Computers & Operations Research 53 (2015) 338–352

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Product mix optimization for a semiconductor fab: Modeling approaches and decomposition techniques

Martin Romauch [a,*], Andreas Klemmt [b]

[a] University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria
[b] Infineon Technologies, Konigsbrücker Straße 180, 01099 Dresden, Germany

ARTICLE INFO

Available online 14 May 2014

Keywords:
Semiconductor production planning
Product mix optimization
Master planning
Resource pooling
Decomposition
Linear programming

ABSTRACT

To optimize the product mix of a semiconductor fab, the production capabilities and capacities are matched with the demand in the most profitable way. In this paper we address a linear programming model of the product mix problem considering product dependent demand limits (e.g. obligations and demand forecast) and profits while respecting the capacity bounds of the production facility. Since the capacity consumption is highly dependent on choosing from different production alternatives we are implicitly solving a static capacity planning problem for each product mix. This kind of planning approach is supported by the fluid flow concept and complete resource pooling in high traffic. We propose a general model that considers a wide range of objectives, and we introduce a heuristic that is based on the decomposition of the static capacity planning problem. A computational study reports on the quality of the decomposition approaches, and examples from practice demonstrate the versatility of the model.

1. Introduction

In this paper we are considering a fab with given capabilities and capacities (technology and scale of the machinery) and we have to decide on the product mix to maximize profit. Typically, we encounter certain obligations (e.g. contracts, work in progress, customer service, development) and expectations on the sales (e.g. demand forecasts, down turns, market saturation). Consequently, we consider the product's demand limits and the marginal profits, while taking the capacity limits of the production facilities into account.

Certainly, the best thinkable model to answer the question for the optimal product mix integrates all necessary aspects; Simulation or Discrete Event Simulation [3,6,20] is a method that comes very close to these needs. Simulation is employed to analyze and forecast the dynamic behavior of complex and stochastic systems. In case of a semiconductor fab the complexity arises from the intricated product routes (re-entrant flow, hundreds of steps, alternative processing, etc.), the large number of equipments (usually hundreds), the variety of tool types (batch processing, pipeline tools, cluster tools, etc.) and the vast number of other factors (transport, dispatching, machine shutdowns, repair, yield, rework, operators, etc.) that can or should be considered [22]. Depending on the purpose of the simulation model (e.g. cycle time forecast) some of the selected factors will be stochastic in nature and they need to be modeled with the proper distributions, therefore several runs are usually necessary to achieve the desired level of significance. Furthermore, there are also cases that go beyond simple dispatching rules (e.g. local scheduling) and it gets necessary to integrate an optimizer in the simulation models (cf. [18] and [2]). We note that factors like that can make the simulation model complex and error-prone [24]. Therefore, dependent on the time span, the level of detail, and the number of runs, the task of developing a model that is suitable for optimizing certain parameters is usually difficult and time consuming. Models that consider various aspects, typically, have a considerably high development cost and simulation runtime. Therefore, only a small number of parameters can be optimized employing such a framework, and optimizing several interdependent parameters at the same time is computationally intractable. Besides simulation, there are several alternative approaches to model a semiconductor fab, for example Zisgen et al. [25] present a framework that enables to analyze steady and transient states for queuing networks to forecast the cycle time and utilization on tool basis (MES level), but there is no extension that covers product mix optimization. In [23] and [7] you can find approaches to optimize the product mix by solving a mixed integer linear program and in both cases the capacity is captured by considering a predefined set of bottlenecks and the process times are not tool dependent which

* Corresponding author.
E-mail addresses: martin.romauch@univie.ac.at (M. Romauch), andreas.klemmt@infineon.com (A. Klemmt).

http://dx.doi.org/10.1016/j.cor.2014.04.017
0305-0548/

prevents to model certain production flexibilities. Multiple periods are considered in both approaches, but the cycle time of the individual products is an input parameter since it is captured in a predefined shift of the capacity consumption.

In contrast to that, we employ a static capacity model that includes all tools and allows to model tool dependent processing times (cf. [14]). Furthermore we will also explain how to model multiple periods and how to link the periods to balance the load between periods.

We assume that the profit arising from each product is linear in the number of units produced, and we will formulate a linear program to model the optimal product mix problem for the steady state system, that is based on the idea of heavy traffic resource pooling [14].

The underlying capacity model balances the load and minimizes the capacity usage for bottlenecks. The capacity limits are determined in advance (in practice, notions like the operating curve [1] and other performance indicators can be used to estimate the limits) therefore cycle times are not addressed directly, but in practice the capacities are properly chosen and the proposed load distribution policy leaves sufficient freedom to absorb short term disruptions encountered in daily business. According to that, the approach combines the product mix decision and load optimization, i.e. we are maximizing the profit while optimizing the load within the given capacity limits.

In [19] you can find a decomposition method for solving the product mix problem (compare Algorithm 1 in Section 4) where the product mix problem is solved on equipment basis in the master problem. It turns out that this method leaves considerable optimization gaps for certain types of instances. The approach presented in this paper aims to improve these results by integrating the resource pools concept in the decomposition.

We note that uncertain demand will not be discussed extensively in this paper, but we indicate that stochastic programming and sensitivity analyses are apparent ways to deal with random demand. A demonstration of the sensitivity to changes in the product mix is contained in Section 7 which covers practical experiences.

For robust planning approaches we refer to [16] and [4] (objective: demand satisfaction and minimizing the number of tools) and in [12] you can find a minimum cost formulation that considers noisy demand that is related to [23].

The paper is organized as follows. In Section 2 we discuss the static capacity problem and we introduce the basic concepts of load balancing, connected components and resource pools. In Section 3 we define a (global) linear program for the optimal product mix problem. In addition, we discuss an extension of this model. In Section 4 we propose a decomposition approach for the (global) product mix problem which originates in the approach presented in [19]. Here we iteratively solve small-sized linear sub problems that decouple the product mix and the resource allocation problem. To evaluate both methods we are considering a set of randomly generated instances. For this purpose, we describe a benchmark scheme in Section 5 and we present the results of our computational experiments in Section 6. We conclude with a discussion of practical experiences from semiconductor industry in Section 7, including a demonstration of select use cases that give an outlook to closely related problem settings.

2. Load balancing, connected components and resource pools

To solve the product mix problem we need to check the expected utilization, which is covered by solving the corresponding capacity problem. In this section we will briefly introduce the notions of the static capacity problem and the concept of load balancing and resource pooling. For more details on capacity planning in semiconductor industry we refer to [5,9]. For the theoretic concept of resource pooling in high traffic we refer to [10,14,15]. We begin with the explanation of an example which illustrates the problem of capacity planning. Suppose, that we are considering a lithography work center with four steppers $\{a, b, c, d\}$ with different setups and processing speeds (different technological generations) while ignoring other work centers. Fig. 1 gives some details of the structure of three routes that consist of particular process steps. We suppose that we already identified the so-called 'job classes' to aggregate those processes that are 'very similar' – and 'very similar' means that the processes of the same job class need to be qualified on the same tools and the corresponding process time can be regarded as identical from a capacity planning point of view. We can also find the service time matrix on level job class and resource in Fig. 1. The units for service time are given in hours per lot and only positive entries can be interpreted as a qualified job class and tool combination. According to the set diagram the job classes are connected and the problem cannot be decomposed. We can observe that the

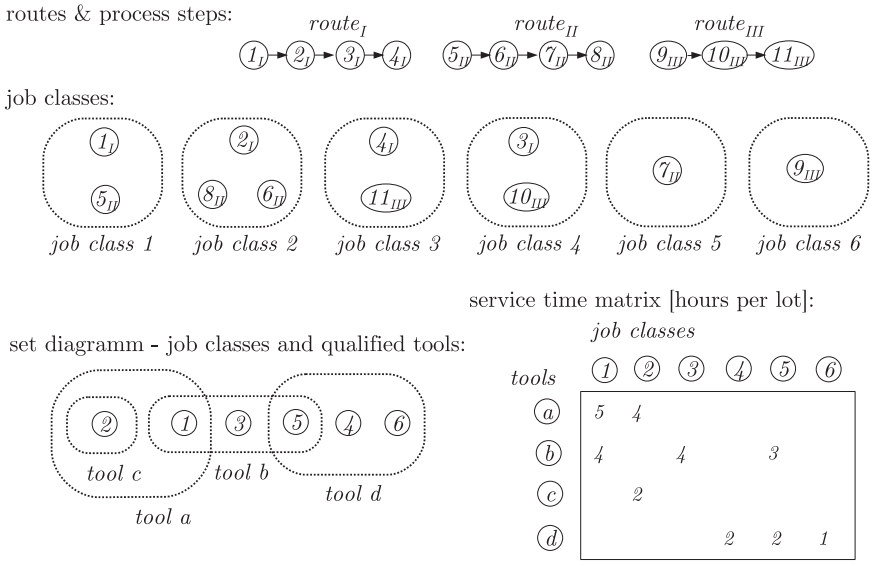

Fig. 1. Example: structure for three routes, job classes and qualified tools.

equipment c is qualified for only one job class, which means that the load of this equipment cannot be larger than the load of the job class. On the other hand, job class 3 is only released on tool b, which means that there is no alternative to shift load. Finally Fig. 2 combines the structure of the routes on the basis job classes and tools. We continue with Fig. 3 which is an example of a static capacity planning problem for the work center of the example given in Fig. 1. The static planning process starts with the demand of three products which are identified by the routes $route_I$, $route_{II}$ and $route_{III}$. The primary demand, the structure of the routes and the service time matrix are the input of the problem. In this example the dimension of the primary demand is given in lot starts per week. Due to the job class aggregation we can interpret each route as a sequence of job classes. And, according to the primary demand of the products we can calculate the demand λ on the basis of job classes and lots per week. Consequently, each job class j has a demand λ_j which needs to be satisfied by allocating its demand to available resources ($i \in I$). To satisfy the demand we need to distribute the demand λ_j to the resources in a convenient way. Let the variable x_{ji} hold the number of units of job

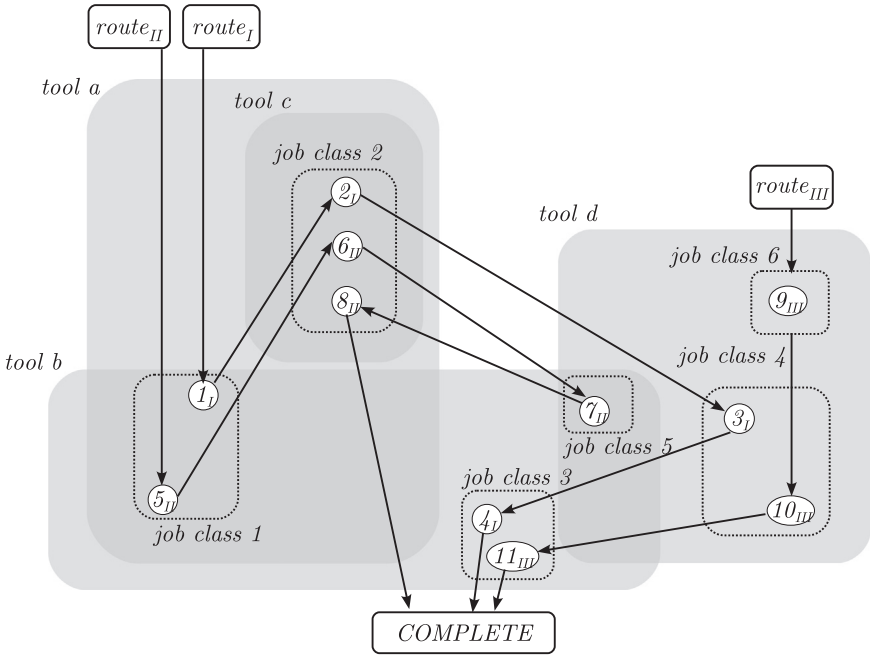

Fig. 2. Example: structural interdependence of job classes and qualified tools for three routes.

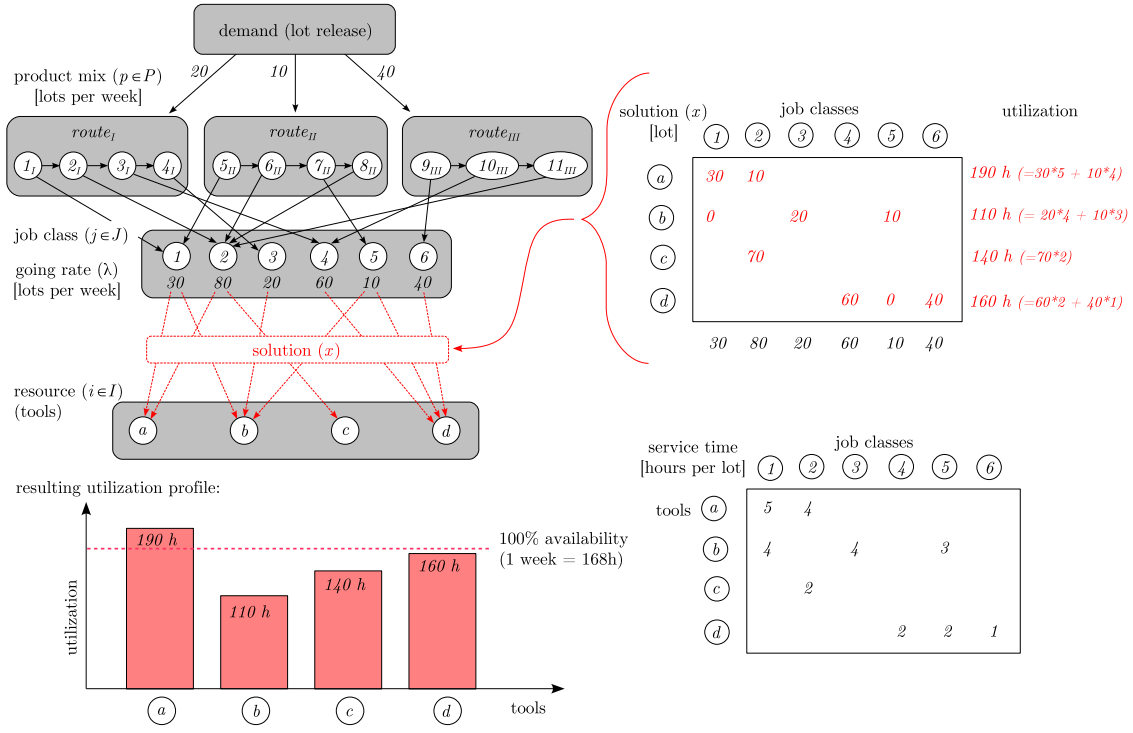

Fig. 3. Example: capacity planning – given a certain demand of products (lots) we have to assign the resulting job classes to the resources, which results in a corresponding utilization profile.

class j that are allocated to resource i and each of the possible 'activities' x_{ji} has a corresponding service rate of μ_{ji} units per time unit (lots per hour). The depicted solution x respects the technical requirements but tool a slightly exceeds the maximum availability of one week (168 h). We can also observe that resource a is connected to other resources that share the same job classes, i.e. job class 1 connects resources a and b and job class 2 connects resources a and b. And, we can use this flexibility to redistribute the work load in such a way that the maximum load is feasible and the utilization profile is more balanced (e.g. transfer six units of job class 1 from tool a to tool b).

Generally, we are interested in solutions that avoid bottlenecks and minimizing the maximum load therefore an intuitive approach. For this reason we formulate a simple linear program (LP) that includes the desired objective:

$$\min \quad \rho \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in I} x_{ji} = \lambda_j \quad (\forall j \in J), \tag{2}$$

$$\sum_{j \in J \wedge \mu_{ji} > 0} \frac{x_{ji}}{\mu_{ji}} \leq \rho \quad (\forall i \in I), \tag{3}$$

$$x \geq 0. \tag{4}$$

Since x_{ji} is the number of units of job class j assigned to resource i, the set of constraints (2) guarantees the satisfaction of the demand. The service rate μ_{ji} is the number of units from job class j that can be completed on resource i within one time unit, therefore $1/\mu_{ji}$ is the time needed to service one unit from job class j on resource i. Consequently, the constraints (3) are limiting the capacity with ρ as an upper bound. For the sake of completeness, we note that if $\mu_{ji} = 0$ then this combination will not be part of the model. The variable ρ is minimized and as a result, the maximum load is minimized. By transforming the variables $\tilde{x_{ji}} \leftarrow x_{ji}/\mu_{ji}$ (which can be interpreted as the time invested in the activity x_{ji}) we get the following equivalent LP, that highlights the queuing aspect:

$$\min \quad \rho \tag{5}$$

$$\text{s.t.} \quad \sum_{i \in I} \tilde{x_{ji}} \mu_{ji} = \lambda_j \quad (\forall j \in J), \tag{6}$$

$$\sum_{j \in J} \tilde{x_{ji}} \leq \rho \quad (\forall i \in I), \tag{7}$$

$$\tilde{x} \geq 0. \tag{8}$$

Solving the example in Fig. 3 with a simplex-based method we may end up with the following solution $(x_{ij}) = (17.5, 0, 12.5, 20, 10, 80, 60, 0, 40)$, where (i,j) is a feasible combination from $((1, a), (2, a), (1, b), (3, b), (5, b), (2, c), (4, d), (5, d), (6, d))$. The corresponding utilization profile is $u = (100, 150, 160, 160)$. We have to emphasize that this kind of objective does not cover all practical needs and therefore we can find several kinds and combinations of objectives in the literature (cf. [4,5]). One of these considerations is the minimization of the number of resources with maximum load. Convex quadratic programming is employed in [11] to tackle this problem; the proposed concept divides the resources into disjoint 'resource pools' with a homogenous load distribution that minimizes the maximum load on each level in a hierarchical cascaded way. These methods result in the following unique solution x and the corresponding unique utilization profile u:

$$x = (21.7647, 8.52941, 18.23529, 10, 10, 71.47059, 60, 0, 40),$$

$$u = (142.94, 142.94, 142.94, 160).$$

We can see that for this solution there is only one resource {d} with maximum load, while the resources {a,b,c} are balanced on a lower level. More general, each level defines a so–called *resource pool*. A resource pool can be understood as a set of resources that share a common work load as if all of them had the same capabilities. Fig. 4 illustrates the decomposition into resource pools for this example.

According to [26], we can get rid of the quadratic terms of the objective function, which is used to identify the resource pools, while keeping up the homogeneity goal. Before we present the concept of resource pools in more detail we introduce the notion of *connected components*. To do so, we define that two resources are called *connected* if a 'common' job class exists, which both resources are able to service.

Therefore a connected component contains resources that are directly or indirectly linked by job classes. We will synonymously use the term *closed machine set* (or *cms*) for connected components. In practice it is very unlikely that all resources are connected, and therefore it is usually possible to decompose the problem. In case that the objective is separable we can considerably reduce the complexity of the problem. A closed machine set can be refined by *resource pools* with respect to the demand. We will give a definition in three steps. First of all, we think of a specific connected component with a specific demand on the basis of job classes. According to the LP given in (5)–(8) we can define the minimal maximum load ρ^*, which is the first step. The second step is to determine the group of resources that 'necessarily' show the maximum load ρ^*, which means that there is no way to decrease the load on these resources without exceeding ρ^* on a resource outside the group. This set is then called the resource pool for the highest level (cf. [11]). The third step is the elimination of the identified resources and the serviced job classes from the LP given in (5)–(8) and we can determine the resource pool of the next level analogously. The procedure stops when no resource is left. Each time a resource pool is defined, we eliminate the corresponding resources and serviced job classes from the LP. The elimination will not lead to any ambiguity since there is no overlapping of the job classes – in fact, the resource pools induce a decomposition of the job classes. We note that it may happen that members of the same resource pool are not connected with respect to the serviced job classes and therefore a post–processing

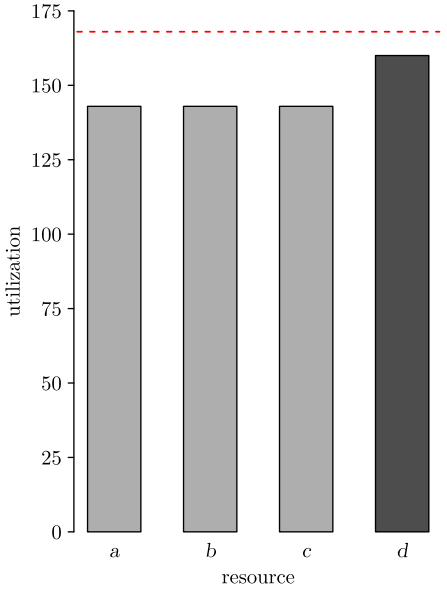

Fig. 4. In this example we find resource pools for two levels of utilization. The set of resources can be decomposed accordingly, i.e. $\mathfrak{R} = \{RP_1, RP_2\}$, where $RP_1 = \{d\}$, $RP_2 = \{a, b, c\}$.

step is necessary to determine the subdivision of resource pools on this level. In practice the members of a resource pool are closely related and the processes are often similar, e.g. it could be a group of furnaces for a specific technology and wafer diameter/thickness. We note that a resource pool indicates the potential of a perfect load balance which may deviate from monitored utilization. Therefore, especially for equipment that are classified as critical the corresponding resource pools may give useful insight on how to adjust the material flow to stabilize the production line.

Now we discussed the static capacity planning, the connection between primary and secondary demand as well as the concepts of connected components (*cms*) and resource pools. These concepts are the basis to understand the capacity aspects of the model in Section 3 and the algorithms described later in Section 4. In the next section we will introduce and formulate a model to maximize profit by changing the product mix with respect to the capacity and the demand – which is our primary goal.

3. Problem statement and model

In Section 2 we considered the problem of distributing the load optimally while the primary demand (product mix) was an input parameter. In this section the quantity of the products and the load distributions are part of the decision variables and we will formulate the corresponding product mix problem, and at the end of this section we will propose extensions of the model. First we introduce the indices, variables and the parameters used in the model:

- *Indices*
  - product index p: enumeration of the products,
  - job class index j: enumeration of the job classes,
  - resource index i: enumeration of the resources,
  - activity index k: an enumeration of the allowed allocations of job classes to resources, i.e. each k identifies a feasible combinations (j_k, i_k) of a job class j_k and a resource i_k where $\mu_{j_k i_k} > 0$. The set of all activity indices is called K.
- *Decision variables*
  - product mix $y = (y_p)$: production quantity,
  - activity $x = (x_k)$: quantity of job class j_k assigned to a resource i_k (allocation).
- *Input parameters*
  - technology matrix $D = (d_{jp})$: d_{jp} says how often product p needs a process step from job class j to be done,
  - dispersal matrix $A = (a_{jk})$: a_{jk} equals to one if $j = j_k$, else it is zero. Therefore each row of the matrix contains the spectrum of valid combinations for the allocation of the corresponding job class,
  - service time matrix $R = (r_{ik})$: r_{ik} contains the processing time per unit, i.e. if $i = i_k$ we set $r_{ik} = 1/\mu_{j_k i_k}$,
  - profit vector $c = (c_p)$: each product has a certain profit,
  - demand limit vectors $\underline{\Delta}$ and $\overline{\Delta}$: each product has a lower and an upper demand limit ($\underline{\Delta}_p \geq 0$ and $\underline{\Delta}_p \leq \overline{\Delta}_p$),
  - maximum capacity vector $\rho_{max} = ([\rho_{max}]_i)$ is a vector, where $[\rho_{max}]_i$ is the maximum capacity of resource i, we note that ρ_{max} impacts the cycle time (cf. operating curve).

In analogy to the formulation in [13] we present a simplified version of the optimal product mix problem:

$$\max \quad c^T y \tag{9}$$

$$\text{s.t.} \quad Ax = Dy, \tag{10}$$

$$Rx \leq \rho_{max}, \tag{11}$$

$$x \geq 0, \tag{12}$$

$$\underline{\Delta} \leq y \leq \overline{\Delta}. \tag{13}$$

The objective (9) is to maximize the profit. The constraints (10) enforce that the secondary demand (Dy) is properly assigned to the resources. The constraints (11) ensure that the maximum ρ_{max} is not exceeded.

Lemma 1. *Without loss of generality we can think of the vector ρ_{max} as a scalar parameter.*

Proof. The constraints (11) are equivalent to

$$\sum_{k:i_k = i} \frac{r_{ik}}{[\rho_{max}]_i} x_k \leq 1 \quad \forall i \in I.$$

Therefore the following transformation completes the proof:

$$r_{ik} \leftarrow \frac{r_{ik}}{[\rho_{max}]_i}. \quad \square$$

For modeling additional capacity (e.g. investment or silicon foundries) and the aspect of soft bottlenecks (planned cycle time deviations) we propose to add the following variables and parameters to the problem (9)–(13):

- additional variables
  - $\overline{\rho}_1 = ([\overline{\rho}_1]_i)$ the consumed amount of the bottleneck capacity on equipment i (makes i a soft bottleneck),
  - $\overline{\rho}_2 = ([\overline{\rho}_2]_i)$ the amount of additional capacity of equipment i (invest and out-sourcing),
  - $w_p^{prod} \in \{0, 1\}$ is equal to one if product p is produced; zero otherwise,
  - $w_i^{equ} \in \{0, 1\}$ is one if resource i is in service; zero otherwise.
- additional input parameters
  - $f^{prod} = (f_p^{prod})$ fixed cost for launching a product p, these costs incur if the production quantity is positive ($y_p > 0$ and therefore the auxiliary binary variable w_p^{prod} will be one). Apart from expenses for the development there are several other fixed costs that are related to the tool setup and the infrastructure.
  - $f^{equ} = (f_i^{equ})$ fixed cost for using an equipment i, these costs incur if at least one activity for i is positive In case of downturns it may save money to shut down (cold steel) or even sell parts of the equipment, (or if $(Rx)_i > 0$, and hence $w_i^{equ} = 1$),
  - $g = (g_i)$ cost for one additional capacity unit,
  - $h = (h_i)$ bottleneck budget consumption for one additional unit of resource i,
  - B_1 available bottleneck budget,
  - B_2 available budget for additional capacity,
  - $\overline{R}_1 = ([\overline{R}_1]_i)$ maximum available bottleneck capacity of type i (soft bottleneck capacity),
  - $\overline{R}_2 = ([\overline{R}_2]_i)$ maximum available additional capacity of type i,
  - the aggregation matrix $V = (V_{vp})$; $V_{vp} \in \{0, 1\}$ where $V_{vp} = 1$ if product p is a member of the aggregation group v. If V is the identity matrix, we can set the lower and upper bound for each product separately, cf. (13).

This leads us to the following (extended) model:

$$\max \quad c^T y - g^T \overline{\rho}_2 - (f^{prod})^T w^{prod} - (f^{equ})^T w^{equ} \tag{14}$$

$$\text{s.t.} \quad Ax = Dy, \tag{15}$$

$$Rx \leq \rho_{max} + \overline{\rho}_1 + \overline{\rho}_2, \tag{16}$$

$$y \le \max(\overline{\Delta}) w^{prod}, \tag{17}$$

$$(Rx)_i \le ([\rho_{max}]_i + [\overline{R}_1]_i + [\overline{R}_2]_i) w_i^{equ} \quad (\forall i \in I), \tag{18}$$

$$h^T \overline{\rho}_1 \le B_1, \tag{19}$$

$$g^T \overline{\rho}_2 \le B_2, \tag{20}$$

$$\overline{\rho}_1 \le \overline{R}_1, \tag{21}$$

$$\overline{\rho}_2 \le \overline{R}_2, \tag{22}$$

$$x, y, \overline{\rho}_1, \overline{\rho}_2 \ge 0, \tag{23}$$

$$w^{prod} \in \{0,1\}^{|P|}, \quad w^{equ} \in \{0,1\}^{|I|}, \tag{24}$$

$$\underline{\Delta} \le Vy \le \overline{\Delta}. \tag{25}$$

The objective function (14) includes the revenue $c^T y$, the costs $g^T \overline{\rho}_2$ for additional capacity and fixed costs $(f^{equ})^T w^{equ}$ for putting a resource into operation. Since each product involves considerable efforts, we also propose fixed costs $(f^{prod})^T w^{prod}$ for launching products and we note that these variables are only relevant for products p where $\underline{\Delta}_p = 0$. The constraints in (15) are used to ensure that the demand on job class level is satisfied, while the constraints (16) are used to limit the maximum load. In contrast to (11) we introduce the vectors $\overline{\rho}_1$ and $\overline{\rho}_2$ that allow to go beyond ρ_{max} by slightly overloading (planned soft bottlenecks) or by providing additional resources (e.g. invest or outsourcing). The constraints (17) are used to model the variable w^{prod}, i.e. $y_p > 0 \Rightarrow w_p^{prod} = 1$ and analogously, the set of constraints (18) enforce that $w_i^{equ} = 1$ if there is a load on resource i. The vector h in (19) contains consumption factors for the soft bottleneck contingent B_1, allowing to put specific weights on different resources, while the constraints (21) allow to set resource specific upper bounds $\overline{R}_1$. According to (20) we define an outsourcing budget of B_2 and in (22) we state resource specific upper bounds $\overline{R}_2$ on the additional capacity. Since each row of V can be interpreted as a selection of products, (Vy) contains the corresponding sums, and therefore the set of constraints (25) can be used to model lower and upper bounds for products on different aggregation levels.

Remark 1. In practice we choose the vector ρ_{max} in such a way that all resources that do not exceed ρ_{max} are running smoothly with stable queues. That means that also if the corresponding limit is reached we still would not expect to see cycle time deviations on the corresponding tools. For allowing 'soft' bottlenecks (which is a slight accepted excess of ρ_{max}) we have to choose h, $\overline{R}_1$ in (19) and (21) accordingly. E.g. by setting $\overline{R}_1 = 0.05\rho_{max}$, $h^T = 1/\rho_{max}^T$ and $B_1 = N$ we allow to add 5% to the maximum capacity using the bottleneck capacity and in total not more than the equivalent of N resource units. In this example we suppose that a 5% increase will not result in a cycle time deviation if appropriate bottleneck management measures are taken into account. For instance it may be possible to accelerate certain lots to keep the potential bottleneck busy, or to take extra care to avoid any waiting times at critical stations, in case of furnaces the batching strategy could also be altered to avoid capacity loss. We also remark that in this formulation we have a limit on the total bottleneck excess of all resources, but the number of resources that show an excess are not limited by a constraint (continuous formulation). Obviously, the parameters g, $\overline{R}_2$ can be used to model the effect of additional capacity. To add fixed cost for products we may choose f^{prod} accordingly and the constraints (17) are used to model the binary variable w^{prod} which indicates if a product is part of the product mix or not. Fixed costs for additional equipment are modeled analogously (compare w^{equ} and (18)), in this case we can interpret g as the costs for outsourcing capacity.

Remark 2. In analogy to (14)–(25) several other extensions of model (9)–(13) are possible, too. For instance, the minimization of the number of allowed bottlenecks, 'cold steel' – (machine shutdowns) and sensitivity analyses. Most of these extensions (like the variable in (24)) lead to Integer Programming formulations which are not in the focus of this paper. In Section 7 we will refer to extension that is also suitable for the decomposition method proposed in Section 4.

Remark 3. The current formulations are purely static and only a single period (or time frame) is considered. To integrate a dynamic aspect we propose to introduce time frames, e.g. weeks or months which are relevant choices for practical needs. To assign the right quantity for each job class to the corresponding time frame, we take the due cycle time (which is dependent on $[\rho_{max}]_i$, cf. operating curve) for each processing step of the product into account. That means that we calculate the cumulative due cycle time for each step of the product, and then we can add the calculated amounts to the corresponding job classes and time frames. We also mention that we are able to consider different release dates by constructing products that are dependent on the time frame. We want to emphasize that from a formal point of view this kind of extension can be interpreted as a pre-processing step that does not change the model itself. The only difference is that the resources and job classes get an additional indicator for the time frame. In fact, the same resource in different time frames can be seen as two different resources and the same job class in different time frames can be seen as two different job classes as well. Therefore the model is identical, but it has a particular structure. We want to mention that in some cases the assignment of job classes based on the cumulative cycle time can be a ambiguous, therefore we propose to introduce transition job classes that can be served by resources from neighboring time frames. Another way to enforce the dynamic aspect is to iteratively adopt the due cycle time according to the achieved utilization profile. Finally we note that all parameters that are dependent on a resource may also vary in time without changing the model itself, and therefore it enables us to evaluate a broad set of scenarios.

To estimate the size of the linear program (9)–(13) we notice that the cardinality of the set of activity indices K is equal to the number of nonzero entries in the service time matrix and let L be the average number of job classes for each product. We summarize some facts about the size of the coefficient matrix of (9)–(13) in the following proposition.

Proposition 1. *The number of rows, columns and nonzeros of (9)–(13) can be estimated according to the following formulas*:

- *number of rows*: $|J| + |I|$,
- *number of columns*: $|P| + |K| + 1$,
- *number of nonzeros*: $|P| \times L + 2|K|$.

We notice that for this formulation it is not possible to directly take advantage of a decomposition on the basis of connected components, because it is very likely that the resources are interconnected by the products and therefore the size of this problem will be quite large. For example if we consider 1000 service stations, 1000 products and 5000 job classes with average of 5 released service stations we will get $|K| = 25{,}000$ and if each product has 100 job classes ($L = 100$) in average, we get in total: 6000 rows, 26,001 columns and 152,000 nonzeros. In Section 4 we propose a decomposition of the problem into a product mix problem (master problem) and a resource allocation problem

(subproblem) that are iteratively solved. The decomposition decouples the product mix step from the detailed level of job classes and the resource allocation step is a capacity planning problem that is decomposable on the basis of connected components. Due to the corresponding complexity reduction and due to the opportunity to employ parallel computations we can expect to gain a considerable speed up for larger instances.

4. A decomposition heuristic

To solve the linear program defined in (9)–(13) we will first present the heuristic method proposed in [19] (Algorithm 1) and later in this section we propose an alternative method that is based on connected components. Instead of solving the problem (9)–(13) in one step, we iteratively solve a master problem (product level) which uses precalculated resource consumption factors on level product and resource to optimize the product mix. The coupling of products and job classes (10) is considered in the subproblem which is used to update the resource consumption factors. We start with introducing the master problem:

$$\max \quad c^T y \tag{26}$$

$$Ty \le \rho_{max}, \tag{27}$$

$$0 \le \underline{\Delta} \le y \le \overline{\Delta}. \tag{28}$$

The objective function (26) and the bounds on y (28) are identical to (9) and (13). And, the resource consumption matrix T in (27) is used to calculate the resulting utilization for each resource i with respect to a given product mix y. To be more precise T_{ip} is the estimated resource consumption of one unit of product p on resource i. We note that the factors T_{ip} are calculated in the initial step of the algorithm, and they are going to be updated in each iteration. Ideally, we can think of these factors as an estimation that is recursively improved by solving static capacity problems. To formulate the algorithm we represent the master problem (26)–(28) and the subproblem (5)–(8) by functions that return the optimal values for given instances. The master problem returns the product mix y and the objective value z and the subproblem returns the optimal distribution $\tilde{x}$ and the value ρ and the corresponding resource pools $\Re$ which will be omitted in the first algorithm:

$$MP : (T, c, \rho_{max}, \underline{\Delta}, \overline{\Delta}) \mapsto (y, z)$$

$$LB : (R, A, \lambda) \mapsto (\Re, \tilde{x}, \rho).$$

We note that the function LB is minimizing the maximum load using the resource pooling concept on all levels. In case that we have more than one connected component we are also able to decompose and parallelize the calculation. For the sub problem we take the output y of the master problem and we solve the corresponding static capacity problem $LB(R, A, \lambda = Dy)$, where Dy denotes the secondary demand. In general ρ_{max} is dependent on the specific resource, therefore we note that according to Lemma 1 and by transforming the service rates accordingly we can substitute the vector ρ_{max} by a scalar in the master problem and therefore it is consistent to minimize the single variable ρ in the sub problem. Part of the output of the static capacity problem is given by the load distribution $\tilde{x}$ which is used to update the load factors. We assume that the lower bound $\underline{\Delta}$ is a feasible product mix, i.e. the load resulting from the arrival rate $\lambda = D\underline{\Delta}$ does not exceed the load limit ρ_{max}. Fig. 5 outlines the flow of information of the proposed decomposition algorithm including the decomposition of the subproblem (load balancing) which enables parallel computation. For the decomposition of the subproblem we consider the connected components $\tau \in \{1, \ldots, Q\}$ which are represented by the corresponding set of feasible combinations CMS_τ, where $(i, j) \in CMS_\tau$ if and only if i is a resource of component τ and $\mu_{ij} > 0$. The formalization of the algorithm covers the decomposition with respect to the decoupling of the product mix decision and the static capacity allocation (master problem and subproblem), the decomposition with respect to the connected components will not be part of the formalization since it is a technical detail restricted to the subproblem $LB(R, A, \lambda)$.

We will now summarize the purpose of the steps of Algorithm 1:

- line 1–2: defines the input and output,
- line 3: by setting the product mix to $y_p = 1$ (for all products p) we will ensure that all products are considered in the resource consumption matrix T. The secondary demand $\lambda = Dy$ with respect to y is also calculated in this step,
- line 4–5: according to the secondary demand λ we calculate a distribution of the load on job class level $\tilde{x}$ and we can use this information to calculate the share of the load for each product on each resource. We note that d_{jp}/λ_j is relative portion of product p on λ_j and therefore T_{ip} is the average load induced by one unit of product p,
- line 6–8: these steps are analogous to 3–5 but with a different product mix (which may not cover all products). We also note that we only update those factors that are contained in the product mix.
- line 9: sets the iteration counter ι to one,
- line 10: the while loop continues as long as we can see relevant improvements, i.e. by setting the input parameter $\epsilon > 0$ the algorithm will terminate if the changes are smaller than ϵ,
- line 11: based on the factors in T we solve the master problem to improve the product mix y,
- line 12–13: is analogous to 6–8,
- line 15: increments the iteration counter ι.

Algorithm 1. Product mix optimization (simple decomposition).

1: input: $D, A, R, c, \rho_{max}, \underline{\Delta}, \overline{\Delta}, \epsilon$
2: output: y
3: $y \leftarrow 1, \quad \lambda \leftarrow Dy$
4: $(\tilde{x}, \rho) \leftarrow LB(R, A, \lambda)$
5: $\forall_p \forall_i : \quad T_{ip} \leftarrow \sum_{j:\lambda_j > 0} \tilde{x}_{ji} \frac{d_{jp}}{\lambda_j}$
6: $y \leftarrow \underline{\Delta}$, $\lambda \leftarrow Dy$, $z_0 \leftarrow c^T y$
7: $(\tilde{x}, \rho) \leftarrow LB(R, A, \lambda)$
8: $\forall_{p:y_p > 0} \forall_i : \quad T_{ip}(x) \leftarrow \sum_{j:\lambda_j > 0} \tilde{x}_{ji} \mu_{ji} \frac{d_{jp}}{\lambda_j}$
9: $\iota \leftarrow 1$
10: **while** $(\iota = 1) \vee (z_\iota - z_{\iota-1} > \epsilon)$ **do**
11: $(y, z_\iota) \leftarrow MP(T, c, \rho_{max}, \underline{\Delta}, \overline{\Delta})$
12: $\lambda \leftarrow Dy$
13: $(\tilde{x}, \rho) \leftarrow LB(R, A, \lambda)$
14: $\forall_{p:y_p > 0} \forall_i : \quad T_{ip}(x) \leftarrow \sum_{j:\lambda_j > 0} x_{ji} \mu_{ji} \frac{d_{jp}}{\lambda_j}$
15: $\iota \leftarrow \iota + 1$
16: **end while**

The following proposition states that Algorithm 1 converges to a feasible solution.

Proposition 2. *Algorithm 1 terminates with a feasible solution.*

Proof. Without loss of generality we can assume that $\rho_{max} = 1$ (see Lemma 1). The first thing to show is that the load factors T_{ip} represent a feasible marginal load distribution in each iteration, i. e. for a given y there exist $T, \tilde{y}, \tilde{\lambda}$ and $\tilde{x}$ from the previous iterations

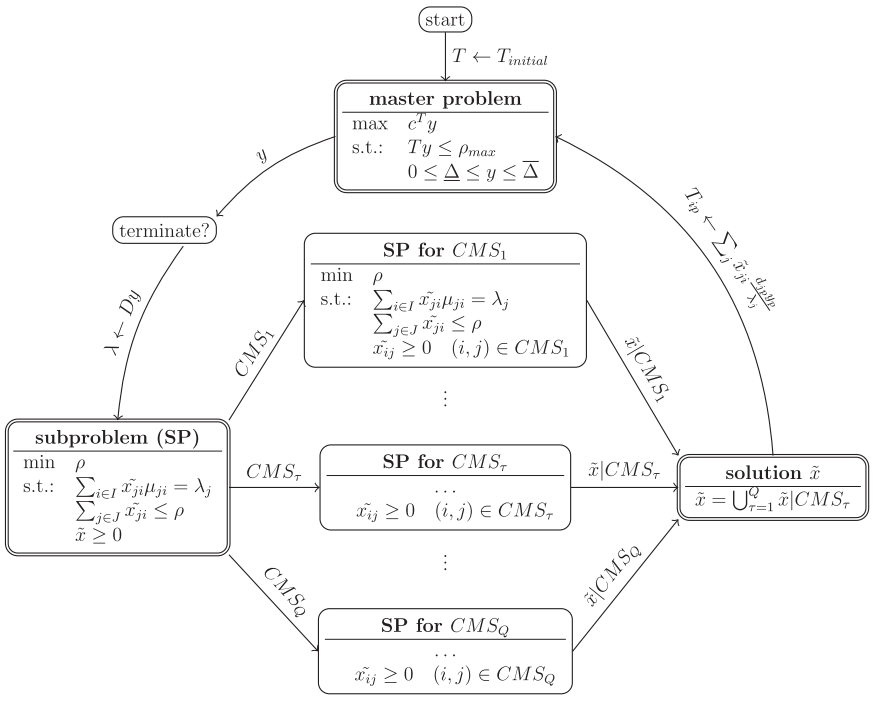

Fig. 5. Master problem and subproblem and the parallelization with respect to connected components CMS_τ.

such that

$$(Ty)_i = \sum_p y_p \sum_{j:\tilde{\lambda}_j > 0} \frac{d_{jp}}{\tilde{\lambda}_j} \tilde{x}_{ji}\mu_{ji} = \sum_{j:\tilde{\lambda}_j > 0} \left(\sum_p \frac{d_{jp} y_p}{\tilde{\lambda}_j} \right) \tilde{x}_{ji}\mu_{ji}.$$

Since $\sum_p d_{jp}\tilde{y}_p = \tilde{\lambda}_j$ we can see that $\sum_j (d_{jp}/\tilde{\lambda}_j)\tilde{x}_{ji}\mu_{ji}$ is the load induced by one unit of product p with respect to the allocation $\tilde{x}$ from the load balancing problem of the previous iteration. Furthermore, if $(Ty)_i \leq 1$ we can construct a distribution x such that $\sum_j x_{ji}\mu_{ji} \leq 1$, namely

$$x_{ji} = \begin{cases} \left(\sum_p \frac{d_{jp} y_p}{\tilde{\lambda}_j} \right) \tilde{x}_{ji} & \text{if } \lambda_j > 0 \\ 0 & \text{else} \end{cases}.$$

Therefore the solution x is feasible for the load balancing problem in the next iteration showing that ρ is not larger than one. After solving the load balancing problem we get a new ρ and updated load factors $\tilde{T}$. Since $y = \tilde{y} \Rightarrow x = \tilde{x}$ we can see that $(\tilde{T}y)_i \leq \rho$ and if $(\tilde{T}y)_i < 1$ we may use the available resource to improve the profit. Hence the sequence of objective values is non-decreasing. We assume that there is no product that has no capacity consumption, so we expect that the profit is bounded and we get a convergent sequence of objective values. Therefore, Algorithm 1 terminates with a feasible solution.

Remark 4. In each iteration of Algorithm 1 we are solving the master problem and the load balancing problem. These two steps determine the complexity of the algorithm. The master problem is a simple LP and the coefficient matrix is usually quite dense with $|I|$ rows and $|P|$ columns. The coefficient matrix of the load balancing problem is a block diagonal matrix, where each block is a connected component. According to the connected components $\tau (\tau = 1 \ldots Q)$ we can decompose the set of resources $(I = \bigcup_{\tau=1}^{Q} I_\tau)$, the index set for the activities $(K = \bigcup_{\tau=1}^{Q} K_\tau)$ and also the set of job classes $(J = \bigcup_{\tau=1}^{Q} J_\tau)$. Therefore, the load balancing problem of component τ has a coefficient matrix with $|J_\tau| + |I_\tau|$ rows, $|K_\tau| + 1$ columns and $2 \times |K_\tau|$ nonzeros. Suppose that we are calculating in parallel, then the largest connected component will be dominant and it will define the time complexity. Besides the effect of employing parallel computing, a comparison with the results here and Proposition 1 (which captures the size of the coefficient matrix of the global problem) points out that the number of columns and nonzeros are considerably reduced.

We notice that the distribution of load in the master problem is dependent on the load distribution x_{ji} and the portions d_{jp}/λ_j. In general the service rate for the same job class is dependent on the equipment and some job classes may be released on only a few resources within a resource pool. Dependent on a specific product mix the resources with maximum load in the optimal distribution may be non-homogeneous with respect to equipment and job classes. We note that the resource consumption factors T_{ip} are tailored to a certain product mix on equipment level. Other more profitable combinations may be unreachable in the master problem, since substantially different distribution of the job classes would be needed to optimize the resulting load. Therefore it is likely that Algorithm 1 can get stuck in a local optimum. With regards to the qualitative type of distribution of job classes, we also note that the tradeoff between capacity consumption and profit is more accurately estimated in the master problem if the load consumption factors are equal for the resources in the bottleneck.

Since Algorithm 1 lacks flexibility of distributing the workload we propose to project the load consumption factors on resource pool level instead of equipment level. Starting with an initial product mix we calculate the initial utilization profile and the corresponding resource pools. A resource pool allows to balance the work load on its components, therefore we can assume that as long as we do not destroy the structure of the resource pools we can balance the load on all its components (which is in particular true for small changes of the product mix). If we encounter a product mix that alters the resource pool structure, we need to refine the resource pools. To formalize the proposed Algorithm we start with defining the refinement of two partitions.

Definition 1. Let $\mathfrak{A}$ and $\mathfrak{B}$ be partitions of a set I. A *refinement* $\mathfrak{C}$ of $\mathfrak{A}$ and $\mathfrak{B}$ is itself a partition and has the following property: each member of $\mathfrak{C}$ is fully contained in exactly one member of $\mathfrak{A}$ and exactly one member of $\mathfrak{B}$, i.e.:

$$\forall C \in \mathfrak{C} \Rightarrow \exists A \in \mathfrak{A}, \quad \exists B \in \mathfrak{B} : C \subset A \wedge C \subset B.$$

Definition 2. Let $\mathfrak{A}$ and $\mathfrak{B}$ be partitions of an arbitrary finite set I (e.g. the set of available resources). We use the symbols $\mathfrak{A} \oplus \mathfrak{B}$ to define the *smallest common refinement* of $\mathfrak{A}$ and $\mathfrak{B}$ and we can calculate $\mathfrak{A} \oplus \mathfrak{B}$ as follows:

$$\mathfrak{A} \oplus \mathfrak{B} = \{A \cap B : A \in \mathfrak{A}, B \in \mathfrak{B}, A \cap B \neq \phi\}. \tag{29}$$

Proof. To show that the identity (29) in Definition 2 is correct we need to show that $\mathfrak{C} = \mathfrak{A} \oplus \mathfrak{B}$ is a partition and that it is the smallest one. Suppose that i is an arbitrary element of I. Since $\mathfrak{A}$ and $\mathfrak{B}$ are partitions there exists an element $A \in \mathfrak{A}$ and an element $B \in \mathfrak{B}$ such that $i \in A \cap B$ therefore $\mathfrak{A} \oplus \mathfrak{B}$ contains all elements of I. To show that the pairwise intersection of elements is empty we take two arbitrary distinct elements C and C' from $\mathfrak{A} \oplus \mathfrak{B}$. According to the Definition 2 we can find corresponding representations $C = A \cap B$ and $C' = A' \cap B'$. W.l.o.g we can assume that $A \neq A'$ and therefore $C \cap C' = \phi$. To show that $\mathfrak{A} \oplus \mathfrak{B}$ is the smallest refinement we only need to note that according to Definition 1 each nonempty intersection $A \cap B$ ($A \in \mathfrak{A}$ and $B \in \mathfrak{B}$) results in at least one representative in the refinement. □

The key to improving Algorithm 1 is to successively refine a set of resource buckets $\mathfrak{B}_\iota$ which are used in an adapted master problem as capacity units. Ideally, instead of using single resources we are combining them to form compounds (buckets) that are strongly connected (resource pooling) to gain flexibility for the load balancing. The resource pooling property is dependent on the product mix, therefore in each iteration ι we are considering refinements of the resource buckets that arise from the maximum resource pools found in the respective load balancing problem. Usually there will be only one resource pool with maximum load, therefore a refinement is splitting some of the buckets into two parts. Before we formalize this concept we define the notions *resource buckets*, *adapted master problem*, *maximum resource pools* and the corresponding partition $\mathfrak{R}^*$.

Definition 3. Suppose a solved load balancing problem is given, i.e. $(\mathfrak{R}, x, \rho) \leftarrow LB(R, A, \lambda)$, where $\mathfrak{R}$ ($\mathfrak{R} \subset \mathcal{P}(I)$ and I, the set of resources) is the set of resource pools, e.g.$\{RP_1, RP_2, \ldots\}$. With respect to $\mathfrak{R}$ we can define the set of maximum resource pools in the following way: $\mathfrak{R}_\rho = \{RP \in \mathfrak{R}, \forall i \in RP : u_i = \rho\}$. To define the partition associated with the maximum resource pools, we only need to add the quasi complement of $\mathfrak{R}_\rho$, i.e.

$$\mathfrak{R}^* = \mathfrak{R}_\rho \cup \left\{ I \Big\backslash \bigcup_{RP \in \mathfrak{R}_\rho} RP \right\}.$$

To illustrate the definitions we refer to the resource pools from the example given in Section 1, whose resource pools are depicted in Fig. 4. Here we get $\rho = 160$ and $\mathfrak{R}_\rho = \{d\}$, the corresponding partition $\mathfrak{R}^*$ is $\{\{d\}, \{a, b, c\}\}$. We continue with the definition of the *resource buckets*.

Definition 4. The set of *resource buckets* $\mathfrak{B}$ is a partition of the set of resources I, i.e. $\mathfrak{B} \subset \mathcal{P}(I)$. Each resource bucket $B \in \mathfrak{B}$ represents a capacity unit in the adapted master problem.

Definitions 3 and 4 allow us to formalize the refinements of the resource buckets in each iteration ι, i.e. $\mathfrak{B}_\iota \leftarrow \mathfrak{B}_{\iota-1} \oplus \mathfrak{R}^*$. According to a specific set of resource buckets $\mathfrak{B}$ (e.g. resource pools) we introduce the adapted master problem $\tilde{MP}(\mathfrak{B}, \tilde{T}, c, \rho_{max}, \underline{\Delta}, \overline{\Delta})$, where $\tilde{T}$ is a matrix with $|\mathfrak{B}|$ rows:

$$\max \quad c^T y \tag{30}$$

$$\tilde{T} y \leq \rho_{max}, \tag{31}$$

$$0 \leq \underline{\Delta} \leq y \leq \overline{\Delta}. \tag{32}$$

To define $\tilde{T}$ we represent a set of resource buckets $\mathfrak{B}$ in an enumerated form, i.e. $\mathfrak{B} = \{B_b, b = 1 \ldots |\mathfrak{B}|\}$ and then the matrix $\tilde{T}$ is defined by the distribution (x_{ji}) of the previously solved load balancing problem and the following formula:

$$\tilde{T}_{bp} = \frac{\sum_{i \in B_b} \sum_{j : d_{jp} > 0} x_{ji} \mu_{ji} \frac{d_{jp}}{\lambda_j}}{|B_b|}$$

Algorithm 2. Product mix optimization (based on resource pools).

1: input: $D, A, R, c, \rho_{max}, \underline{\Delta}, \overline{\Delta}, \epsilon$
2: output: y
3: $\iota \leftarrow 1$
4: $y^{(\iota)} \leftarrow \underline{\Delta}$, $\lambda \leftarrow Dy^{(\iota)}$, $z_\iota \leftarrow c^T y^{(\iota)}$
5: $(\mathfrak{R}, x, \rho) \leftarrow LB(R, A, \lambda)$
6: $\mathfrak{B}_\iota \leftarrow \mathfrak{R}$, $\mathfrak{B}_{\iota-1} \leftarrow \mathfrak{B}_\iota$
7: $\forall\, p \in \mathcal{P} : y_p^{(\iota)} > 0 \;\; \forall\, i \in I$: $T_{ip} \leftarrow \sum_{j : d_{jp} > 0} x_{ji} \mu_{ji} \frac{d_{jp}}{\lambda_j}$
8: $y \leftarrow 1$, $\lambda \leftarrow Dy$
9: $(\mathfrak{R}, x, \rho) \leftarrow LB(R, A, \lambda)$
10: $\forall\, p \in \mathcal{P} : y_p^{(\iota)} = 0 \;\; \forall\, i \in I$: $T_{ip} \leftarrow \sum_{j : d_{jp} > 0} x_{ji} \mu_{ji} \frac{d_{jp}}{\lambda_j}$
11: $\forall\, p \in \mathcal{P} \;\; \forall\, 1 \leq b \leq |\mathfrak{B}_\iota|$: $\tilde{T}_{bp} \leftarrow \frac{\sum_{i \in B_b} T_{ip}}{|B_b|}$
12: $\delta \leftarrow \epsilon$
13: **while** $\delta \geq \epsilon \vee \mathfrak{B}_\iota \neq \mathfrak{B}_{\iota-1}$ **do**
14: $\iota \leftarrow \iota + 1$
15: $(y^{(\iota)}, z_\iota) \leftarrow \tilde{MP}(\mathfrak{B}_\iota, \tilde{T}, c, \rho_{max}, \underline{\Delta}, \overline{\Delta})$
16: $\lambda \leftarrow Dy^{(\iota)}$
17: $(\mathfrak{R}, x, \rho) \leftarrow LB(R, A, \lambda)$
18: $\mathfrak{B}_\iota \leftarrow \mathfrak{B}_{\iota-1} \oplus \mathfrak{R}^*$
19: $\forall\, p \in \mathcal{P} : y_p^{(\iota)} > 0 \;\; \forall\, 1 \leq b \leq |\mathfrak{B}_\iota|$: $\tilde{T}_{bp} \leftarrow \frac{\sum_{i \in B_b} \sum_{j : d_{jp} > 0} x_{ji} \mu_{ji} \frac{d_{jp}}{\lambda_j}}{|B_b|}$
20: $\delta \leftarrow \left| \frac{z_\iota - z_{\iota-1}}{z_\iota} \right|$
21: **end while**

In contrast to Algorithm 1, we allow infeasible solutions and, step by step, the solutions are pushed towards feasibility. The pseudo code of Algorithm 2 contains the details about the calculation of the resource consumption factors as well as the refinements of the resource buckets. We will now summarize the purpose of the steps of Algorithm 2:

- line 1–2: defines the input and output,
- line 3: sets the iteration counter ι to one,
- line 4–7: here we calculate the resource pools for the lower bound $\underline{\Delta}$ and initialize the resource buckets with the resource pools. The resource consumption factors (T_{ip} on equipment level) are calculated only for those products that are part of $\underline{\Delta}$,
- line 8–9: to calculate the resource consumption factors (T_{ip} on equipment level) for those products that are not part of $\underline{\Delta}$ we use $y = 1$ in the load balancing problem,
- line 10–12: in these steps we calculate the resource consumption factors ($\tilde{T}_{ip}$) on resource bucket level and we initialize the relative change δ which is used in the termination condition of the while loop,

- line 13: the while loop continues as long as we can see relevant changes in the objective value, but also if we detect a further refinement of the resource pools,
- line 14: incrementing the iteration counter ι,
- line 15: with respect to the actual resource consumption factors we are calculating the optimal product mix,
- line 16–19: according to the optimized product mix y we are updating the resource buckets and load factors, and finally
- line 20: updating δ.

Proposition 3. *Algorithm 2 is convergent with regard to the objective values and the solutions of Algorithm 2 are finally feasible.*

Proof. We start with showing the feasibility part: suppose that a solution y_ι is infeasible and there is one or more resource pools that exceed ρ_{max}. For simplicity we suppose that there is only one resource pool RP_k ($RP_k \in \mathfrak{R}$) that exceeds ρ_{max} then one of the following alternatives is true:

- A1: all elements of resource buckets B_b ($B_b \in \mathfrak{B}$) that are connected to RP_k are balanced,
- A2: at least one resource bucket B_b ($B_b \in \mathfrak{B}$) that is connected to RP_k is unbalanced.

In case of Alternative A1 the infeasibility is due to the underestimated resource consumption in a specific resource bucket. And according to the Algorithm we automatically adjust the matrix of resource consumption factors $\tilde{T}$ in the next iteration in such a way that the solution of the succeeding solution $y_{\iota+1}$ will be feasible for this bucket. For Alternative A2 the corresponding resource bucket is no longer homogenous and therefore it will be split according to the maximum resource pools $\mathfrak{R}^*$ and this kind of infeasibility cannot occur in the subsequent iterations. In order to argue that Algorithm 2 is convergent we note that the set of resource buckets $\mathfrak{B}$ will finally reach a segmentation of the equipment that will not be refined anymore (otherwise I is infinite) and therefore we can expect convergence. □

Remark 5. We note that we initialize the resource buckets in Algorithm 2 with the resource pools that correspond to the product mix $\underline{\Delta}$ and that there are several other alternative structures for the resource buckets to start with. We experimented with no or almost no initial structure, allowing the algorithm to build everything on its own. In this case we experienced that in some cases this approach was good, but in many others the quality of the solutions was very poor. To end up in a robust algorithm that gives high quality solutions the choice of the initial structure is therefore crucial.

Remark 6. To make the extended product mix model (14)–(25) fit to the decomposition algorithm we need to get rid of the variable x in the master problem. To do so, we can adopt (16)–(18) from the master problem in the following way:

$$Ty \le \rho_{max} + \overline{\rho}_1 + \overline{\rho}_2, \tag{16'}$$

$$(Ty)_i \le ([\rho_{max}]_i + [\overline{R}_1]_i + [\overline{R}_2]_i) w_i^{equ} \quad (\forall i \in I). \tag{18'}$$

Since the subproblem is identical we can use the identical framework. Later in Section 7, we will also discuss the sensitivity of the utilization with regards to the worst case behavior, and we indicate that this problem may also be solved within the decomposition framework.

To discuss the quality of the decomposition algorithms we will present the computational experiments for Algorithms 1 and 2 in Sections 5 and 6, respectively.

5. Design of experiments

We are investigating the effect of different factors of the instances on the potential improvement of the profit and the quality of the heuristic. For this reason we generated benchmark instances with different size and structure (the GNU Octave [8] code for the problem generator and the instances can be found at the following URL: http://homepage.univie.ac.at/martin.romauch/goldenmix/). Some of the control parameters are directly affecting the factors (like the number of products), but others (e.g. the density of the service time matrix) are dependent on several control parameters. Each instance contains an initial product mix y_0 that guarantees that at least one resource in each closed machine sets (*cms*) is loaded close to ρ_{max}. The demand limits are defined by a 5% corridor, i.e. $\underline{\Delta} = 0.95 y_0$ and $\overline{\Delta} = 1.05 y_0$.

To produce a structural broad set of instances of different sizes, we consider a factorial design based on the settings of the control parameters in Table 1. For instance, we are considering the number and the size of the closed machine sets (*cms*) because it is relevant for the decomposition into independent subproblems. Furthermore, we are considering the density of entries in A (e.g. if A is very dense then we have a large flexibility) and the variance of the resource consumption entries in R (e.g. if the variance is very small then the process time is independent on the resources) and we also let the size of the product spectrum vary.

For each parameter setting we use five different seeds. Therefore the total number of instances is $3^5 \cdot 5 = 1215$. To formulate the factors and variates we refer to the initial profit z_0, the profit of the global optimum z_{opt} and the profit of the decomposition method z_{decomp} (referring to Algorithm 1 and 2). Table 2 summarizes the factors and variates used in the analysis. In some places, we employ the coefficient of variation $CV(X)$ to normalize the variation. The $CV(X)$ is defined by $CV(X) = \sqrt{\mathbb{V}\mathrm{ar}(X)}/\mathbb{E}(X)$.

Since both Algorithms converged quickly, the tolerance limit for recognizing changes in the objective value was mainly set in compliance with our need for numerical stability. Since the default tolerance of the GLPK solver is $1e-7$ (tolerance for improvements in the objective) we have chosen $\epsilon = 1e-5$, which is 100 times larger. We experienced that this is sufficient to cover the cumulative error arising from a large number of resources and *cms*.

6. Results

Based on the experiments on the instances from Section 5 we are presenting our main results about the influence of the structure of the problem (parameters that describe the instance) on the profit increase and on the solution quality of the heuristic methods (Algorithms 1 and 2). Instead of giving a lower and upper bounds for the product mix we use a 5% corridor around y_0 – the corresponding maximum profit increase z_{opt} is used as a reference to measure the quality of the heuristic solutions, i.e. we are reporting the gap $z_{opt} - z_{decomp}$ relative to the best possible improvement $z_{opt} - z_0$ where z_0 denotes the initial profit. We will start with a summary of the main results of the experiment

Table 1
The parameters in this table are used to make the factorial design of the experiment.

Control parameter	Settings	Number of settings
cms size	Small, medium, large	3
Number of *cms*	Small, medium, large	3
Variance parameter for R	Small, medium, high	3
Density parameter for A	Sparse, medium, dense	3
Number of products	Small, medium, large	3

Table 2
The factors and variates in this table are used for the analysis. Since the problem is decomposed into sub components the letter τ indicates the corresponding *cms*. E.g. A_τ refers to the dispersal matrix of component τ.

Factor/variate	Formula/symbol	Values/range
Number of *cms*	Q	{20, 40, 60}
Number of resources	$neq = \|I\|$	[57;2349]
Number of job classes	$njc = \|J\|$	[74;1500]
Number of products	$nprod = \|c\|$	[5;1132]
Average density of $A_\tau = (a_{jk}^{(\tau)})$	$Density = \text{avg}_{\tau=1}^{Q}\left(\frac{\|\{k : a_{jk}^{(\tau)} > 0\}\|}{\|I_\tau\|\|J_\tau\|}\right)$	0–50%
Relative variance of service time	$rttvar = \text{avg}_j(CV(R_{j\|>0}))$	0-35%
Coefficient of variation of profit	$CV(c)$	11–36%
Maximum relative improvement	$Rel.\ potential = \frac{z_{opt} - z_0}{z_0}$	0–4.75%
Relative gap	$Gap = \frac{z_{opt} - z_{decomp}}{z_{opt} - z_0}$	0–100%

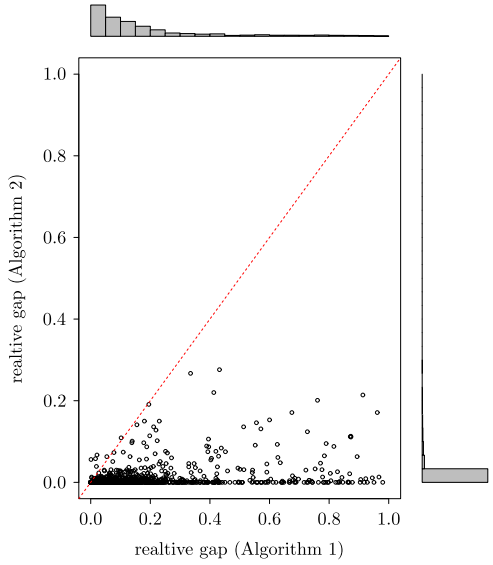

Fig. 6. All instances: for each instance, the relative gaps of Algorithms 1 and 2 are compared. The points below the dashed line are improvements. The class of exclusively larger instances (Q=60) is depicted separately.

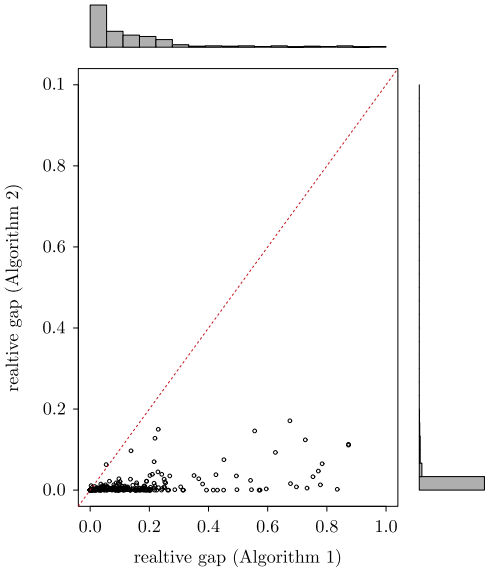

Fig. 7. Large instances (Q=60): for each instance, the relative gaps of Algorithms 1 and 2 are compared. The points below the dashed line are improvements. The class of exclusively larger instances (Q=60) is depicted separately.

(Figs. 6–9). Then we discuss some structural influences on the relative potential (cf. Figs. 10 and 11) and in Section we will also showcase a short study with real world data that also considers sensitivity issues.

We used different frameworks in our implementations, for computing the optimal solution of the product mix problem (global approach) we used ILOG CPLEX from IBM [17], and for the decomposition algorithms we used GNU Octave [8] employing the GNU Linear Programming Kit [21] for solving the LP problems. A quad core (2.6 GHz) computer with 4 GB RAM was used for the experiments and we note that in some cases (especially for large problem instances) GLPK was not able to find the optimal solution of the global problem. With respect to the benchmark instances the simple decomposition approach (Algorithm 1) always converged after five or less iterations (less than five minutes computational time) and in average resulted in solutions that take 82% of the maximum possible profit (average relative gap 17%), approximately 12% of the instances were optimally solved. We note that in some cases the relative gap was considerably high. Algorithm 2 converges after ten or less iterations (less than five minutes computational time) and reaches an average gap of less than 1%, while solving 57% of the instances optimally. Tables 3 and 4 give some further statistics about the relative gap of the two methods.

In Table 3 we compare the relative frequency of the relative gap of both Algorithms 1 and 2. The first two columns contain the lower and the upper bound of the interval for the relative gap. We can see that in more than 97% of the cases Algorithm 2 has a relative gap below 10% whereas it is only 50% of the instances in case of Algorithm 1. In case of Algorithm 1 the worst case is almost a 100% gap, whereas it is 27% in case of Algorithm 2.

In Fig. 6 we compare the relative gap for each instance. Since almost all points are located below the dashed red line we can see that Algorithm 2 improves the results of Algorithm 1 with only a few exceptions (i.e. in eight cases the basic algorithm gives better results). On the upper margin and also on the right margin of Fig. 6 you can also find histograms of the relative gaps for the respective algorithms, they demonstrate the qualitative differences of the shapes of the distributions.

Fig. 7 holds the same type of information, but limited to larger instances (where Q=60). It seems that both methods, Algorithms 1 and 2, perform better on larger instances.

In Fig. 8 we directly compare the cumulative distribution functions. The color blue is used for the results of Algorithm 2 and black indicates the results of Algorithm 1. The blue line is significantly above the black one, which also shows the benefit of Algorithm 2.

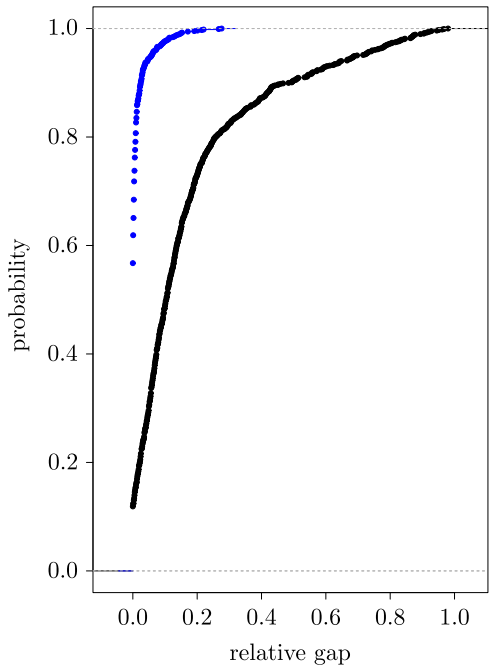

Fig. 8. The cumulative distribution function of the relative gap $\mathbb{P}(gap \leq x)$ for Algorithm 1 is shown as a black line. For Algorithm 2 we use the color blue. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

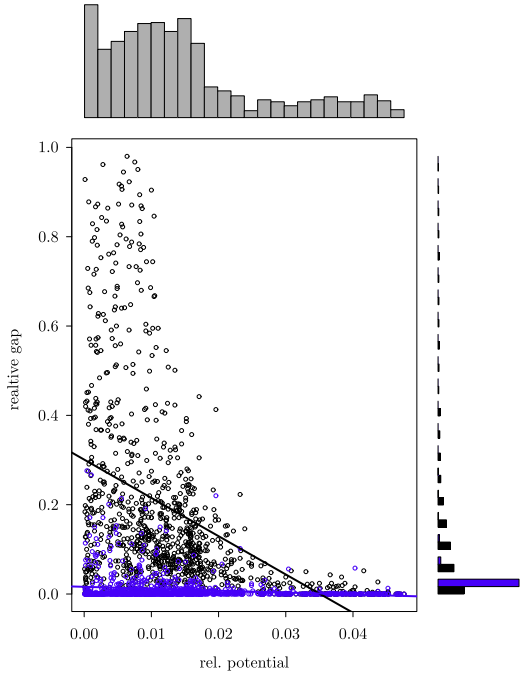

Fig. 9. Each black point represents the results for a single instance of Algorithm 1. The results for Algorithm 2 are indicated by blue points. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

We also investigated the relative potential and the relative gap (cf. Table 2) for correlations. In Fig. 9 we directly compare the influence for Algorithm 1 (black) and Algorithm 2 (blue). In case of Algorithm 1 we can see that – with an increasing optimization potential – the decomposition heuristic gradually approaches to the global optimum. There is also a very light tendency regarding a correlation for Algorithm 2, but it is vanishingly small.

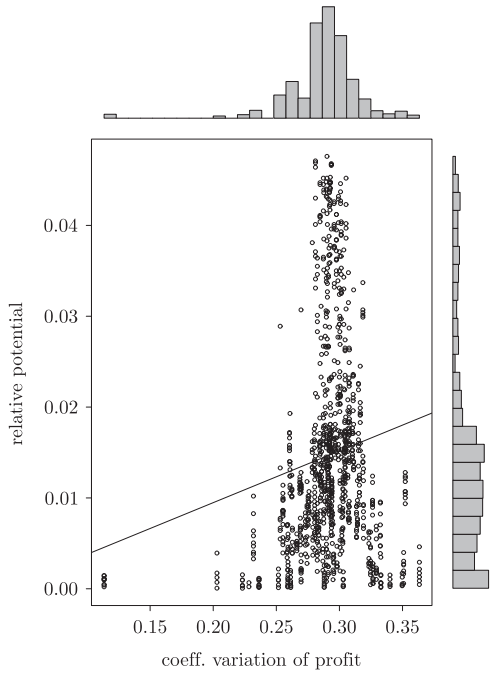

Fig. 10. The relative potential tends to be smaller if the coefficient of variation is small.

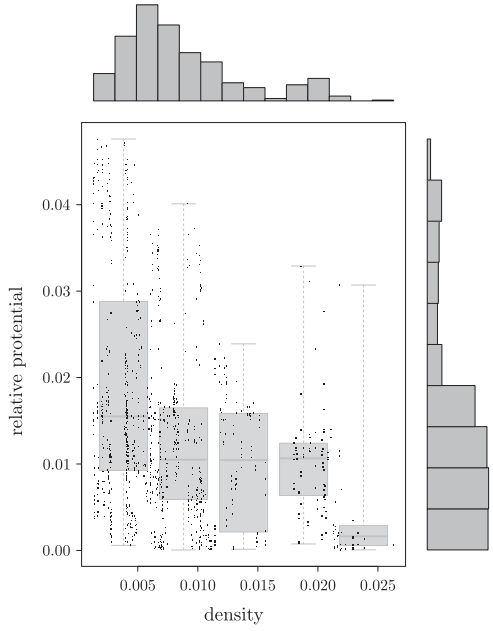

Fig. 11. High values for the relative potential are more likely if the density is smaller.

To discuss dependencies of parameters varied in the benchmark scheme on the optimization potential we give two examples. First, we investigate the influence of the structure of A by considering the density of nonzero elements in the matrix A on *cms* level (cf. *density* in Table 2) and then we discuss the influence of variation of the profit of products.

In Fig. 10 the relative potential and the coefficient of variation of the profit have a positive correlation. We can also observe that the marginal distribution of the profit is bell shaped. This is due to the generation of benchmark instances where the entries of the profit vector c are modeled as independent and uniformly distributed random variables. According to the central limit

Table 3
Comparison of the relative frequency of the relative gaps of Algorithms 1 and 2 in percent.

lb	ub	Algorithm 1 (%)	Algorithm 2 (%)
0.00	0.05	29.89	94.59
0.05	0.10	18.68	3.15
0.10	0.15	14.85	1.38
0.15	0.20	10.13	0.39
0.20	0.25	5.90	0.29
0.25	0.30	3.05	0.20
0.30	0.35	2.75	0.00
0.35	0.40	2.06	0.00
0.40	0.45	2.36	0.00
0.45	0.50	0.88	0.00
0.50	0.55	1.08	0.00
0.55	0.60	1.47	0.00
0.60	0.65	0.98	0.00
0.65	0.70	1.08	0.00
0.70	0.75	0.88	0.00
0.75	0.80	1.18	0.00
0.80	0.85	0.88	0.00
0.85	0.90	0.79	0.00
0.90	0.95	0.69	0.00
0.95	1.00	0.39	0.00

Table 4
Statistics for the relative gaps of Algorithms 1 and 2.

Statistic	Algorithm 1	Algorithm 2
Min	0.0000	0.000000
1st Qu.	0.0360	0.000000
Median	0.1040	0.000000
Mean	0.1753	0.009827
3rd Qu.	0.2110	0.006000
Max.	0.9800	0.276000

theorem, the sum of squares tends to a normal distribution, and therefore we observe (cf. Fig. 10) that the profit's marginal distribution of the coefficient of variation is approximately Gaussian. When having a look at the points in Fig. 11 we can find some evidence for a dependency of the relative potential and the density. Applying the Kolmogorov–Smirnov test on the first half of samples (relative potential for a density smaller than 0.012) and the second half (relative potential for a density ≥ 0.012) gives a p-value of $2.2e-16$. The boxplots in the background indicate a tendency for instances with a low density to have a larger relative potential. The reason for that is probably the more complicated structure of the problem. The likelihood of finding an initial product mix whose resource consumption is suboptimal might be higher – and therefore the degree of freedom could be used more efficiently to improve the solution.

7. Practical experiences

In the last section we have seen that a profit improvement of up to 4% is possible even if each component in the initial solution is loaded close to ρ_{max}. If the product mix portfolio is adaptable in the range of $+/-5\%$ this also matches with our practical experiences based on real–world data.

Next to the profit mix optimization, some extensions of the model (cf. Remark 2) are very helpful for various use cases in daily business. Questions concerning product mix robustness can be answered, proposals for ramp- or downturn scenarios (e.g. cold steel) can be derived, machine qualification schemes (tool dedications) can be proved and optimized, and bottlenecks can be identified and eliminated. The key for the usability and flexibility of the modeling framework lies in the decomposition (job class concept, connected components). The method allows a calculation on the highest granularity level – the MES level. Therefore, all routes, all products, all operations, all resources and all dedications in the manufacturing system can be considered. Assuming that the MES information is always up to date and accurate this modeling approach bypasses solutions with separately aggregated, maintained and validated meta-data-layers. The decomposition avoids losing any detail and performs with an overall calculation time of only a couple of minutes.

Without discussing details of the models or the fab data lying behind them, Figs. 12 and 13 should give a brief impression for two use cases. Both figures show the solution of a product mix optimization on the basis of an altered fab-data set. The x-axis refers to the connected components (typically work centers) and on the y-axis measures the utilization. In Fig. 12 the red bar in the center of the blue bars indicates the maximum tool utilization within the work center resulting from the current (given) product mix. The average utilization is marked by a black disk and if the black and the white mark do not match, different RPs must exist within the work center. The first use case (drafted in Fig. 12) is about how to deal with unstable demands: let us imagine that y is a demand proposal and $\underline{\Delta} \leq y \leq \overline{\Delta}$ are boundaries of the projected demand. Then it would be interesting to know how sensitive some (critical) work centers will react on changes in terms of a best-/worst mix scenario. Because we have a direct interaction between product mix and the related equipment utilization profile we are able to answer this question by calculating sensitivity intervals. In Fig. 12 different sensitivity intervals are displayed in different shades of blue.

The sensitivity analysis helps us to identify products which are drivers for bottlenecks and to estimate how the equipment will react on product changes.

A second use case is shown in Fig. 13. Here the (minimum) profit is set as a bound in the model. Also the overall fab load (layer starts per week) is not changed or fixed to a certain value. The task is to calculate a product mix adaption that significantly reduces the bottleneck utilization in the fab. The result is shown in Fig. 13, where the arrows indicate the change of the average and maximum utilization. As one can see, the maximum bottleneck utilization is reduced drastically. Thereby, the fab throughput is not changed – but the product mix.

8. Conclusions and further research

The contribution of this paper can be seen as twofold. On one hand the theoretical part: we have investigated which structural parameters affect the maximum profit improvement of the product mix problem. Here we could show that the number of products and the variation of the profit are affecting the relative potential in a significant way. Furthermore, we have introduced two decomposition heuristics for the product mix problem and we have inspected them on the basis of randomly generated test instances. Both approaches solve a sequence of master problems that rely on resource consumption estimates that are successively improved in the corresponding sub problem. Thereby, the second approach is an advancement of the first version presented in a former paper. This new (resource pool based) decomposition outperforms the previous version in nearly every experiment. So, we were able to prove that this new method converges to good quality solutions within reasonable time. Due to its reduction of complexity it is also interesting that we could show that within this concept already free solvers like GLPK are sufficient to solve large instances. We also want to note that mixed integer extensions, like

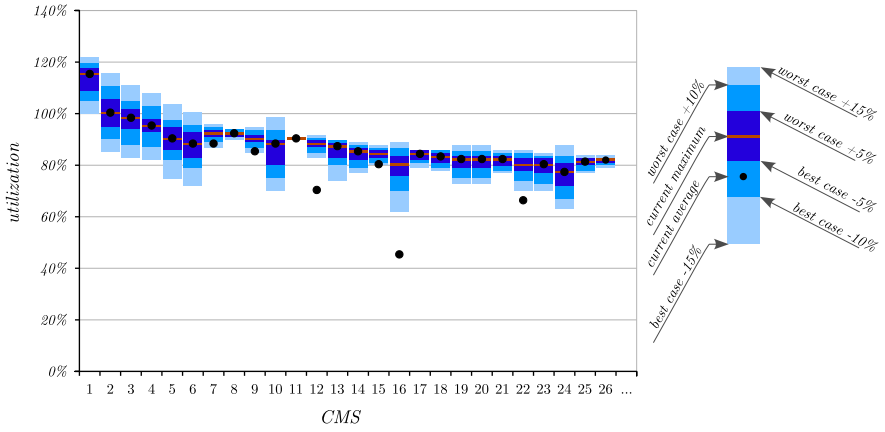

Fig. 12. Sensitivity analysis – effects of product mix fluctuations (proposed mix).

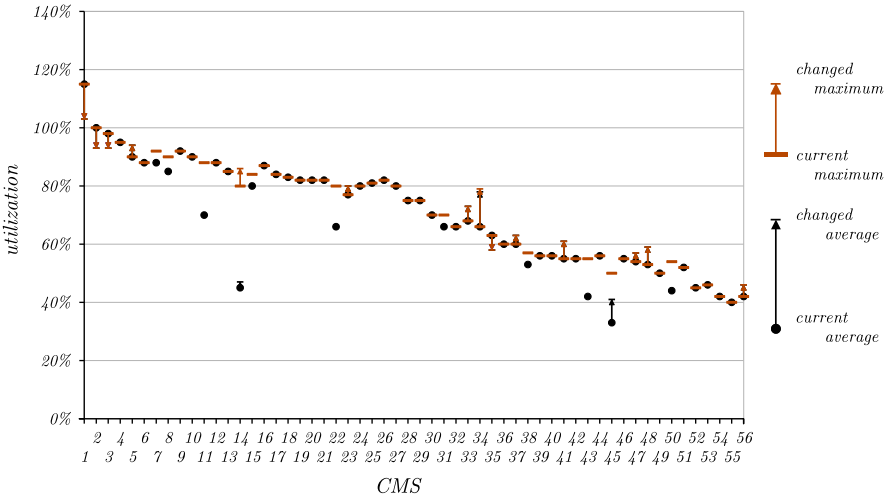

Fig. 13. Optimized product mix – product mix change relaxes the bottleneck situation.

fixed costs for products (compare (17) and (24)) which only affect the master problem, are also suitable for the proposed decomposition approach.

On the other hand the practical part: we drafted, that the approach covers numerous problems within semiconductor industry. The method helps in evaluating risks, improving lot release strategies and supports bottleneck management. Furthermore, based on more accurate data, sensitivity analyses and proposals for capacity investment decisions are available more quickly.

In future we will focus our work on optimizing the whole supply chain within a semiconductor company including capacity invest and product mix decisions. In combination with dynamic techniques this method will support an optimized product flow and reduce logistics costs within the supply chain.

References

[1] Aurand SS, Miller PJ. The operating curve: a method to measure and benchmark manufacturing line productivity. In: Advanced semiconductor manufacturing conference and workshop. IEEE/SEMI, IEEE; 1997. p. 391–7.

[2] Bang JY, Kim YD. Hierarchical production planning for semiconductor wafer fabrication based on linear programming and discrete-event simulation. IEEE Trans Autom Sci Eng 2010;7:326–36.

[3] Banks J, Carson JS, Nelson BL, Nicol DM. Discrete-event system simulation. 3rd ed. Upper Saddle River, New Jersey: Prentice-Hall Inc.; 2000.

[4] Barahona F, Bermon S, Gunluk O, Hood S. Robust capacity planning in semiconductor manufacturing. Nav Res Logist 2005;52:459–68.

[5] Bermon S, Hood S. Capacity optimization planning system (CAPS). Interfaces 1999;29(5):31–50.

[6] Bratley P, Fox BL, Schrage L. A guide to simulation. New York: Springer-Verlag; 1983.

[7] Chou YC, Hong IH. A methodology for product mix planning in semiconductor foundry manufacturing. IEEE Trans Semicond Manuf 2000:278–85.

[8] Eaton JW, Bateman D, Hauberg S. Gnu octave manual version 3; 2008. ⟨http://www.octave.org⟩.

[9] Geng N, Jiang Z. A review on strategic capacity planning for the semiconductor manufacturing industry. Int J Prod Res 2009;47(13):3639–55.

[10] Gold H. Dynamic optimization of routing in a semiconductor manufacturing plant. In: Hein Fleuren, Dick den Hertog PK. (Ed.), Operations research proceedings 2004: selected papers of the annual international conference of the GOR; jointly organized with the NGB, GOR and NGB, Tilburg, The Netherlands; 2004. p. 76–83.

[11] Gold H. Lastverbund, European patent EP 1567921 (German), July 2012; 2008 ⟨https://data.epo.org/publication-server/rest/v1.0/publication-dates/20081112/patents/EP1567921NWB1/document.html⟩.

[12] Habla C, Mönch L. Solving volume and capacity planning problems in semiconductor manufaturing: a computational study. In: Mason SJ, Hill RR, Mönch L, Rose O, Jefferson T, Fowler JW. (Eds.), Proceedings of the 2008 winter simulation conference; 2008. p. 2260–6.

[13] Hager V, Spannraft K. A product mix and a material flow problem concerning the semiconductor manufacturing industry [Master thesis]. Department of Business Administration, University of Vienna, Vienna, Austria; 2009. Available via ⟨http://othes.univie.ac.at/8092/⟩[accessed 30.05.12].

[14] Harrison JM, Lopez MJ. Heavy traffic resource pooling in parallel-server systems. Queueing Syst 1999;33:339–68.

[15] Harrison JM, Williams RJ. Workload interpretation for Brownian models of stochastic processing networks. Math Oper Res 2007;32(4):808–20.

[16] Hood SJ, Bermon S, Barahona F. Capacity planning under demand uncertainty for semiconductor manufacturing. IEEE Trans Semicond Manuf 2003;16:273–80.

[17] IBM. Cplex: 12.2 user's manual; 2010.
[18] Klemmt A, Horn S, Weigert G, Wolter KJ. Simulation-based optimization vs. mathematical programming: a hybrid approach for optimizing scheduling problems. Robot Comput-Integ Manuf 2009;25:917–25.
[19] Klemmt A, Romauch M, Laure W. Product mix optimization for a semiconductor fab: modeling approaches and decomposition techniques. In: 2012 Proceedings of the winter simulation conference; 2012. p. 187.
[20] Law AM, Kelton WD. Simulation modeling and analysis. 3rd ed. New York: McGraw-Hill Inc.; 2000.
[21] Makhorin A. Gnu linear programming kit (version 4.34); 2012. ⟨http://www.gnu.org/software/glpk/glpk.html⟩.
[22] Mönch L, Fowler JW, Dauzère-Pérès S, Mason SJ, Rose O. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. J Sched 2011;14:583–99.
[23] Ponsignon T, Mönch L. Heuristic approaches for masterplanning in semiconductor manufacturing. Comput Oper Res 2012;39:479–91.
[24] Rose O. General simulation applications in semiconductor manufacturing: why do simple wafer fab models fail in certain scenarios? In: Proceedings of the 32nd conference on winter simulation, Society for Computer Simulation International, San Diego, CA, USA; 2000. p. 1481–90.
[25] Zisgen H, Meents I, Wheeler BR, Hanschke T. A queueing network based system to model capacity and cycle time for semiconductor fabrication. In: Winter simulation conference; 2008. p. 2067–74.
[26] Gold H. Why our company uses programming languages for mathematical modeling and optimization. In: J K, editor. Modeling Languages in Mathematical Optimization. Berlin, Heidelberg: Springer-Verlag; 2012. p. 161–9.