually.

4.2. Priority-based decomposition

There are several possibilities to decompose a multi-line PL/M into several independent single-line problems SL_l as requested by step 1 of Fig. 1. In practical applications priority rules are quite popular. They define how to dedicate products to lines and vice versa, i.e., which product should preferably be produced on which line or which line should at best exclusively be used for which product. Common rules are to produce a product on as few lines as possible, but also to produce as much of a product as possible on its most suitable line in order to avoid setups and to save production costs. Unfortunately, this neglects that producing a single product on several production lines in parallel can save holding costs. Further problems arise if capacity is tight so that products cannot be produced on their preferable lines or a product's demand has to be split and distributed over several lines.

Algorithm 2 tries to resemble these ideas in order to serve as a benchmark for testing the less intuitive, but more sophisticated aggregation-based decomposition approach of Sect. 4.3. The basic idea is to start with products that can only be produced on a single line. If many of them exist, the one with the highest demand should be chosen first in order to allow a high utilization or even exclusive use of the line to be selected (see lines 1–2 of Alg. 2). Within all lines being able to produce this product j (i.e., $a_{\cdot j} > 0$), the line l with the cheapest relative production costs $\frac{c_{lj}}{a_{lj}}$ is most attractive (see l. 3). If the capacity K_{lt} of the line is sufficient to completely

Algorithm 2: Priority-based decomposition

```
main input: original $d_{jt}$, $I_{j0}$, $K_{lt}$
output    : line-specific demand $d_{ljt}$ and initial inventory $I_{lj0}$
1  $NL_j$ := number of lines with $a_{lj} > 0$;
2  forall $j$ in sequence of ascending $NL_j$, descending $\sum_t d_{jt}$ do

       //calculation of line-specific demand:
3      forall $l$ with $a_{lj} > 0$ in sequence of ascending $\frac{c_{lj}}{a_{lj}}$ do
4          forall $t$ do
5              if $K_{lt} \geq d_{jt}a_{lj}$ then
6                  $d_{ljt} := d_{jt};\ K_{lt} := K_{lt} - d_{jt}a_{lj};\ d_{jt} := 0;$
7              else
8                  $d_{ljt} := \frac{K_{lt}}{a_{lj}};\ d_{jt} := d_{jt} - \frac{K_{lt}}{a_{lj}};\ K_{lt} := 0;$
9              end
10         end
11     end
12     forall ( do
           //
13     end
14     *[f]assign remaining demand to line 1)$t$ with $d_{jt} > 0$ $d_{1jt} := d_{1jt} + d_{jt};$
       //calculation of line-specific initial inventory:
15     forall $l$ with $a_{lj} > 0$ and $\sum_t d_{ljt} > 0$ in sequence of descending $\frac{c_{lj}}{a_{lj}}$ do
16         if $\sum_t d_{ljt} \geq I_{j0}$ then
17             $I_{lj0} := I_{j0};\ \rightarrow$ continue with next $j$;
18         else
19             (
20         end
           //*
21         [f]single-line problem for $j$ trivial) $I_{lj0} := \sum_t d_{ljt};\quad I_{j0} := I_{j0} - \sum_t d_{ljt};$
22     end
23     if ( then
           //
24     end
25     *[f]assign remaining inventory to line 1)$I_{j0} > 0$ $I_{1jt} := I_{1jt} + I_{j0};$
26 end
```

fulfill the product's demand d_{jt}, the total demand will be assigned to the single-line problem SL_l via d_{ljt} (l. 5–6). Otherwise, only the share of d_{jt} that can still be produced on line l within the corresponding macro period t will be used (l. 8). This procedure is iterated over all lines and macro periods. Since it is of a heuristic and "greedy" nature, some unassigned demand may be left over at the very end. It should also be represented in the single-line decomposition. For ease of simplicity, this is completely assigned to the first line $l = 1$, unfortunately possibly causing that the corresponding single-line problem will become infeasible (l. 14–14).

An analogous procedure is necessary to decompose potential initial inventories I_{j0} into line-specific initial inventories I_{lj0}. Because initial inventories avoid production costs, here, we prefer production lines with high relative production costs (l. 15). Additionally, we only consider lines that are used according to our previous assignment, i.e., that show a positive line-specific demand $\sum_t d_{ljt} > 0$. If this demand already exceeds the initial inventory, the product's whole initial inventory is assigned to the line (l. 16–17). Otherwise, only the required share is used, so that this product does not need to be produced on the line at all (l. 21). This procedure is iterated over all lines. Again, some initial inventory may remain left over because of an inappropriate heuristic decomposition of the d_{jt} into d_{ljt}. Thus the remaining initial inventory is also reserved for the first production line $l = 1$ (l. 25–25).

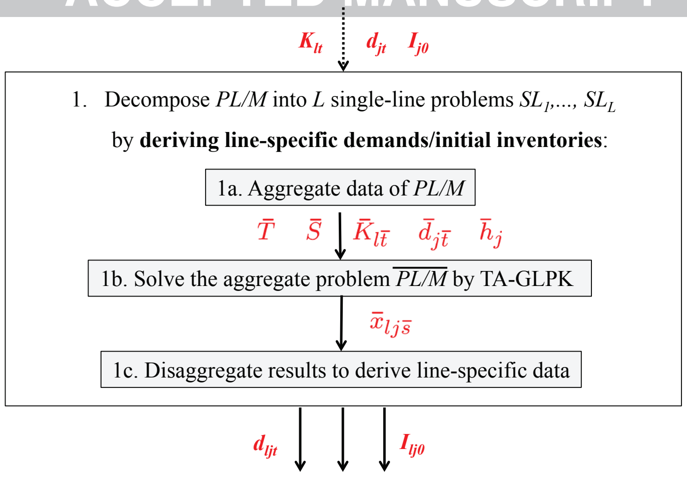
Please note that Alg. 2 is just a simple, illustrative example of a decomposition using priority rules. Of course, one can imagine other – and maybe better – rules and priorities. Nevertheless, Alg. 2 can give a fair impression of general advantages and disadvantages of this kind of approach.

4.3. Aggregation-based decomposition

An alternative way of deriving line-specific demands and initial inventories (see step 1 of Fig. 2) is to aggregate the detailed, large problem PL/M into a smaller $\overline{PL/M}$ (step 1a), solve the smaller one (step 1b) and disaggregate the results again to achieve d_{ljt} and I_{lj0} (step 1c). There are different ways of aggregation possible. In practice, an aggregation according to the product structure is very common because often final items can easily be aggregated to setup families. Then only setup families are considered for lotsizing. Indeed, for most of the practical problems that will be discussed in Sect. 5.3, this has been done beforehand. So there does not seem to be much potential left for a further aggregation according to the product structure. Instead, in the following we will explore an aggregation by time, where many smaller time buckets are aggregated to a few larger ones. This will be discussed in more detail in Sect. 4.3.1. Also different ways to disaggregate the results of the optimized (see Sect. 4.3.2) $\overline{PL/M}$ are possible. For example, an LP or NFP could be solved as proposed by [36, Chap. 7.2.2.2] for a product aggregation. However, Sect. 4.3.3 will show that this is not necessary in our case because for aggregation by time, a rather simple, rule-based approach is sufficient.

4.3.1. Aggregation by time

For ease of simplicity we assume that an integer aggregation factor f can be defined so that the original planning horizon T is an integer multiple of the new planning horizon $\bar{T}$ of the aggregated problem, consisting of fewer time buckets of longer, but equal length ($\bar{T} \cdot f = T$). For example, a detailed planning problem PL/M covering $T = 8$ weeks could be aggregated to a planning problem $\overline{PL/M}$ consisting of $\bar{T} = 2$ months by an aggregation factor $f = 4$. This sounds restrictive



K_{lt} d_{jt} I_{j0}
1. Decompose PL/M into L single-line problems SL_1,..., SL_L by **deriving line-specific demands/initial inventories**:
1a. Aggregate data of PL/M
$\bar{T}$ $\bar{S}$ $\bar{K}_{l\bar{t}}$ $\bar{d}_{j\bar{t}}$ $\bar{h}_j$
1b. Solve the aggregate problem $\overline{PL/M}$ by TA-GLPK
$\bar{x}_{lj\bar{s}}$
1c. Disaggregate results to derive line-specific data
d_{ljt} I_{lj0}

Figure 2: Aggregation-based decomposition

at the first moment. However, on the one hand, the original planning horizon could easily be extended to such an integer multiple by simply copying all data of the last macro period, e.g., if $T = 7$, we could first copy the last macro period once and then apply the aggregation. On the other hand, as we will see later on, time aggregation would also work for an arbitrary mapping of many smaller periods to a few longer ones if time-indexed holding costs h_{jt} can be respected when solving the aggregate problem $\overline{PL/M}$.

Only data that depend on (the lengths of) the time buckets of the detailed problem PL/M have to be adapted. Thus, capacities K_{lt}, demands d_{jt}, and holding costs h_j have to be summed to their aggregate counterparts $\bar{K}_{l\bar{t}}$, $\bar{d}_{j\bar{t}}$, and $\bar{h}_j$ to get the smaller problem $\overline{PL/M}$. For example, the capacities are aggregated by

$$\bar{K}_{l\bar{t}} := \sum_{t=(\bar{t}-1)\cdot f+1}^{\bar{t}\cdot f} K_{lt} \quad \forall l, \bar{t} = 1, \ldots, \bar{T}. \tag{8}$$

The demands have to be changed accordingly. The holding costs can simply be adapted to the new time pattern by $\bar{h}_j := f \cdot h_j$ because they account for longer macro periods now.

Last, the micro period related data have to be considered. Assuming that the number of micro periods per macro period was constant ($| S_{lt} | \equiv C \; \forall l, t = 1, \ldots, T$), the aggregate macro periods should comprise the same constant number of micro periods per aggregate macro period, i.e., $| \bar{S}_{l\bar{t}} | \equiv C \; \forall l, \bar{t} = 1, \ldots, \bar{T}$, in order to reduce the problem size of $\overline{PL/M}$. Thus, the total number $\bar{S}$ of micro periods of $\overline{PL/M}$ is calculated by $\bar{S} := \bar{T} \cdot C$.

4.3.2. Solving the aggregate problem

The aggregate problem $\overline{PL/M}$ is of the same type as the detailed problem PL/M, just smaller. Thus, all methods for solving the GLSPPL can be applied, hopefully leading to better and/or faster solutions because the aggregate problem is easier to solve. Accordingly, both solution methods TAPLS and TA-GLPK are available. For our experiments of Sect. 5 we will use TA-GLPK. The reasons are two-fold. First, its embedded LPs can more easily be adapted if extensions of the GLSPPL, e.g., for time-indexed production coefficients or time-indexed holding costs, become necessary. Second, as we will see later on, also its computational performance is preferable to TAPLS when instances exceeding a certain size have to be considered.

4.3.3. Disaggregation

After solving $\overline{PL/M}$ aggregate production quantities $\bar{x}_{lj\bar{s}}$ are available. They can be used to derive the line-specific (detailed) demands d_{ljt}. Since backlogging is not allowed in step 1b of Fig. 2 and in Sect. 4.3.2, an aggregate production quantity does only fulfill aggregate demand of its current aggregate macro period or later ones. Thus, it can also only satisfy detailed demand in the corresponding detailed periods. This is ensured by using a backward-oriented disaggregation procedure, where – moving backward in time – the latest aggregate production quantity is always assigned to latest remaining demand.

A small example shall illustrate this procedure: Let us assume original demand d_{jt} is given as $d_{11} = 10$, $d_{12} = 3$, $d_{13} = 7$, $d_{14} = 25$, $d_{15} = 10$ and $d_{16} = 5$ for a detailed problem instance with a single product $j = J = 1$, with $L = 2$ production lines and with $T = 6$ macro periods, consisting of one micro period each ($S = 6$). Using an aggregation factor $f = 3$, a feasible production plan $\bar{x}_{l1\bar{s}} = 15$ had been computed for $l = 1, 2$ and $\bar{s} = 1, 2$ in step 1b, which fulfills the aggregate demand $\bar{d}_{11} = 20$ and $\bar{d}_{12} = 40$ for $\bar{T} = \bar{S} = 2$. Then the latest production quantity $\bar{x}_{112} = 15$ of the first line $l = 1$ could satisfy the latest demands d_{16} and d_{15}, leading to $d_{116} = 5$ and $d_{115} = 10$. The latest production quantity $\bar{x}_{212} = 15$ of the second line $l = 2$ could lead to $d_{214} = 15$. The remaining 10 units of d_{14} would have to be satisfied by production of an earlier aggregate macro period. Using $\bar{x}_{111}$, an assignment $d_{114} = 10$ and $d_{113} = 5$ could result. The disaggregation procedure would end with $d_{213} = 2$, $d_{212} = 3$ and $d_{211} = 10$.

A more formal description of the disaggregation procedure is given by Alg. 3. The original demand d_{jT} of a product j in the last macro period T is looked at first (l. 2). It is tested whether it can be fulfilled by a production quantity $\bar{x}_{lj\bar{S}}$ of some line l in the last aggregate micro period $\bar{S}$. If this production quantity is sufficient, the whole demand will be assigned to line l and $\bar{x}_{lj\bar{S}}$ is set to 0 (l. 5–8). Otherwise, only the possible share of demand will be used, $\bar{x}_{lj\bar{S}}$ will be reduced accordingly (l. 10–12), and the procedure is repeated for the other production lines (l. 4). If afterwards still some remaining demand RD_{jT} of product j and macro period T is left, the lines are searched for in the preceding micro periods $\bar{S} - 1, \bar{S} - 2$, etc. (l. 3). After this, original demand $d_{j,T-1}$ of the preceding original macro period $T - 1$ needs to be checked (l. 2). However note that this will only take effect if d_{jT} has been completely fulfilled (i.e., $RD_{jT} = 0$) because, otherwise, all production quantities of product j would have been completely consumed ($\sum_{l,\bar{s}} \bar{x}_{lj\bar{s}} = 0$).

After executing Alg. 3, some demand $RD_{jt} > 0$ may be left over, which has been satisfied by initial inventories in the solution of the $\overline{PL/M}$. Since the initial inventories do not affect production capacities, they can be assigned to any production line. Without loss of generality, we use line $l = 1$ for this purpose and set $I_{lj0} := I_{j0}$ for $l = 1$ and $I_{lj0} := 0$ for $l > 1$. Note that the line-specific demands d_{1jt} have to be increased and that the RD_{jt} have to be decreased accordingly in periods t where $RD_{jt} > 0$.

If after this procedure still some remaining demand $RD_{jt} > 0$ exists, this indicates that the heuristic TA-GLPK has not been able to find a feasible solution for the aggregate problem $\overline{PL/M}$ and that these quantities RD_{jt} show the share of orig-

Algorithm 3: Aggregation-based decomposition

```
main input: original $d_{jt}$, $\bar{x}_{lj\bar{s}}$
output    : line-specific demand $d_{ljt}$
$RD_{jt} := d_{jt}\ \forall j, t = 1, \dots T$ //*
[r]Remaining Demand forall $j, t = T, \dots, 1$ with $RD_{jt} > 0$ do
    forall $\bar{s} = \bar{S}, \dots, 1$ do
        forall $l$ with $\bar{x}_{lj\bar{s}} > 0$ do
            if $\bar{x}_{lj\bar{s}} > RD_{jt}$ then
                $\bar{x}_{lj\bar{s}} := \bar{x}_{lj\bar{s}} - RD_{jt}$;
                $d_{ljt} := d_{ljt} + RD_{jt}$;
                $RD_{jt} := 0$;
            else
                $RD_{jt} := RD_{jt} - \bar{x}_{lj\bar{s}}$;
                $d_{ljt} := d_{ljt} + \bar{x}_{lj\bar{s}}$;
                $\bar{x}_{lj\bar{s}} := 0$;
            end
        end
    end
end
```

inal demand d_{jt} that got lost.[2] Nevertheless, a feasible solution might exist, which just has not been found due to the heuristic nature of TA-GLPK. The heuristics for the decomposed, detailed single-line problems might be more successful. Thus, this "lost demand" should also be assigned to the single-line problems, there increasing their line-specific demands d_{ljt}. The basic idea for this assignment is to distribute the lost demand of product j equally over all production lines l, which already produced it in the aggregate (infeasible) solution ($\sum_{\bar{s}>0} \bar{x}_{lj\bar{s}} > 0$). If a product j has not been produced at all ($\sum_{l,\bar{s}>0} \bar{x}_{lj\bar{s}} = 0$), its whole lost demand should be allocated to the cheapest production line, that would be able to produce it.

Finally note, if TA-GLPK instead of TADR were used to solve the single-line problems SL_l of step 2 in Fig. 1, also time-indexed holding cost h_{jt} could be respected. In this case, a user-defined mapping of smaller periods into longer ones – leading to time buckets of various lengths – would be possible. For example, a planning horizon of 12 weeks could be aggregated to a planning horizon $\bar{T} = 6$, consisting of 4 weeks and two months. This appears very interesting for practical applications, where often a rolling horizon planning of this type helps to mitigate forecast inaccuracies.

5. Computational results

In the following we will test the performance of the new algorithms TA-GLPK and the decomposition based approaches TA-Prio and TA-Agg as compared to the TAPLS of [38]. Section 5.1 is devoted to medium-sized practical problems that have also been introduced and used by [38]. Section 5.2 investigates the effects of varying the aggregation factor f and changing the parameters of the threshold accepting meta-heuristic. Finally, in Sect. 5.3 further large, practical problems are considered.

[2]Note that both solution procedures TAPLS and thus also TA-GLPK can generate such infeasible solutions because they relax the capacity constraints of the GLSPPL by means of penalty costs [38].

All computational tests have been executed on a personal computer using an Intel Core i7-860 2.8GHz QC CPU, 8 GB RAM, the Ubuntu 10.04 operating system, GLPK 4.44, and the gcc 4.4.3 compiler.

For TADR and TAPLS the threshold values and threshold accepting parameters of [37] and [38], respectively, are used. However, to limit computation times, for TADR the best solution out of 10 instead of 25 independent runs is chosen. Furthermore, for TAPLS the maximum number of candidate tests before changing the threshold value is replaced by a threshold multiplier TM, which can be varied in order to test different threshold settings. The threshold is lowered when $5 \cdot TM$ tests have not improved the current objective value. If the current solution has not changed within $2 \cdot TM$ steps, a threshold accepting run is stopped. These configurations can also be applied to the new algorithms of Sect. 4.

5.1. Performance for medium-sized practical problems

Within this section we consider twelve practical problems $T01, \ldots, T12$ of consumer goods industries, firstly presented in [38, Sect. 4.2]. They comprise $T = 8$ macro periods and $L = 2$ identical production lines each. The number of items J to be produced varies between 15 and 19. The problems are of "medium size" in the sense that they are not small enough for being solved to optimality by an up-to-date standard MIP solver in a reasonable amount of time. Thus, only objective function values of different heuristics – and especially the already known TAPLS of [38, Sect. 4] – will serve as a benchmark. For the following experiments, the threshold multiplier TM is set to 2000 (which is also a medium choice as we will see later on).

5.1.1. Comparing the different solution approaches

Table 1 compares the algorithms TAPLS of [38], its analog TA-GLPK using the dual simplex of GLPK for reoptimization, and the priority-based and decomposition-based methods TA-Prio and TA-agg4, respectively, for the problem instances $T01$, ..., $T12$. TA-agg4 applies an aggregation factor of $f := 4$ to condense the original 8-period problem into a 2-period problem. Both decomposition based methods allow a maximum of 2 iterations. They either use a fixed capacity reduction of 1 percent ("fix"), i.e., $redfac = 0.99$, or the variable capacity reduction policy ("var") as described in Sect. 4.1 if the first iteration did not lead to a feasible solution. The table shows the average percentage deviation of the objective values from the best solution found over all heuristics, for 120 runs of the respective heuristic, and the percentage of runs that did not lead to a feasible solution after the second iteration *(percentage of infeasible solutions)*. Note that due to the random elements of the threshold accepting implementation of [37, 38] different runs (with different seed values for the random number generator) typically create different heuristic solutions. A missing entry in the column of infeasible solutions means that all 120 runs were successful. For each problem instance the – with respect to solution quality – best performing solution method is marked bold.

As can be seen in the row "aver. obj", the priority based decomposition performs quite bad in terms of solution quality. If feasible solutions are found at all, they are on average more than 17 % away from the best solution found so far for a problem instance. However, since no linear program has to be solved for the aggregate problem, the computation times of 5 seconds on average (row "aver. cpu") are very favorable. The solution quality of TAPLS and TA-GLPK is with 7-7.5 % similar

Table 1: Average percentage deviation from the best objective value *(percentage of infeasible solutions)* for problems $T01, \ldots, T12$ and algorithms TAPLS, TA-Prio, TA-agg4, and TA-GLPK for a fixed (fix) or variable (var.) capacity reduction policy; average of the objective deviations (aver. obj) and average computation times (aver. cpu) in seconds over all problem instances.

	TAPLS	TA-Prio				TA-agg4				TA-GLPK
		fix		var.		fix		var.		
T01	6.4	13.4		12.3		**4.5**		**4.5**		6.6
T02	6.1	10.3		10.2		**3.4**		3.6		5.6
T03	6.9	13.7		13.7		3.1		**3.0**		6.6
T04	7.3	9.7		9.6		3.9		**3.8**		6.0
T05	9.0		*100*	44.4	*61*	**5.4**		5.5		7.1
T06	10.3	11.6		11.5		5.3	*27*	**5.1**	*28*	9.3
T07	9.0	40.4		40.4		6.2		**6.1**		7.7
T08	**8.2**	31.7		33.5		9.2		9.3		9.9
T09	**6.1**	23.3		23.4		7.0		6.6		7.0
T10	6.4	9.3		9.3		**4.0**		**4.0**		5.2
T11	6.9	17.2		17.3		6.3		**6.2**		8.0
T12	7.2	16.6		16.6		5.7		5.8		**5.4**
aver. obj	7.5	17.9		20.2		**5.3**		**5.3**		7.0
aver. cpu	196	5		5		28		28		95

on average. Astonishingly, TA-GLPK outperforms TAPLS in terms of computation time despite of its more general way of solving the embedded generalized network flow problems. The implementation of the relaxation algorithm [7] as used in [38] is not numerically stable enough to be competitive. Maybe the improved versions of [8, 44] would change the picture. However, looking at TA-agg4 this does not really seem to be necessary. On average both time decomposition variants create better solutions in a significantly shorter computation time than TAPLS and TA-GLPK.

The two different variants of capacity reduction (fix, var.) show similar results when applied to TA-agg4. For TA-Prio the fixed capacity reduction seems to be advantageous at first sight because of its lower average objective values. However, problem $T05$ cannot be solved at all when using a fixed reduction, whereas "only" 61% of these instances remain unsolved when using the variable capacity reduction. This phenomenon is further investigated in Tab. 2. Here the percentage share of infeasible solutions for a maximum number (max) of 1, 2 and 3 iterations are shown (see Sect. 4.1 and Fig. 1). The values for $max = 2$ correspond with the results of Tab. 1. On the one hand, it can be seen that this problem really seems to be hard for the fix-variant because even after the third iteration feasible solutions cannot be found. With respect to this, the variable capacity reduction appears superior to the fixed one. For this reason and because the fixed capacity reduction presumes that a reduction level has been determined somehow (1% capacity reduction in this example), whereas the variable reduction policy does not need such a parameter choice, the latter one will solely be used in the following experiments. On the other hand, Tab. 2 shows that a maximum number of two iterations ($max = 2$) seems to be a good choice. When using TA-Prio quite a lot of instances cannot be solved by just a single iteration. The second run – with a reduced capacity – leads to a feasible solution, however.

For all problem instances tested, the computation times of the decomposition approaches were favorable as compared to TAPLS and TA-GLPK. Unfortunately, Tab. 1 has shown that this does not hold true for every instance for the solution quality, even when only looking at the better solution method TA-agg4. For the problem instances $T08$ and $T09$, TAPLS performed best in terms of average solution

Table 2: Percentage share of infeasible solutions for a maximum number (max) of 1, 2 or 3 iterations, problems $T01, \dots, T12$, and algorithms TA-Prio and TA-agg4 with a fixed (fix) or variable (var.) capacity reduction policy.

	TA-Prio						TA-agg4					
	fix			var.			fix			var.		
max	1	*2*	3	1	*2*	3	1	*2*	3	1	*2*	3
T01	40			43								
T02	2			1								
T03	19			22								
T04	7			9								
T05	100	*100*	100	100	*61*	63	5			3		
T06	1			2			53	*27*	17	55	*28*	16
T07												
T08	100			100		1						
T09												
T10	1			2								
T11	1											
T12	1											

quality. Amongst them problem $T08$ seems to be especially difficult because of its worse percentage deviation from the best objective value and because the priority-based decomposition never found a feasible solution for this instance after the first iteration (see Tab. 2 for $max = 1$). Thus problem instance $T08$ will be investigated in more detail in the next section.

5.1.2. Objective value as a function of computation time

For problem instance $T08$ additionally 500 runs of the different solution methods have been executed. The results of these tests are shown in Fig. 3. In order to allow a better comparison of the solution methods, a different way of illustrating the results has been chosen. Figure 3 shows the improvement of a solution method's solution quality as a function of computation time – thus allowing to compare the two conflicting objectives "quality" and "time" simultaneously. For this purpose, again the average percentage deviation of a solution method's objective value from the best objective value found overall is measured. However, now this is done for "packages" of several runs of an algorithm that have been executed consecutively. The best objective value of any run within a package determines the objective value of the package as a whole. Whereas the sum of the computation times of the package's runs determines the overall computation time of the package. Packages of the same size (i.e., with the same number of runs) are averaged. Their average computation times [secs] constitute the x-values of Fig. 3, while the average deviations [%] define the y-values.

A more precise definition shall ease understanding: Let $k = 1, \dots, K$ denote several independent runs of a certain heuristic h that have been executed. Let rd_k^h denote the (percentage) deviation of the best objective function value that has been found by heuristic h within run k from the best solution found overall. Let rc_k^h denote the computation time of this run k of heuristic h. K times after each other, for $s = 1, \dots, K$, the overall sequence of K runs is partitioned into packages $n = 1, \dots, N(s)$ of different sizes s where $N(s) := \left\lfloor \frac{K}{s} \right\rfloor$. For example, all K runs are first partitioned into $n = 1, \dots, K$ packages of size $s = 1$ and later into a single package of size $s = K$. All other integer package sizes in between are subsequently handled, too. The objective value of package n for heuristic h is given by the deviation of the

Table 3: $K = 4$ runs, partitioned into four packages of size $s = 1$, two packages of size $s = 2$ and one package of sizes $s = 3$ and $s = 4$, respectively.

			$n = 1$		$n = 2$		$n = 3$		$n = 4$			
k, s	$\boldsymbol{rd_k}$	$\boldsymbol{rc_k}$	pd_1^s	pc_1^s	pd_2^s	pc_2^s	pd_3^s	pc_3^s	pd_4^s	pc_4^s	$\bar{pd}^s$	$\bar{pc}^s$
1	**6**	*1*	6	*1*	4	*3*	2	*2*	8	*2*	$\frac{6+4+2+8}{4}$	$\frac{8}{4}$
2	**4**	*3*	$\min\{6; 4\}$	*1+3*	$\min\{2; 8\}$	*2+2*					$\frac{4+2}{2}$	$\frac{4+4}{2}$
3	**2**	*2*	$\min\{6; 4; 2\}$	*6*							2	*6*
4	**8**	*2*	$\min\{6; 4; 2; 8\}$	*8*							2	*8*

best run within this package. For a partitioning into packages of size s its value $pd_n^{h,s}$ can be computed by $pd_n^{h,s} := \min_{k=(n-1)\cdot s+1}^{n\cdot s} \{rd_k^h\}$ for $n = 1, \dots, N(s)$. In contrary, the computation time of a package is given by the sum of the computation times of all runs within the package. Thus, for a partitioning into packages of size s, the computation time $pc_n^{h,s}$ of package n of the heuristic h can be computed according to $pc_n^{h,s} := \sum_{k=(n-1)\cdot s+1}^{n\cdot s} rc_k^h$ for $n = 1, \dots, N(s)$.

For each size s, the packages $n = 1, \dots, N(s)$ represent observations of $N(s)$ random processes. Therefore, we are interested in their average values. Thus, the average deviation $\bar{pd}^{h,s}$ of a package containing s runs of heuristic h is computed by $\bar{pd}^{h,s} := \frac{1}{N(s)} \sum_{n=1}^{N(s)} pd_n^{h,s}$ for $s = 1, \dots, K$. Analogically, the average computation time $\bar{pc}^{h,s}$ of a package containing s runs of heuristic h is calculated by $\bar{pc}^{h,s} := \frac{1}{N(s)} \sum_{n=1}^{N(s)} pc_n^{h,s}$ for $s = 1, \dots, K$. Table 3 shows an example for $K = 4$ runs, subsequently being partitioned into $N(1) = 4$ packages of size $s = 1$, into $N(2) = 2$ packages of size $s = 2$ and into $N(3) = N(4) = 1$ package of sizes $s = 3$ and $s = 4$, respectively. Note that the smaller the package size, the more observations are available. For this reason – and because computation times of packages should not grow too much – we are mainly interested in packages of small size. As already mentioned, [37] worked with packages of size $s = 25$ runs when applying TADR.

For $K = 500$, the values $(\bar{pc}^{h,s}, \bar{pd}^{h,s})$ mark the (x,y)-coordinates of the heuristics $h = TAPLS, \dots$ in Fig. 3, starting with $s = 1, 2, \dots$ from left to right. For ease

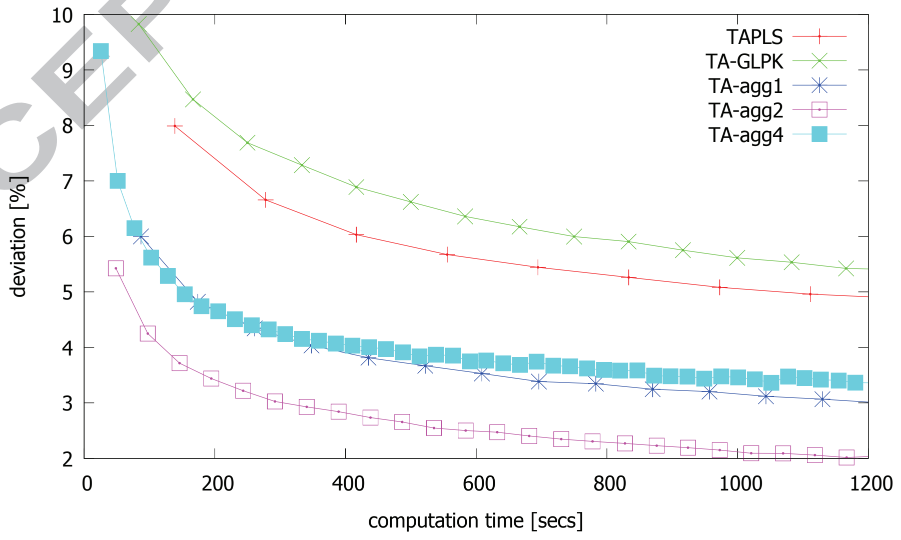


Figure 3: Percentage deviation from the best known objective value plotted as a function of computation time [secs] for the algorithms TAPLS, TA-GLPK, TA-agg1, TA-agg2, and TA-agg4 (problem $T08$, threshold multiplier 2000, variable capacity reduction).

of illustration, these coordinates have been connected by straight lines for each heuristic h. For example, the average computation time $\bar{pc}^{h,s}$ of a single run $s = 1$ of $h = TA - GLPK$ is about 84 seconds with an average objective deviation $\bar{pd}^{h,s}$ of 9.8 percent. The average time $\bar{pc}^{h,s}$ of a package of $s = 2$ runs is 167 seconds with an average objective deviation $\bar{pd}^{h,2}$ of 8.5 percent etc. The closer to the left bottom corner the graph (resulting from connecting these data points) is, the better the respective algorithm performs in terms of *both* solution quality *and* computation time.

Note that the results of Tab. 1 can be retrieved from the left hand extreme data points of each algorithm if the same data basis is used. However, here slight differences occur because for Fig. 3 a substantially higher number of runs had to be executed (leading to other averages).

Now it can be seen that TA-agg4 is not only preferable to TA-GLPK but also to TAPLS because for a certain computation time given it always leads to solutions of higher quality. The results of Tab. 1 were misleading because TAPLS is so slow that no objective values are available within the time necessary for a single run of TA-agg4. Around 5-6 runs of TA-agg4 can be executed during a single run of TAPLS, which together lead to clearly better objective values.

Additionally to TA-agg4 also results for aggregation factors $f = 2$ and $f = 1$ are shown in Fig. 3. An aggregation factor $f = 1$ not only means that first TA-GLPK is solved in order to assign items to production lines and afterwards the decomposed single-line problems are (re-)optimized by TADR. If this does not bring up a feasible solution, additionally the line assignment of TA-GLPK is executed a second time for a reduced capacity in bottleneck periods on bottleneck lines, etc. Of course, a better solution quality can be expected then. However computation times may also increase. Fig. 3 shows that TA-GLPK can clearly be improved using this strategy. However, a further improvement can be gained by choosing an aggregation factor of 2. TA-agg2 obviously outperforms all other solution methods for problem instance $T08$. For the above tests, a threshold multiplier $TM = 2000$ had been used. The next section will investigate whether the above findings also hold for other thresholds.

5.2. *The influence of aggregation factors, thresholds, and numbers of micro periods*

In order to check this, we first stay at the single “difficult” problem $T08$ and then verify whether these results also hold for the remaining medium-sized problem instances.

5.2.1. *Investigating the medium-sized problem* $T08$

Figure 4 shows TAPLS and TA-GLPK not only for the threshold multiplier $TM = 2000$, but also for a prolonged local search with $TM = 6000$. As can be seen, the performance of TAPLS is not very sensitive on the thresholds used. With TAPLS and $TM = 2000$ feasible solutions can obviously be found earlier. But for higher computation times thresholds of $TM = 6000$ lead to more or less the same objective level. This is different for TA-GLPK. Its (new) implementation of threshold accepting indeed depends on the thresholds chosen. A higher multiplier TM clearly not only leads to a better overall objective level, but also improves the solution quality for a certain computation time given.

This should be investigated more thoroughly for other threshold multipliers and our different alternatives of aggregating by time. Unfortunately, a graphical illus-

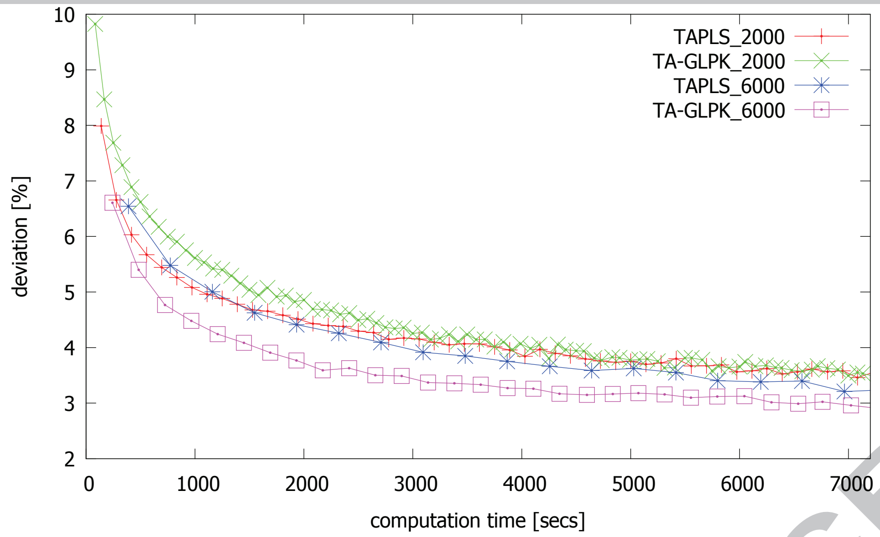


Figure 4: Percentage deviation from the best known objective value for threshold multipliers 2000 and 6000 and the algorithms TAPLS and TA-GLPK (problem $T08$).

tration of the solution performance as used in Figs. 3 and 4 quickly loses it clarity when the number of solution methods increases. Thus it is necessary to condense the information given by each graph (solution method) of the figures into a single number. For this, for each solution method the convex hull of its data points (average objective value and computation time per package size) is determined. Additionally, the convex hull of all data points of all solution methods to be compared is determined. The latter one expresses the "best" objective value that could be reached with any tested solution method for a certain computation time. Finally, for each solution method the average vertical distance between its convex hull and the "best" convex hull of all data points is measured and set into percentage relation to the best convex hull's objective level. The corresponding measure will be denoted as the "Average percentage Deviation of the best Objective Line (ADOL)" in the following. The upper left section of Tab. 4 shows the ADOLs for the solution methods that result from running TAPLS, TA-GLPK and TA-agg (aggregation factors 1, 2 and 4) with threshold multipliers $TM = 100, 2000$, and 6000 up to 7200 seconds. Additionally, in the upper right section the average computation times CPU1 [secs.] are shown that are necessary to find a first feasible solution with each combination of algorithm and threshold. They give an impression of the computational efforts that are necessary. (Please ignore the columns th for the moment.)

First we should check whether the ADOLs really express the intended information content. When comparing the column $TM = 2000$ of Tab. 4 with Fig. 3 it can be seen that TA-agg2 indeed performs best and TAPLS and TA-GLPK are clearly worst. However, looking at the ADOLs of TA-agg1 and TA-agg4, the ADOL of TA-agg1 seems better than Fig. 3 indicates on the first sight. Nevertheless, the ADOLs measure correctly because they represent the average deviation over the *longer* horizon of 7200 seconds also underlying Fig. 4. As Fig. 4 has further shown, the ADOLs of TAPLS for $TM = 2000$ and $TM = 6000$ have to be quite close to each other which is indeed the case (3.1 and 2.7, respectively). Finally, the larger gap between the corresponding ADOLs of TA-GLPK (3.8 and 2.4) again indicates the higher sensitivity of TA-GLPK with respect to a variation of threshold values.

Looking again at the upper left section of Tab. 4 one can see that TA-GLPK

Table 4: Average percentage deviation of the best objective line (ADOL) and average running time to compute a first feasible solution (CPU1) for different combinations of algorithms (TAPLS, ..., TA-agg4-s) and thresholds (TM = 100, ..., th) for problem instance $T08$.

	ADOL [%]				CPU1 [secs]			
	100	2000	6000	*th*	100	2000	6000	*th*
TAPLS	6.7	3.1	2.7		15	139	389	
TA-GLPK	16.3	3.8	2.4	*2.7*	8	84	242	*138*
TA-agg1	3.1	1.3	**1.1**	*1.0*	11	87	**246**	*144*
TA-agg2	**0.8**	**0.3**	**0.7**	***0.1***	**8**	**49**	**131**	*43*
TA-agg4	**1.1**	1.7	2.3	*1.4*	**6**	26	63	*15*
TA-GLPK-s	10.8	2.6	1.6	*2.1*	7	91	276	*160*
TA-agg1-s	1.5	0.8	**0.7**	*0.8*	9	96	**277**	*161*
TA-agg2-s	**0.5**	**0.2**	**0.4**	***0.3***	**5**	**49**	**141**	*44*
TA-agg4-s	**0.5**	1.9	2.6	*1.3*	**4**	25	62	*12*

only seems to be competitive to TAPLS if a high threshold multiplier is used. For $TM = 100$ and $TM = 2000$ TAPLS is favorable. However, time aggregation is always the best choice, i.e., for each aggregation factor and threshold multiplier the ADOLs of TA-agg are better than the corresponding TA-GLPK and TAPLS values. Among the different aggregation possibilities an aggregation factor of 2 performs best. There seems to be an interesting correlation between aggregation factors and threshold multipliers (bold diagonal values in Tab. 4 marking the five best combinations of TA-agg1, TA-agg2 or TA-agg4 with thresholds $TM = 100, 2000$ or 6000): the higher the aggregation is the smaller the threshold multipliers can and should be, respectively. The reason is probably that due to the aggregation error and corresponding loss of information it does not make sense to solve the aggregate problem too accurately. This computation time should better be invested to execute more "short runs" and draw advantage of the variance of solutions generated.

It would be tedious to determine a suitable threshold multiplier TM for every problem instance to be solved, separately, by running experiments on a discrete grid of potential multipliers. Instead it would be helpful to calculate an appropriate multiplier automatically from a problem instance's characteristics like the number of products J, number of production lines L, and number of macro periods T. In order to identify such a relation, some tests – for sake of brevity not further described here – have been executed for a selected set of test problems. The result was that a threshold multiplier TM should be determined according to the function $TM(J, L, T) := \max\{250;\ \max\{\lfloor f(T) \rfloor ; 1\} \cdot J \cdot T \cdot L\}$ with $f(t)$ being a polynomial of the form $f(t) = a \cdot t^2 + b \cdot t + c$. Its coefficients $a = -0.000744$, $b = -0.053571$, and $c = 12.476190$ have been calculated by solving the linear system of equations $y = f(a, b, c, x)$ for the three sample points $(x; y) = (8; 12), (32; 10)$, and $(64; 5)$ measured in the tests. The columns th of Tab. 4 show the results when TM has been calculated automatically according to this function. As can be seen this choice results in high quality solutions for problem instance $T08$ while the computation times for finding a first solution are still acceptable.

Whereas $\mid S_{lt} \mid$ had – as in [38] – been set to 20 for all problem instances until now in order to allow each product to be produced at least once per line, this was changed to $\max_l \left\lfloor 1 + \sum_{j \in \hat{J}_l} \frac{1}{|\hat{L}_j|} \right\rfloor$ if $L > 1$ and to J if $L = 1$ for all solution methods of Tab. 4 that end with "*-s*". Here $\hat{J}_l$ denotes the set of products that can be produced on line l and $\hat{L}_j$ denotes the set of lines, product j can be produced on. The idea is that in a "good solution" the products are more or less evenly spread over their parallel lines and only a few ones connect the lines in order to increase flexibility. Thus, also the number of micro periods per macro period can be set automatically now — for TA-GLPK as well as for the aggregate multi-line and decomposed single-line models of TA-agg. This leads almost always to an improvement of the ADOL. The only exceptions are TA-agg2 in the automatic version *th* and TA-agg4 in combination with $TM = 2000$ and $TM = 6000$, i.e., when both aggregation level and intensity of local search are highest. Interestingly, decreasing $\mid S_{lt} \mid$ by automatically setting it does not necessarily help to get a first solution faster. TA-agg2 and the aggregation factor 2 also perform best for the "-*s*" versions. Using $TM = 100$ it is possible to compute good solutions very quickly. But $TM = 2000$ and the automatic threshold *th* also allow very good solutions in an acceptable computation time for getting a first solution.

5.2.2. Investigating all medium-sized problem instances

Table 5 verifies whether the results of the single problem instance $T08$ can be generalized for all twelve medium-sized problem instances. It uses the measures introduced in Sect. 5.2.1 and in Table 4. However, the ADOLs and CPU1 computation times are now averaged over all twelve problem instances $T01, \ldots, T12$. At least 4500 runs for $TM = 100$, 500 runs for $TM = 2000$ and 200 runs for $TM = 6000$ have been executed per problem instance and heuristic.

Table 5: Average percentage deviation of the best objective line (ADOL) and average running time to compute a first feasible solution (CPU1) for different combinations of algorithms (TAPLS, ..., TA-agg4-s) and thresholds ($TM = 100$, ..., *th*) averaged over all medium-sized problem instances $T01, \ldots, T12$.

	ADOL [%]				CPU1 [secs]			
	100	2000	6000	*th*	100	2000	6000	*th*
TAPLS	9.3	3.6	3.0		33	200	520	
TA-GLPK	22.1	3.2	2.5	*2.8*	8	94	269	*154*
TA-agg1	6.4	1.2	1.3	*1.2*	12	98	276	*160*
TA-agg2	**1.0**	**0.8**	**1.2**	***0.7***	**9**	**55**	**145**	***48***
TA-agg4	**0.4**	**1.1**	1.5	*0.8*	**7**	**30**	79	*17*
TA-GLPK-s	10.6	2.1	1.6	*1.8*	7	99	278	*163*
TA-agg1-s	2.6	**0.8**	**0.9**	*0.8*	10	**102**	**287**	*165*
TA-agg2-s	**0.4**	**0.7**	1.1	***0.7***	**6**	**56**	155	***50***
TA-agg4-s	**0.1**	1.0	1.5	***0.7***	**4**	30	82	***15***

As can be seen most of the statements for $T08$ can be confirmed: TA-GLPK is only competitive to TAPLS if a high threshold multiplier is used, but time aggregation is always the better choice. The higher the aggregation is the smaller the threshold multipliers should be. When comparing the average performance over all threshold multipliers, the aggregation factor 2 still does best. The automatic choice of threshold multipliers in the *th* versions shows good results. The automatic choice of the number of micro periods per macro periods in the "-s" version leads to a better solution quality, but does not necessarily help to find a first solution earlier.

However, there is also an interesting difference: The overall best performance is achieved when combining the shortest threshold $TM = 100$ with the highest aggregation aggregation factor 4. Both TA-agg4-s and TA-agg4 not only perform best in terms of the ADOL (0.1 and 0.4 %, respectively), but also show very short computation times CPU1 to find a first feasible solution (4 and 7 seconds, respectively).

When looking at all problem instances the differences between the aggregation factors 2 and 4 are by far smaller than it was the case in Sect. 5.2.1 and Tab. 4. Since $T08$ can be assumed to be a rather "difficult" problem instance, the remaining rather "simple" instances obviously profit from a higher aggregation.

5.3. Performance for large practical problems

Finally we consider four "large" real world problems, $T13, \ldots, T16$, that arose from cooperations with practice during the past years. All of them deal with several heterogeneous, parallel production lines, significant sequence-dependent setup costs and setup times, and tight line capacities. The first instance $T13$ stems from producing incontinence pads in health care industry. $J = 10$ setup families are produced on $L = 3$ parallel production lines. Here, the GLSPPL is solved to support mid-term master planning [23, Chap. 3.2.3]. Thus the planning horizon consists of $T = 48$ weeks as macro periods. The instances $T14$ and $T15$ originate from acrylic glass production. They have been introduced by [36, chap. 7.2.2.4], comprising also $J = 3$ production lines, but $T = 12$ months each. In $T14$ the final items to be considered have been aggregated to $J = 12$ setup families. Since the setup times of one to two hours within the families are still rather high, solving the original (disaggregate) problem for the $J = 51$ individual final items is desired. Instance $T15$ represents this target. Last, $T16$ tackles the problem of printing labels that get stuck on consumer goods like food, cosmetics, or household detergents. Here $L = 7$ production lines are available to produce a total of $J = 72$ setup families. The planning horizon again consists of $T = 12$ months.

Table 6 shows the results when applying the most successful thresholds and algorithms of Sect. 5.2, i.e., TA-GLPK-s and TA-agg-s with aggregation factors 1, 2 and 4, to these instances. TAPLS leads to inferior results for these problem sizes and its computation times soon get prohibitive. Thus, TAPLS has not systematically been tested for $T13, \ldots, T16$ anymore. Nevertheless, sporadic tests have confirmed these insights.

Table 6: Average percentage deviation of the best objective line (ADOL) and average running time to compute a first feasible solution (CPU1) for different combinations of algorithms (TA-GLPK-s, ..., TA-agg4-s) and thresholds (TM = 100, ..., th). The tested problem instances $T13, ..., T16$ comprise L production lines, J products, and T macro periods.

					ADOL [%]				CPU1 [secs]			
	L	J	T		100	2000	6000	*th*	100	2000	6000	*th*
T13	3	10	48	TA-GLPK-s	69.8	4.4	1.7	***0.7***	21	188	560	***1101***
T13				TA-agg1-s	7.2	2.6	**0.4**	***0.0***	20	196	**567**	***1106***
T13				TA-agg2-s	2.8	2.1	4.0	*3.5*	14	88	244	*313*
T13				TA-agg4-s	4.0	5.6	6.6	*6.1*	11	42	107	*78*
T14	3	12	12	TA-GLPK-s	5.5	0.4	0.3	*0.3*	5	84	249	*211*
T14				TA-agg1-s	0.9	**0.0**	**0.1**	***0.0***	6	**88**	**256**	***214***
T14				TA-agg2-s	0.3	0.5	0.7	*0.5*	5	48	135	*62*
T14				TA-agg4-s	0.7	1.6	1.9	*1.3*	5	43	127	*30*
T15	3	51	12	TA-GLPK-s			1.1	***0.0***		>2h	2760	***4919***
T15				TA-agg1-s		**0.5**	1.4			**1207**	4102	*>2h*
T15				TA-agg2-s	4.9	**0.4**			4021	**1615**	>2h	*>2h*
T15				TA-agg4-s	0.7		2.1	*2.4*	796	>2h	6535	*6561*
T16	7	72	12	TA-GLPK-s		0.6	**0.2**			2072	**5373**	*>2h*
T16				TA-agg1-s		**0.1**	**0.0**		>2h	**2367**	**6277**	*>2h*
T16				TA-agg2-s	0.6	**0.2**			826	**5940**	>2h	*>2h*
T16				TA-agg4-s	0.4				583	>2h	>2h	

For each problem instance the – according to the ADOL – best three combinations of algorithms / thresholds and their respective computation times CPU1 are marked bold. If both ADOL and CPU1 do not contain a value, no feasible solution has been found at all for this combination. If CPU1 is marked with "$> 2h$" and ADOL does not contain an entry, feasible solutions have been found, but not within the time limit of 7200 seconds that has again been used to compute and fairly compare the ADOLs. Note that the computation times for the th-variants of instance $T16$ can grow up to more than one day in the extreme case. Therefore, an automatic computation of the thresholds according to Sect. 5.2 seems mainly helpful for instances up to the size of problems $T13$ and $T14$.

Looking at all problem instances, here TA-agg1-s performs best. For every problem instance at least one TA-agg1-s combination can be found within the best three ADOLs, for $T14$ even all three. The capacity reduction (if no feasible solution has been found during the first iteration of the "aggregate" problem) and the reoptimization of the single-line problems by TADR prove to be advantageous as compared to TA-GLPK-s. Unfortunately, computation times can get quite high if threshold multipliers $TM \geq 2000$ are used.

Higher aggregation with factors of 2 and 4 generates quite good values, again, if a small threshold multiplier $TM = 100$ is used. For the rather small instances $T13$ and $T14$ very short running times to compute a first feasible solution (< 20 seconds) can be realized this way. However, for the larger instances $T15$ and $T16$ the computation times of $TM = 100$ can become quite long because the share of infeasible solutions increases. Nevertheless, even for these very large instances an acceptable compromise between solution quality and computation time for a first solution can be found if TA-agg4-s is combined with $TM = 100$ or TA-agg1-s is combined with $TM = 2000$. In the first case a running time of less than 15 and in the second case of less than 40 minutes can be realized on average to get a first feasible solution. When considering the fact that the GLSPPL is intended for mid-term master planning instead of short-term operational scheduling in these practical applications, this still appears to be acceptable.

6. Summary and conclusions

A new solution method to the General Lotsizing and Scheduling Problem for Parallel production Lines (GLSPPL) has been introduced. The GLSPPL addresses the problem of simultaneously determining the sizes and schedules of production lots on non-identical, parallel production lines when sequence-dependent setup times reduce the limited capacities of the production lines, deterministic, dynamic demand has to be met without backlogging, and inventory holding, sequence-dependent setup and production costs are to be minimized. The novel solution method iteratively decomposes the multi-line "master problem" into a series of single-line subproblems, which can quite efficiently be solved by the solution heuristic TADR proposed in [37]. Two different methods for modifying the capacities of the multi-line master problem between two different iterations have been proposed. While the first one applies a fixed capacity reduction, the second one variably reacts on capacity overloads of the single-line production plans. Similarly, two different approaches – a priority-based and an aggregation-based approach – for decomposing the multi-line problem into single-line subproblems have been introduced. These different solution methods have been tested for varying parameter configurations and have been compared with

TAPLS, a competing solution heuristic for the GLSPPL proposed in [38].

The computational tests show the superiority of the decomposition-based approach. It is faster and generates higher-quality solutions than TAPLS. This especially holds true if the problem size increases. Using the decomposition it is possible to find feasible solutions for large practical problem instances that are not solvable with TAPLS in a reasonable computation time. In particular, a variable reduction of capacity and an aggregation-based decomposition can be recommended. The aggregation-based decomposition requires to solve an aggregate master GLSPPL of reduced size by means of the local search meta-heuristic threshold accepting. Then, the corresponding thresholds and the level of aggregation have to be aligned. Interestingly, "short" thresholds, allowing fast computation times, come along with high levels of aggregation and vice versa. A procedure to automatically balance this relationship has been proposed.

To be more specific: The shortest threshold (expressed by a low "threshold multiplier" $TM = 100$) and the most aggregating heuristic TA-agg4-s (showing an aggregation factor of 4) performed best on average when medium-sized problem instances were tested. This combination has also shown to be a good compromise between solution quality and computation time to find a first feasible solution for all other instances. However, an aggregation factor of 2 combined with a higher threshold multiplier $TM = 2000$ was able to further improve the solution quality for the rather difficult medium-sized problem instance $T08$. The same effect occurred when combining an aggregation factor of 1 with $TM = 2000$ for the even larger practical problem instances of Sect. 5.3. Thus it seems beneficial to lower the aggregation level and simultaneously increase the threshold multipliers if problems grow larger and get more difficult.

For the future, it appears promising to test other types of decomposition and aggregation. For example, production coefficients could be increased instead of reducing capacities when iterating between the multi-line master problem and the single-line subproblems. This would allow a more specific tuning of the modified master problem. Furthermore, a product-based aggregation instead of or additionally to the current time-based aggregation might be interesting. This would resemble the building of setup families, which is very common in practice, anyway. The decomposition-based approach is also open for further extensions of the GLSPPL. The embedding of general LPs into the dual reoptimization context of threshold accepting, as it has been done for TA-GLPK (see Sect. 4), allows to tackle any kind of SLS problem. By replacing TADR with such a method in step 2 of Fig. 1, the decomposition approach could be adapted for extensions of the GLSPPL like time-indexed holding costs, time-indexed production coefficients, backlogging, or overtime.

Acknowledgment. The authors are grateful to Tomasz Bobinski, Fridolin Lander, and Daniel Müller for their helpful support in adapting the TAPLS code to TA-GLPK.

References

[1] B. Almada-Lobo, D. Klabjan, M. Carravilla, J. Oliveira, Multiple machine continuous setup lotsizing with sequence-dependent setups, Computational Optimization and Applications 47 (2010) 529–552.

[2] B. Almada-Lobo, J.F. Oliveira, M. Carravilla, Production planning and

scheduling in the glass container industry: A VNS approach, International Journal of Production Economics 114 (2008) 363–375.

[3] C. Almeder, B. Almada-Lobo, Synchronisation of scarce resources for a parallel machine lotsizing problem, International Journal of Production Research 49 (2011) 7315–7335.

[4] P. Amorim, C. Antunes, B. Almada-Lobo, Multi-objective lot-sizing and scheduling dealing with perishability issues, Industrial and Engineering Chemistry Research 50 (2011) 3371–3381.

[5] T. Baker, J. Muckstadt Jr., The CHES problems, Technical Report, Chesapeake Decision Sciences, Inc., 1989.

[6] J. Benders, Partitioning procedures for solving mixed variables programming, Numerische Mathematik 4 (1962) 238–252.

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

[8] D. Bertsekas, P. Tseng, RELAX–IV: a faster version of the RELAX code for solving minimum cost flow problems, Technical Report, Dept. of Electrical Engineering and Computer Science, M.I.T., Cambridge, Mass. 02139, 1994.

[9] L. Buschkühl, F. Sahling, S. Helber, H. Tempelmeier, Dynamic capacitated lot-sizing problems: a classification and review of solution approaches, OR Spectrum 32 (2010) 231–261.

[10] A. Clark, S. Clark, Rolling-horizon lot-sizing when set-up times are sequence-dependent, International Journal of Production Research 38 (2000) 2287–2307.

[11] COIN-OR, Coin-OR LP code (clp) homepage, https://projects.coin-or.org/Clp, 2013. Date: March 2013.

[12] G. Dantzig, P. Wolfe, Decomposition principle for linear programs, Operations Research 8 (1960) 101–111.

[13] S. Dauzère-Péres, J.B. Lasserre, Integration of lotsizing and scheduling decisions in a job-shop, European Journal of Operational Research 75 (1994) 413–426.

[14] 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.

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

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

[17] D. Ferreira, A. Clark, B. Almada-Lobo, R. Morabito, Single-stage formulations for synchronised two-stage lot sizing and scheduling in soft drink production, International Journal of Production Economics 136 (2012) 255–265.

[18] D. Ferreira, P. França, A. Kimms, R. Morabito, S. Rangel, C. Toledo, Heuristics and meta-heuristics for lot sizing and scheduling in the soft drinks industry: A comparison study, Studies in Computational Intelligence 128 (2008) 169–210.

[19] D. Ferreira, R. Morabito, S. Rangel, Solution approaches for the soft drink integrated production lot sizing and scheduling problem, European Journal of Operational Research 196 (2009) 697–706.

[20] FICO, Xpress Optimization Suite, http://www.fico.com/en/Products/DMTools /xpress-overview/Pages/Xpress-Optimizer.aspx, 2013. Date: March 2013.

[21] B. Fleischmann, The discrete lotsizing and scheduling problem, European Journal of Operational Research 44 (1990) 337–348.

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

[23] B. Fleischmann, H. Meyr, Planning hierarchy, modeling and Advanced Planning Systems, in: A. de Kok, S. Graves (Eds.), Supply Chain Management: Design, Coordination, Operation, volume 11 of *Handbooks in Operations Research and Management Science*, Elsevier, Amsterdam et al., 2003, pp. 457–523.

[24] B. Fleischmann, H. Meyr, M. Wagner, Advanced planning, in: H. Stadtler, C. Kilger (Eds.), Supply Chain Management and Advanced Planning, Springer, Berlin, Heidelberg, 2008, 4th edition, pp. 81–106.

[25] S. Ghosh Dastidar, R. Nagi, Scheduling injection molding operations with multiple resource constraints and sequence dependent setup times and costs, Computers and Operations Research 32 (2005) 2987–3005.

[26] Gnu Linear Programming Kit (GLPK), Homepage, http://www.gnu.org/soft ware/glpk/, 2013. Date: March 2013.

[27] H.O. Günther, M. Grunow, U. Neuhaus, Realizing block planning concepts in make-and-pack production using MILP modelling and SAP APO, International Journal of Production Research 44 (2006) 3711–3726.

[28] R. Jans, Z. Degraeve, Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches, European Journal of Operational Research 177 (2007) 1855–1875.

[29] J. Józefowska, A. Zimniak, Optimization tool for short-term production planning and scheduling, International Journal of Production Economics 112 (2008) 109–120.

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

[31] B. Karimi, S. Fatemi Ghomi, J. Wilson, The capacitated lot sizing problem: a review of models and algorithms, Omega 31 (2003) 365–378.

[32] U. Karmarkar, L. Schrage, The deterministic dynamic product cycling problem, Operations Research 33 (1985) 326–345.

[33] J.C. Lang, Production and Inventory Management with Flexible Bills-of-Materials and Substitutions, Springer, Berlin, 2009.

[34] J. Lasserre, An integrated model for job-shop planning and scheduling, Management Science 38 (1992) 1201–1211.

[35] G. Mateus, M. Ravetti, M. De Souza, T. Valeriano, Capacitated lot sizing and sequence dependent setup scheduling: An iterative approach for integration, Journal of Scheduling 13 (2010) 245–259.

[36] H. Meyr, Simultane Losgrößen- und Reihenfolgeplanung für kontinuierliche Produktionslinien, Deutscher Universitäts–Verlag, Wiesbaden, 1999.

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

[38] H. Meyr, Simultaneous lotsizing and scheduling on parallel machines, European Journal of Operational Research 139 (2002) 277–292.

[39] H. Meyr, Simultane Losgrößen– und Reihenfolgeplanung bei mehrstufiger kontinuierlicher Fertigung, Zeitschrift für Betriebswirtschaft 74 (2004) 585–610.

[40] D. Quadt, H. Kuhn, Capacitated lot-sizing with extensions: A review, 4OR 6 (2008) 61–83.

[41] F. Seeanner, B. Almada-Lobo, H. Meyr, Combining the principles of Variable Neighborhood Decomposition Search and the Fix&Optimize heuristic to solve multi-level lot-sizing and scheduling problems, Computers & Operations Research 40 1 (2013) 303–317.

[42] F. Seeanner, H. Meyr, Multi-stage simultaneous lot-sizing and scheduling for flow line production, OR Spectrum 35 1 (2013) 33–73.

[43] C. Toledo, P. França, R. Morabito, A. Kimms, Multi-population genetic algorithm to solve the synchronized and integrated two-level lot sizing and scheduling problem, International Journal of Production Research 47 (2009) 3097–3119.

[44] P. Tseng, D. Bertsekas, An ϵ-relaxation method for separable convex cost generalized network flow problems, Mathematical Programming 88 (2000) 85–104.

[45] X. Zhu, W. Wilhelm, Scheduling and lot sizing with sequence-dependent setup: A literature review, IIE Transactions 38 (2006) 987–1007.

Highlights

A novel solution heuristic for the General Lotsizing and Scheduling Problem for Parallel Production Lines is presented.

The idea is to iteratively decompose the parallel-line problem into a series of single-line problems.

The new heuristic improves already existing approaches.

Large practical applications can now be solved.