Performance quantification and simulation optimization of manufacturing flow lines

Performance Quantification and Simulation Optimization of Manufacturing Flow Lines

Johannes Henricus Jacobs

Voorkant: Artistieke weergave van Weverijmuseum Geldrop. Het museum heeft het doel om "een bewogen textielverleden in Zuidoost Brabant levend te houden". Sinds 2000 is het museum gevestigd in de oude spinnerijhal uit 1908 van de Wollenstoffenfabriek van den Heuvel. Dit gebouw in nu een rijksmonument. Meer informatie is beschikbaar op `http://www.weverijmuseum.nl/`.
Het typische sheddak wordt gebruikt in fabriekshallen. De ramen zijn altijd gericht op het noorden en zorgen voor een regelmatige lichttoevoer.

Cover: Artistic impression of the weaving museum in Geldrop, The Netherlands. The museum aims to "keeping fresh the memory of the textile industry in south-east Brabant". The building is an old textile mill and is currently a national monument. More information is available on `http://www.weverijmuseum.nl/`.
The typical shed roof is used for industrial buildings. The windows always face north for optimal daylighting performance.

This work was sponsored by the Dutch Technology Foundation STW, Applied Science Division of NWO, and the Technology Program of the Ministry of Economic Affairs of The Netherlands.

This work has been carried out under the auspices of the Engineering Mechanics research school.

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Jacobs, Johannes H.

Performance quantification and simulation optimization of manufacturing flow lines / by Johannes H. Jacobs. – Eindhoven : Technische Universiteit Eindhoven, 2004.
Proefschrift. – ISBN 90-386-2626-6
NUR 804

Subject headings: manufacturing systems / manufacturing flow lines / manufacturing performance quantification / Sequential Approximate Optimization / discrete-event simulation / queueing theory / factory physics / semiconductor industry

Reproduction: Universiteitsdrukkerij Technische Universiteit Eindhoven

Performance Quantification and Simulation Optimization of Manufacturing Flow Lines

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. R.A. van Santen, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 13 september 2004 om 16.00 uur

door

Johannes Henricus Jacobs

geboren te Harderwijk

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. J.E. Rooda
en
prof.dr.ir. F. van Keulen

Copromotor:
dr.ir. L.F.P. Etman

Summary

This PhD project was part of the ADOPT-project funded by the Dutch Technology Foundation STW. ADOPT is a joint effort of research activities at the Eindhoven University of Technology and the Delft University of Technology. The ADOPT-project aims to develop numerical optimization techniques for engineering design problems with a computationally expensive analysis model in the loop. The acronym ADOPT stands for Approximate Design OPTimization referring to the optimization approach followed in the project. A sequence of approximate optimization subproblems is generated and solved. This Sequential Approximate Optimization (SAO) approach avoids a direct coupling between the computationally expensive analysis model and the evaluation intensive optimizer. Through the approximations specific problem properties observed in certain application domains can be taken into account. In this regard, the ADOPT-project considers the treatment of stochastic and noisy responses, discontinuities in responses, uncertain design variables, and integer design variables, as well as the application to a range of mechanical engineering design problems.

The contribution of the thesis is threefold. First, a software framework for sequential approximate optimization has been developed in the scope of the ADOPT-project. The framework provides an open environment for the specification and implementation of SAO strategies. This enables one to re-define the SAO strategy and to tailor the approximations such that they match well with the problem properties at hand. Secondly, approximation methods are developed specifically for simulation-based optimization of manufacturing flow lines. The analysis model involved in this type of problems is a discrete-event simulation model. Optimization properties that arise relate to the stochastic responses from the simulation model and the integer design variables, such as the numbers of machines and batch sizes. Thirdly, the thesis contributes on the development of the so-called Effective Process Time (EPT) approach. EPT is a means to characterize flow time performance from a queueing physics point of view. The new metric gives the opportunity to arrive at simple but accurate simulation models since it avoids the detailed modeling of all kinds of processing disturbances present at the shop floor.

The framework for sequential approximate optimization has been developed using an object-oriented class structure and contains a toolbox of methods, (external) numerical routines, and interfaces to other software. The framework enables to specify (i) the optimization problem, including the simulation model for the evaluation

of the objective function and constraints, (ii) the SAO sequence defining the order of computational steps, and (iii) the numerical routines used in each step. A typical SAO sequence consists of a number of computational steps, regarding to, e.g., the design of experiments, the approximation building, and the approximate optimization solving. The numerical routines used in the SAO steps are represented as 'black-box' functions, e.g. from external software libraries. The framework enables the user to (re-) specify or extend the SAO sequence and computational steps, which is generally not possible in most available SAO implementations. A ten-bar truss design problem with fixed loads and uncertain loads demonstrates the flexibility of the framework. Within the ADOPT-project, the SAO framework has been applied to a broader range of optimization problems, including optimization of Microelectromechanical Systems (MEMS) and manufacturing systems, and optimization for robustness and reliability.

The Effective Process Time (EPT) approach is a means to characterize capacity and variability in workstations of manufacturing systems. The concept of EPT has already been formulated in work of others, but no method has been proposed to actually measure EPT in operating factories or simulation models. EPT is an overall measure that includes process time of a product and process disturbances, such as machine failure, setup times, and operator availability. The aim is to develop algorithms which compute the EPT capacity and variability measures based on data from the manufacturing system. The computed EPT values can be used as performance measure that relate to the queueing physics of workstations and indicate the flow time performance of a manufacturing system. Algorithms have been developed for workstations that manufacture single products and workstations that manufacture products in batches. Both types of workstations are commonly used in semiconductor manufacturing. The EPT algorithms have been tested using small examples of simulation models and have been applied on operational data of a Philips Semiconductor wafer fab. This application resulted in simulation meta models of each workstation, which may be combined into a meta model of the complete factory.

Sequential approximate optimization of manufacturing systems requires good quality approximation models. In this thesis manufacturing systems are optimized for flow time performance. Queueing theory shows that flow time behaves highly non-linear for capacity-related design variables. The newly proposed approximation functions developed for the SAO method are able to characterize these non-linearities. In earlier work the use of linear regression models was suggested. In this thesis, the method is extended and generalized using the concept of EPT. With the proposed EPT parameters available, the flow time performance of a workstation can be characterized. The idea is to use the EPT parameters in the flow time approximation model and include queueing physics in the response surfaces. The parameters of the response surface model are estimated based on simulation responses of the discrete-event model. This new type of approximation has been implemented in the SAO framework. The proposed optimization approach has been successfully tested on two simulation-based design problems: a four-station flow line and a twelve-station re-entrant flow line. Both pure integer and mixed-integer cases were considered and the optimization problems included up to twelve design variables.

Contents

Chapter 1

Introduction

Manufacturing systems are becoming increasingly complex. Especially in technologically advanced industries, manufacturing is characterized among others by many process steps, various products types, and advanced process equipment. An example of one of the world's most complex manufacturing systems is found in semiconductor industry. The production of Integrated Circuits (ICs) is a large-scale operation and consists of hundreds of process steps. The IC production facility as discussed in Van Campen (2001) is a multi-process multi-product wafer fab producing about 10 different main IC product types (technologies). Due to the high cost of equipment, it is not possible to build one long manufacturing flow line with a machine for each process step. Instead, a re-entrant flow line is used for the production of ICs, gathering equipment of the same functionality into workstations. Each workstation is typically related to one of the process operations, such as lithography, implantation, etching, oxidation, and deposition. The wafers with ICs revisit the workstations multiple times during their flow through the production facility.

Semiconductor Manufacturing

An IC consists of millions of microscopic electronic components, mainly transistors, as well as passive components like resistors. An IC is built up from several layers produced on a silicon wafer. Each separate layer has different electrical properties and a different geometric pattern representing the circuit design information. Each layer itself is created by a sequence of operations. A number of these (more or less the same) sequences is needed to complete a working IC. The electrical properties and patterns are created by adding, altering, and removing material in each layer. The basic operation that transfers the geometric pattern information to the wafer surface is lithography.

The manufacturing system design aims at maximizing output, yield, delivery performance, and flexibility and at minimizing flow time to obtain maximal revenues. This thesis focuses on optimization for flow time performance. Understanding the *factory physics* of the discrete-event manufacturing system plays a key role here. Helpful tools for the analysis and the (re-) design of manufacturing systems

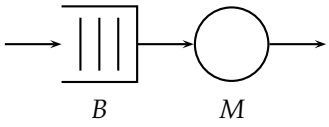

Figure 1.1: A workstation with one buffer B and one machine M.

are analytical models as well as simulation models. These models are used to predict manufacturing system performance regarding operational or design decisions.

Much effort is done to reduce *flow times* in manufacturing systems, especially in make-to-order environments. Flow time is the total amount of time from release of the product at the beginning of the flow until the product reaches the end of the flow. In a real-world factory, flow time of a product can be identified by measuring the time that the product enters the line and the time the product departs (is finished). This product flow time is the sum of flow times accumulated at each (re-entrant) visit of a workstation. The workstation flow time includes waiting time to be processed and process time.

Two other important measures are *throughput* and *work-in-progress*. Throughput is the amount of products produced per time unit and work-in-progress is the total amount of products in the factory. Together with flow time, these measures are related to *factory physics* following Little's law (Little, 1961):

$$w = \varphi \cdot \delta \tag{1.1}$$

where w is the mean work-in-progress, φ is the mean flow time, and δ is the mean throughput. This equation relies on the precondition of product conservation, that is, the amount of non-manufactured products that enter the factory should on the long run be equal to the amount of manufactured products that leave the factory (steady-state).

1.1 Why do queues arise?

"Variability causes congestion." This statement from the textbook of Hopp and Spearman (2001) represents another issue of factory physics. We will illustrate this statement by a very simple example of a workstation with one infinite buffer B representing the queue and one machine M representing the server (see Figure 1.1). The buffer is the waiting line where arriving parts have to wait before they can be manufactured by the machine. It is assumed that the parts are served on first-come-first-served basis. We will now discuss the deterministic case in which there is no variability, followed by the stochastic case in which there is variability.

Suppose parts arrive at a pace of 12 parts/hour, i.e. the interarrival time of the parts is 5 minutes. Suppose that the maximum capacity of the machine is 15 parts/hour, which means that the machine is able to complete one part in 4 minutes. Assume that *exactly* every 5 minutes a part arrives, and that each part is processed for *exactly* 4 minutes. As soon as a part arrives at the queue (which is empty), the part

will immediately be processed for 4 minutes. The machine now has to wait for one more minute for the next part to arrive. That part, again, arrives at an empty queue. In the end, none of the arriving parts has to wait in line for an other part that is still being processed and the average waiting time is 0 minutes.

Now suppose variable process times, for example, 50% of the parts needs processing for exactly 1 minute (product type A), and the other 50% needs processing for exactly 7 minutes (product type B). The average process time is still 4 minutes, but now it has become stochastic. Assume the same arrival pace of exactly 12 parts/hour, but the arrival stream is a random mix of product type A and product type B. Due to this random arrival mix, it might happen that two or more parts of product type A arrive successively. In that case none of the parts has to wait, since the process time is just 1 minute and each product has finished processing before the next part arrives. It might also happen that two or more parts of product type B arrive successively. In this case a queue arises, because process time takes 7 minutes. Thus after 5 minutes, the next arrived part of type B has to wait for 2 minutes. The waiting time further increases when the following part is again of type B. As a consequence, the *average* waiting time is always larger than 0 minutes, i.e. a queue arises.

Variability in the example relates to the processing times at the machine. Generally, also variability regarding interarrival times is present. Thus we have both an interarrival time *distribution* and a process time *distribution*. Considering the first two moments, the interarrival time distribution can be represented by mean interarrival time t_a and standard variance σ_a. The same holds for the process times; the process times are distributed with mean effective process time t_e and standard deviation σ_e. A dimensionless measure of variability c can be defined to interpret the amount of variability: $c_a = \sigma_a/t_a$ for the interarrival time distribution and $c_e = \sigma_e/t_e$ for the process time distribution. In the manufacturing system context Hopp and Spearman (2001) identify the following three categories:

- $c < 0.75$: low variability
- $0.75 < c < 1.33$: moderate variability
- $c > 1.33$: high variability

Exponential distributed process times or Poisson arrivals correspond with a coefficient of variation of 1.0, an assumption that is frequently made in mathematical queueing models (Kleinrock, 1975).

With t_e, c_e, t_a, and c_a available, queueing theory can be used to determine the expected mean waiting time over the long run. The mean waiting time φ_q in a single-server workstation with generally distributed arrival times and process times (i.e. no assumption is made regarding the shape of the distribution) can be approximated by (see, e.g., Hopp and Spearman, 2001):,

$$\varphi_q = \frac{c_a^2 + c_e^2}{2} \frac{u}{1-u} t_e \tag{1.2}$$

where $u = t_e/t_a$ is the utilization. Utilization is a measure to indicate how busy the machine is, i.e. the fraction of time a machine is busy processing. Utilization u is

bounded by the range $0.0 \leq u < 1.0$. If $u \geq 1.0$ the workstation is not stable and the queue length and waiting time will go to infinity. From (1.2) we can learn that for high levels of variability and high levels of utilization, the system will show long queue times. For systems with low variability and low utilization, queueing time is short. Furthermore, mean flow time increases in a non-linear fashion for increasing utilization towards the asymptote at $u = 1.0$.

1.2 Analysis of flow time performance

Variability and utilization are the main drivers for flow time. In fact, in real-world factories there are lots of reasons why variability in arrivals and processing occur. Typically, variability in processing is due to disturbances such as:

- failure of machines,
- operator availability,
- setup times,
- maintenance,
- starvation of material, and
- different product types.

All these disturbances are events that happen in reality at the shop floor. The disturbances cause loss of capacity, i.e. affect utilization, and cause variability in process times. Variability in processing implies variability in interdeparture times which causes variability in arrivals at the next workstation.

To understand flow time performance of an operating factory, both capacity losses and variability should be quantified at each of the workstations. In industry it is therefore common practice to measure important disturbances such as time to failure and corresponding time to repair. A typical performance measure used in industry to quantify capacity losses is the Operational Equipment Effectiveness (OEE), which relates to the time a machine is effectively processing products (Nakajima, 1988; SEMI, 2000). It is however remarkable to observe that no measure for variability is available. Adding up the contributing sources of variability at a workstation is not straightforward, since it requires the identification of all sources of variability. Unfortunately, this means that all, or at least the most important disturbances have to be measured separately. This is a difficult task, since the number of different disturbances can be large and some disturbances may even be unknown. By only focusing on utilization one may overlook the opportunities for performance improvement by reducing variability.

When considering changes in the manufacturing system design or designing a new one, methods for flow time prediction are needed. Two main fields of analysis methods can be identified: analytical models and simulation models. In queueing theory many analytical models of queueing networks have been developed for the

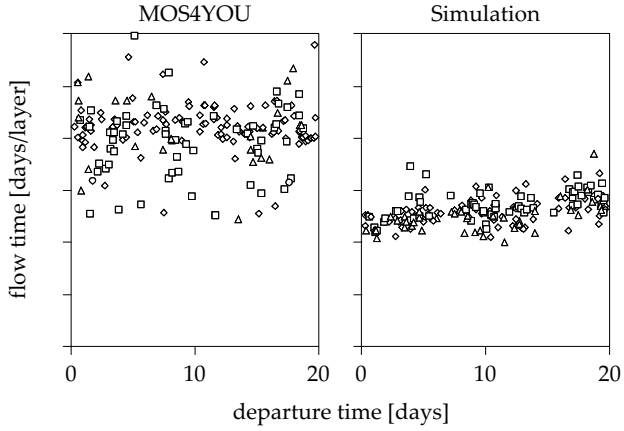

Figure 1.2: Real-life flow times vs. simulation flow times.

analysis of manufacturing systems. Examples of such models are queueing models, such as (1.2), that predict mean flow time. These models are well designed to describe the physics of manufacturing systems. However, the scope of these models is restricted by assumptions that have to be made in the development of the model. Furthermore, the variety of shop floor realities that can be covered is limited. For example, no mathematical equations are available to describe the flow time behavior of batching equipment with recipe-dependent batch forming rules.

This thesis focuses on the second option to use a discrete-event simulation model to predict flow time in manufacturing systems. The method of discrete-event simulation is much more suited to incorporate all kinds of shop floor realities in the model. The simulation simply mimics the flow of parts through the manufacturing system and may include all sorts of events related to disturbances the modeler deems important. Here again, the difficulty arises that it is impossible to include all shop floor realities. Although a lot of shop floor detail can be included, it may still be difficult to get all the appropriate factory data, to fit appropriate distributions to the data, and to validate the simulation models. This was also observed by Van Campen (2001) as described in the example below.

Modeling of MOS4YOU wafer fabrication

During the study of Van Campen (2001) discrete-event simulation studies in the context of the design and the operation of the Philips Semiconductor factory MOS4YOU were carried out. One simulation study dealt with the modeling of the flow of wafers. A bottom-up approach was used such that all individual processes and the most important sources of process disturbances were modeled. The discrete-event simulation model included (i) 161 production machines, (ii) the routings of all important production types, (iii) normal production lots and priority lots, (iv) failure of equipment, and (v) hold lots. All parameters in the model were determined using available historical data from the manufacturing

execution system. Although the model was detailed and the product flow was a good representation of MOS4YOU, the estimated mean flow time by the simulation was only half of the real flow time of the MOS4YOU wafer fabrication facility as shown in Figure 1.2. Van Campen concluded that the amount of variability present in the model was not sufficient. He suggested to further identify and include other disturbances.

1.3 Optimization with a discrete-event simulation model

With a simulation model available, it is possible to pose what-if questions related to relevant design decisions. This can be done by performing simulations for various settings of the design parameters. For a small number of design variables, this search for the best possible parameter setting can be done rigorously by exhaustive search and simulating a grid of design points. For a larger set of design variables, this cannot be applied especially when the simulation model is needed for flow time prediction. A simulation model typically requires a considerable amount of time (may range in the order of minutes to hours), which limits the number of simulation evaluations that can be carried out.

Simulation optimization is a means to find an optimal set of parameters for computationally expensive simulation models. The design of manufacturing systems can be formulated as an optimization problem. The following constrained optimization problem is considered:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{r}(\mathbf{x}), \mathbf{x}), \\ \text{subject to} & \mathbf{g}(\mathbf{r}(\mathbf{x}), \mathbf{x}) \leq \mathbf{0}, \text{ and} \\ & \mathbf{x}_\ell \leq \mathbf{x} \leq \mathbf{x}_\mathrm{u} \end{array} \tag{1.3}$$

with $\mathbf{x}$ the vector of design variables, $f(\mathbf{r}(\mathbf{x}), \mathbf{x})$ the objective function, and $\mathbf{g}(\mathbf{r}(\mathbf{x}), \mathbf{x})$ the vector of constraint functions. Design variable x_i belongs to the vector of design variables $\mathbf{x}$, with $i = 1, \ldots, n$ and n the number of design variables. Vectors $\mathbf{x}_\ell$ and $\mathbf{x}_\mathrm{u}$ represent the lower and upper bounds of the design space for each of the design variables in $\mathbf{x}$. Constraint g_j belongs to the set of constraint functions $\mathbf{g}$, with $j = 1, \ldots, q$ and q the number of constraint functions.

In this thesis it is assumed that a discrete-event simulation model is used to compute responses $\mathbf{r}$ for certain design variable values $\mathbf{x}$. The responses of the simulation model are used to evaluate objective function $f(\mathbf{r}(\mathbf{x}), \mathbf{x})$ and constraint functions $\mathbf{g}(\mathbf{r}(\mathbf{x}), \mathbf{x})$. Objective f and constraints $\mathbf{g}$ are functions of the simulation responses and/or the design variables itself. Some functions may be fully explicit relations that can be calculated without using the computationally expensive simulation model. But as long as the simulation model is in the loop for response computation for at least one objective or one constraint, optimization of manufacturing systems is a computationally expensive task in which the number of simulation evaluations has to be limited. A decision support tool for the (re-) design of manufacturing systems would be helpful here.

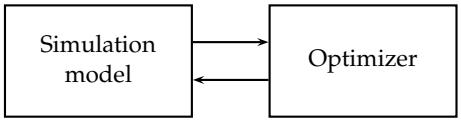

Figure 1.3: Simulation-based optimization.

In operations research, optimization problem (1.3) is called a simulation optimization problem. Simulation optimization approaches can be visualized as shown in Figure 1.3 (Gosavi, 2003): a simulation model is connected to an appropriate optimization algorithm. In reviews on simulation optimization, see, e.g., Fu (1994) and Carson and Maria (1997), a number of different approaches can be distinguished. The major techniques are: (i) gradient based search methods, (ii) stochastic methods, and (iii) Response Surface Methodology (RSM).

Gradient based methods estimate the response function gradient. This estimated gradient is used to compute a new search direction iteratively. For discrete-event simulation, gradients can be obtained using two different approaches: (i) single-path methods, e.g., perturbation analysis (Glasserman, 1991) and the likelihood ratio method (also called the score function method) (Rubinstein, 1986), or (ii) finite-difference schemes. The single-path methods obtain gradient information based on one single run of the discrete-event simulation model. However, single-path methods are not suited to determine gradient information for 'structural' parameters in the discrete-event model such as number of machines or number of operators, which are strict integer variables. The idea of, for instance, perturbation analysis is that it applies small perturbations on distributional parameters. Furthermore, single-path methods cannot be applied to black-box models, since it requires the incorporation of some additional features in the simulation model. Gradients for black-box simulation models, can only be computed through finite-differences. The computationally expensive finite-difference method is however also not able to compute gradients for 'structural' type of parameters, which can usually only be simulated at integer levels (e.g. number of machines, buffer sizes, and batch sizes). Moreover, due to the stochastic simulation responses, gradients via finite-differences may be inaccurate which slows down the convergence rate of the optimization algorithm that is used.

Stochastic optimization methods represent a second group of methods for simulation optimization. Examples are genetic algorithms and simulated annealing. These stochastic methods can be more generally applied, since these methods have no requirements on the design variables and can be used for black-box models. Major drawback is however that these methods require a large number of simulation evaluations, especially for an increasing number of design variables. Due to the random nature of these methods it is generally impossible to predict the required number of simulations beforehand.

Thirdly, RSM techniques are employed for simulation optimization. Basic RSM is a means to obtain an approximation of the objective function for the entire design space. RSM is explained in detail in textbooks such as of Myers and Montgomery (2002). For optimization purposes *sequential* RSM is used. A pure linear approxima-

tion is built in a subdomain of the design space based on a design of experiments of simulation evaluations. This local linear approximation is used to determine the steepest-descent direction. In this direction a line search method is carried out which delivers an improved design. This design is starting point for the next iteration in which a new local approximation is built that determines a new steepest-descent direction. When the optimal solution is approached, sequential RSM typically switches to quadratic approximations to catch the location of the optimal solution. Within the operations research field, optimization with a discrete-event simulation model is usually considered for unconstrained optimization problems. Recently, a constraint version of sequential RSM for simulation optimization has been proposed by Angün et al. (2003). Activity of constraints is accounted for in the determination of the search direction.

1.4 Sequential approximate optimization

Simulation-based optimization problems are encountered in various research fields, such as mechanics, thermodynamics, fluids, acoustics, and electrodynamics. In each of these research fields a computationally expensive simulation model is employed for analysis purpose. This implies that the optimization has to be carried out with a computationally expensive simulation model in the loop. The term simulation refers here to any analysis model that takes a considerable amount of computing time. Examples of simulation models other than discrete-event simulation are, e.g., finite element method analysis in a linear or non-linear setting, and numerical integration of a large set of ordinary differential equations or differential-algebraic equations.

In the field of structural optimization simulation-based optimization problems are studied with a computationally expensive Finite Element Method (FEM) calculation included in the optimization loop. The FEM calculation is in most cases however deterministic, contrary to the stochastic discrete-event simulation. In the journal issues of *Structural and Multidisciplinary Optimization* and *International Journal for Numerical Methods in Engineering* many different techniques can be found to solve structural optimization problems. Structural optimization problems are defined for, e.g., truss design problems of towers and bridges, and vehicle design problems, such as minimizing vibrations in cars and airplanes.

Sequential Approximate Optimization (SAO) is a well-known technique in the structural optimization field. SAO is similar to sequential RSM, with two major differences: (i) SAO builds approximation models for objective *and* constraint functions, and (ii) SAO does not apply a line search method in each iteration to obtain a better design. For each iteration, SAO builds new local approximations for objective and constraints in a search subregion of the design space. Within this subdomain, a mathematical programming solver is used to solve the *approximate* optimization problem with respect to the approximate objective function and the approximate constraint functions. This results in a new approximate optimal solution within the subdomain, without applying line search. If the newly obtained optimal solution of the approximate optimization problem has improved compared to the earlier obtained solutions, this new approximate optimal solution is starting point for the next

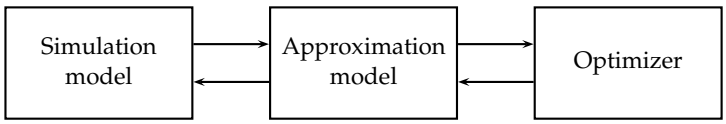

Figure 1.4: Sequential approximate optimization in structural optimization.

iteration. The search subregion is moved in this direction. This concept of SAO is depicted in Figure 1.4 and shows that the mathematical optimizer always applies to the explicit approximation model and never directly to the computationally expensive simulation model.

In structural optimization, two major SAO methods can be distinguished: (i) single-point methods and (ii) multiple-point methods. The single-point methods build approximation models based on function evaluations and gradient evaluations in one single point of the subdomain. In many FEM calculations, gradients can be calculated at only little extra computational cost. Then, the single-point method can be applied efficiently. Whenever gradients are not available or cannot be easily computed by finite differences, multi-point SAO can be applied.

Toropov et al. (1993) introduced a well-known multi-point SAO strategy within the structural optimization field. In each iteration, this method builds response surface models using a restricted number of simulation evaluations at carefully selected design points within each search subregion. A move limit strategy is used to define the sequence of search subregions. Relocation of the search subregion is often done in the direction of the last accepted approximate optimal design. Toropov et al. (1993) take the last cycle optimal design as corner point in the next search subregion. One may also move the search subregion such that the last accepted design is in the center of the subdomain of the next iteration. Several other rules for resizing and moving the search subregion are possible (see, e.g., Wujek and Renaud, 1998). Typically, these move limit strategies are heuristic in nature and may differ for the various application domains.

SAO methods are useful in many application domains. A challenge for each domain is the development and use of good quality approximation models. In structural optimization, linear and reciprocal approximation models are used to describe the underlying physics of stress and displacement functions. Reciprocal approximations were introduced by Schmit and Farshi (1974) to describe stress and displacement constraints more accurately. Fleury and Braibant (1986) developed the convex linearization approach (CONLIN) combining linear and reciprocal approximation functions. Another example is the use of reciprocal intervening variables in the method of moving asymptotes of Svanberg (1987). These approximation models match well with the monotonically decreasing or increasing stress and displacement constraints.

In the field of simulation optimization, flow time responses also have asymptotes (for utilization near one), but the underlying physics of the flow time is different compared to the physics of stresses and displacements. This means that for each application domain, different approximation models have to be developed. Unfortu-

nately, SAO strategies are usually offered as some (black-box) code that call external user routines. Changes of the approximation function or the SAO method cannot be easily realized.

1.5 ADOPT-project

The ADOPT-project is a joint effort of three research groups at Eindhoven University of Technology and Delft University of Technology. The project is financed by the Dutch technology foundation STW. The research project starts from SAO as method for simulation-based optimization that can be successfully applied to a wide range of engineering application domains. Aim of the project is to further develop existing SAO methodologies, with the focus on applications that deal with the following characteristics: uncertainties, discontinuities, and discrete design variables. In the context of mechanical engineering, these characteristics are common for simulation-based optimization.

The idea is to develop methods for SAO which should be included in one SAO software tool. The use of good quality approximation models in the SAO method is essential. Each application domain may require application specific approximation models. Besides the development of new approximations models, each application domain may also require the development of new SAO strategies. The SAO strategy defines, e.g., how the subdomains are moved through the design space or which (and how many) design points are evaluated that built the approximation model in each iteration.

Within the ADOPT-project three different research fields are involved: the Dynamics and Control group and the Systems Engineering group at the Eindhoven University of Technology and the Structural Optimization and Computational Mechanics group at the Delft University of Technology. Four PhD students are involved for each of the following application domains:

- (multi-body) dynamical systems,
- composite structures,
- Microelectromechanical Systems (MEMS), and
- manufacturing systems.

Each of these four application domains deals with optimization including a computationally expensive simulation model in the loop. The simulation models involved are: numerical integration of a set of differential equations (multi-body systems), FEM analysis (composites structures and MEMS), and discrete-event simulation (manufacturing systems).

Furthermore, each application domain deals with one or more of the characteristics mentioned before. Focus of the multi-body dynamical systems PhD research project is on uncertainties with known distributions, specifically related to reliability-based optimization, and optimization for robustness. More specifically, this research field deals with uncertain design variables with known distribution functions. The

MEMS research project considers bounded-but-unknown uncertainties, i.e. uncertainties for which upper and lower bounds are given, but the distribution functions are not known. Variation of length-scale parameters of 10% in a MEMS structure is very common. The focus of the composite structures research project is on the inclusion of FEM gradient information in the response surface model building, and the treatment of discontinuities. Typical design variables are, e.g., fiber angles. Some fiber angle combinations may lead to impossible solutions (incomplete function evaluations) and small deviations in fiber angles may lead to large jumps in response functions (discontinuities).

The research field of this thesis focuses on manufacturing systems. The use of discrete-event simulation models implies the following characteristics: design variables may have integer restrictions, simulation responses are stochastic, and some design points may lead to non-analyzable simulations. Typical integer design variables are, e.g., number of machines, number of operators, number of buffer sizes, and number of batch sizes. The simulation model can only be evaluated at the integer levels of these design variables. The underlying physics, however, is continuous. Therefore, the approximation models may still be continuous functions. The integer restriction influences, however, the implementation of: the move limit strategy, i.e. how the subregions are placed in the design space, the design of experiments, and the approximate optimization solver. Typical discrete-event simulation responses, such as flow time and throughput, are stochastic. This is caused by the stochastic nature of discrete-event simulation models itself. Certain design points may lead to non-valid simulations. This is the case whenever the required throughput exceeds the bottleneck capacity of the system. Such a system does not deliver a steady-state solution and cannot be analyzed. Such non-analyzable regions in the design space lead to discontinuities in response functions.

1.6 Research questions

Within the context of the ADOPT-project, this thesis concentrates on the following two main research questions:

(i) Can approximate optimization concepts be employed for simulation optimization of discrete-event manufacturing systems regarding flow time performance, and which methods and tools are required for this purpose?

(ii) Can operational time variability be quantified in a single performance measure using shop floor data without identifying the individual contributing disturbances, and can such a method also be employed to obtain simple but accurate simulation models of manufacturing flow lines?

The first research question studies the use of RSM techniques in a multi-point sequential approximate optimization approach for the optimization of flow time performance in discrete-event manufacturing systems, taking into account the properties as mentioned before. In order to reduce computational time of simulation-based optimization of manufacturing systems, it is suggested to build computationally

fast approximation models based on a small number of carefully selected simulation runs of the simulation model. The SAO concepts from structural optimization will be used for this purpose. The approximation models can be used iteratively by the optimizer to search for an approximate optimal design. The main challenge is to develop good quality approximations that are suited to model the typical flow time responses in manufacturing systems.

Recently, Abspoel et al. (2001) developed a multi-point SAO technique for simulation-based optimization of manufacturing systems. Abspoel et al. (2001) approximated the objective and constraint functions using pure linear approximation models. Only integer design variables, such as number of machines and batch sizes, were considered. The pure linear approximations are not able to represent the curvature in the flow time response functions. Abspoel et al. (2001) observed that, as a result, optimization runs often ended in neighbors of the known discrete optimal solution.

Gijsbers (2002) extended the method of Abspoel et al. (2001) to more general linear regression models. The regression model he proposed is able to approximate more accurately the flow time responses of discrete-event manufacturing systems. The flow time was approximated by a linear regression model in which the number of machines per workstation is treated as a design variable. This regression model included one asymptote for each workstation which is typical for the flow time response. This idea significantly improved the convergence behavior of the test problems presented in Abspoel et al. (2001). However, design variables other than the integer number of machines and machine types other than single-lot machines were not considered.

The second research question is motivated as follows. Simulation-based optimization of manufacturing systems requires good quality and valid discrete-event simulation models. Such simulation models have to reflect the capacity and variability effects in order to predict flow times accurately. To build accurate models, it is required that the model includes all important shop floor realities. Examples of shop floor realities are processing, setups, failure of machines, operator availability, and dispatching rules. For the process times as well as the processing disturbances the distribution function has to be taken into account. These distribution functions may be estimated from measured data of the operating factory. In practice, however, it is difficult to measure all individual process disturbances. Some process disturbances cannot be measured at all and others may even be unknown. This is also the reason why the measure for variability as mentioned in Section 1.2 is still missing.

In the book of Hopp and Spearman (2001) variability is estimated from the individual contribution of each of the process disturbances. This requires knowledge of all (or the most important) contributing disturbances, which is often not available. The idea followed in this thesis is to quantify variability of an operating factory, without measuring each individual processing disturbance. Hopp and Spearman use the concept of effective process time to include all variability effects into a single measure. Basically, the effective process time includes processing time and all process disturbances. However, a method that can be used to measure effective process time in an operating factory without identifying the individual disturbances is not available. Such a measure would be highly valuable to understand factory physics and

may also provide new opportunities to build simple but accurate simulation models of manufacturing flow lines.

1.7 Objectives and approach

Based on the two research questions posed in the previous section, this thesis focuses on the following three research objectives:

- development of a framework for sequential approximate optimization that can be used in various application domains, including simulation-based optimization of manufacturing systems,
- development of variability measures based on the concept of effective process time that can be used in flow time performance analysis, and
- development of good quality flow time approximation models based on queueing physics for use in sequential approximate optimization of manufacturing systems modeled by discrete-event simulation.

The first objective is motivated by one of the main goals of the ADOPT-project. The aim is to develop an open environment for the development and use of SAO approaches in various application domains. This thesis proposes an SAO framework that gives access to the different computational steps of the SAO strategy. This results in a flexible software tool in which computational steps for SAO can easily be changed, added, and rearranged. The developed framework has a different purpose than existing software packages. Most available packages have a more general multidisciplinary optimization scope of application and are not specifically designed for the development of sequential approximate optimization strategies.

Regarding the second objective, flow time performance relies mainly on the two important factory physics entities: utilization and variability. To understand the factory physics of a manufacturing system, a performance measure that quantifies both utilization and variability is needed. Unfortunately, available performance measures that can actually be measured in an operating factory, like OEE, are only related to utilization. The concept of effective process time introduced by Hopp and Spearman (2001) is promising to measure both utilization and variability in an operating factory. However, definitions to actually measure effective process times based on operating factory data are missing. But the basic idea of the effective process time to include time losses does give a starting point. The idea is to compute effective process time realizations of individual parts based on a list of events with arrival and departure times of parts at each workstation. We propose a new method to compute effective process times from such a data set. This approach enables one to estimate the mean and variance of the effective process time of a workstation. This gives the desired quantification of mean and variance of effective process time.

The third objective focuses on the development of good quality flow time approximation models. In order to build high quality approximation models that describe flow time performance, the factory physics of manufacturing systems has to

be analyzed. The idea is to take as starting point the observation that utilization and variability are the main drivers for flow time performance. The regression approximation model of Gijsbers (2002) already included the utilization effects when the number of machines is treated as a design variable, but did not include variability. This regression model is further improved and generalized to include the effect design variables may have on both utilization and variability. This allows to include various types of design variables. For instance, variables that affect disturbances may be treated as a design variable. The new flow time regression model will be used to define the approximate optimization problems in which the design variables can be continuous or integer. Since capacity and variability are the main drivers for flow time, the use of effective process time in the flow time approximations is essential here. The good quality approximation model for flow time performance is now based on utilization and variability measures.

1.8 Outline

This thesis consists of four research chapters, Chapter 2 till Chapter 5. Chapters 2 and 3 have been published as journal articles in *Structural and Multidisciplinary Optimization* and *IEEE Transactions on Semiconductor Manufacturing*, respectively. Chapter 4 has been submitted for publication in *IEEE Transactions on Semiconductor Manufacturing*. This paper was written together with P. P. van Bakel for which he is gratefully acknowledged. Chapter 5 has yet to be submitted. The original text of each journal article is presented here. Each chapter is self-contained. Finally, the main conclusions of this work and recommendations for further research are given in Chapter 6.

Chapter 2 describes the newly developed software framework for sequential approximate optimization. This framework has been developed within the ADOPT-project. Chapter 3 introduces the concept of effective process time. This chapter proposes algorithms that can be used to measure Effective Process Time (EPT) realizations for single-machine workstations and multiple-machine workstations. Chapter 4 continues on the EPT concept and proposes EPT algorithms for batching equipment. This type of equipment processes a collection of products at once. Batching equipment is common in semiconductor industry. Chapter 5 merges the ideas of SAO simulation-based optimization with the EPT concept. The main contribution is the development of a high quality flow time approximation model. This flow time approximation uses the EPT algorithms developed in Chapters 3 and 4. These flow time approximations are embedded in an SAO strategy that has been developed using the SAO framework described in Chapter 2.

Chapter 2

Framework for Sequential Approximate Optimization

An object-oriented framework for Sequential Approximate Optimization (SAO) is proposed. The framework aims to provide an open environment for the specification and implementation of SAO strategies. The framework is based on the Python programming language and contains a toolbox of Python classes, methods, and interfaces to external software. The framework distinguishes modules related to the optimization problem, the SAO sequence, and the numerical routines used in the SAO approach. The problem-related modules specify the optimization problem, including the simulation model for the evaluation of the objective function and constraints. The sequence-related modules specify the sequence of SAO steps. The routine-related modules represent numerical routines used in the SAO steps as 'black-box' functions with predefined input and output, e.g. from external software libraries. The framework enables the user to (re-) specify or extend the SAO dependent modules, which is generally not possible in most available SAO implementations. This is highly advantageous since many SAO approaches are application-domain specific due to the type of approximation functions used. A ten-bar truss design problem with fixed loads as well as uncertain loads is used as an illustration and demonstrates the flexibility of the framework.

2.1 Introduction

Approximations often play a key role in the efficient solution of multidisciplinary design optimization problems. Several different approximation approaches have been proposed (see for an overview, e.g., Barthelemy and Haftka, 1993) and have been

Reproduced from: Jacobs, J. H., Etman, L. F. P., Van Keulen, F., and Rooda, J. E. (2004). Framework for sequential approximate optimization. *Structural and Multidisciplinary Optimization*, 27(5):384–400

successfully used in various optimal design applications. Many approaches follow a Sequential Approximate Optimization (SAO) approach to build and solve a series of approximate optimization subproblems. An introduction to SAO techniques can be found in Haftka and Gürdal (1991).

SAO methods are used especially when a computationally expensive and/or noisy simulation model is part of the optimization loop. In order to assure a reasonably fast optimization, the number of calls to the simulation model, i.e. the number of evaluations of objective function $f(\mathbf{x})$ and constraint functions $\mathbf{g}(\mathbf{x})$, has to stay small. Therefore, SAO methods build computationally inexpensive (explicit) approximations for $f(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ using a restricted number of simulation evaluations in a subregion of the total design space. The resulting approximate optimization subproblem can be easily solved within the search subregion using any suitable mathematical programming algorithm. A move limit strategy (or trust region strategy) is used to successively define the sequence of search subregions in which approximate optimization subproblems are built and solved. Generally, each approximate optimal design is evaluated using the simulation model. The SAO process is stopped whenever certain stopping criteria are met, e.g., when no further improvement is observed or the maximum number of iterations is reached.

SAO techniques do not necessarily require gradient information. Many simulation models do not provide gradient information. If gradient information is available, gradients can be incorporated in the approximation building. In this way, computational time can be reduced or more design variables can be included.

Two basic types of SAO techniques are commonly employed: (i) single-point approximations, and (ii) multi-point approximations. A single-point approximation is based on simulation results in just a single design point of the design space. Typically, function value and gradient information is used for this purpose. Various single-point approximation methods have been proposed. The simplest one is to build linear approximations of objective function and constraints, which yields a sequential linear programming (SLP) approach (Pedersen, 1981). For structural optimization, reciprocal approximations were introduced by Schmit and Farshi (1974) to describe stress and displacement constraints more accurately. Fleury and Braibant (1986) developed the convex linearization approach (CONLIN) combining linear and reciprocal approximation functions. The Method of Moving Asymptotes (MMA) presented by Svanberg (1987, 1995, 1999) generalizes the convex linearization approach by allowing the (convex) curvature in the approximation to change. An overview of move limit strategies for single-point approximations can be found in Wujek and Renaud (1998). Alexandrov et al. (1998) give a mathematical foundation of the use of trust regions in the single-point SAO context.

Multi-point approximations require simulation data in more than one design point to build the approximation. Several different approaches can be found. Vanderplaats (1979) introduced a sequential approximate optimization strategy, which in each cycle adds one simulation of a design point to the total set of evaluated designs. The complete set of evaluated designs is used to create an approximation using the best possible Taylor series expansion, until a full quadratic approximation is possible. In each cycle, the CONMIN optimizer was used by Vanderplaats (1976)

to solve the approximate optimization problem within the search subregion. The approximate optimum is simulated and then added to the set of evaluated designs. The updated set is used to create a new approximation. The search subregion is moved such that the last added design becomes the new center, while keeping the subregion size constant. Brekelmans et al. (2004) also add one simulation evaluation at a time in their sequential approach, but they chose between the objective improving point (i.e. the last obtained approximate optimal design) and a geometry improving point. The latter point aims to improve the approximation.

Toropov et al. (1993, 1996) proposed a multi-point approximation method that adds more than one design point in every new search subregion. They used linear or posynomial function relations to approximate objective and constraint functions. Rectangular search subregions are used. The size of the search subregion is reduced if (i) the approximations are inaccurate or (ii) the newest approximate optimum is found within the search subregion (no active move limit). The search subregion is moved in the direction of the newest approximate optimum. Similar multi-point approaches have been presented in Etman et al. (1996), Stander et al. (2003), and Craig and Stander (2003); Craig et al. (2003).

Haftka et al. (1987) followed by Fadel et al. (1990), Wang and Grandhi (1995), and others, aim to improve the local approximation by using function values and gradient data in two points of the design space. Fadel and Cimtalay (1993) used the exponents from the two-point exponential approximations as a measure for the curvature which determines the move limit. Another example is the method of Snyman and Hay (2002), who build spherically quadratic approximations on the basis of function values and gradient information in the current design point and function values in the previous point. Instead of a rectangular shaped search subregion, they use spherical search subregions. Bruyneel et al. (2002) recently developed a two-point generalization of MMA.

Solution of the approximate optimization subproblem is usually carried out by means of a suitable optimization algorithm. Various algorithms are available varying from gradient-based mathematical programming solvers to heuristic optimization solvers. Quite a number of software implementations of these algorithms are available (commercially or public domain). See, e.g., OTC (2003) for an overview. Optimization functionality may also be provided through an optimization language environment such as AMPL (Fourer et al., 1993), AIMMS (Bisschop and Roelofs, 2002), or GAMS (Brooke et al., 1998), or through more general computing languages such as Matlab (Mathworks, 2002). Typically, the mathematical programming implementations aim to be good quality (accurate) and robust general purpose minimizers without a designation to a specific application domain. These solvers are generally not designed to minimize the number of function evaluations in the first place.

Implementations of SAO strategies are usually offered in a similar fashion as standard mathematical programming algorithms. That is, the SAO strategy is implemented in some (black-box) code which calls external user routines. The SAO strategy, as opposed to the above mentioned mathematical minimizers, assumes that objective and constraint evaluations are expensive and that in general they cannot be evaluated separately. From a user-perspective, these SAO implementations can

be seen as a special class of optimization algorithms with specific simulation 'reduction' features. The price to pay is that they may not converge equally well for various types of problems. MMA has proved to work well in structural type of applications, since the reciprocal intervening variables in the approximations matches well with the monotonically decreasing or increasing stress and displacement constraints (Papalambros and Wilde, 2000). In other application domains different approximation functions (or intervening variables) may be needed to take advantage of the SAO approach compared to general purpose line search or trust region based mathematical programming algorithms.

We have developed a framework for the development and use of SAO approaches in simulation-based optimization applications. The framework enables the user to (re-) specify or adapt an SAO strategy easily, which is generally not possible in most available SAO implementations. The developed framework has a different purpose than existing software packages such as iSIGHT (Koch et al., 2002), ModelCenter (Phoenix, 2004), DAKOTA (Eldred et al., 2002), and others. These packages facilitate easy integration of various simulation analysis software and offer a range of optimization algorithms and design of experiments routines. They have a more general MDO scope of application and are not specifically designed for the development of sequential approximate optimization strategies. DAKOTA, for example, offers the SAO strategy described in Giunta and Eldred (2000) as one of the various optimization algorithm options.

This paper describes the framework and focuses on the layout and data structure of the framework, as well as the implementation in the programming language Python (Van Rossum and Drake, 2003a). Section 2.2 gives the optimization problem formulation. Section 2.3 describes the basic elements of the sequential approximate optimization method. Section 2.4 elaborates these elements by providing a layout of the framework. These elements are explained in an object-oriented way by means of class diagrams. Section 2.5 presents the accompanying data structure. Section 2.6 describes the implementation of the framework in Python and explains the advantage of using Python. In Section 2.7, the framework is illustrated by means of a ten-bar truss design example for two design cases: (i) fixed deterministic loading, and (ii) bounded uncertain loading. Section 2.8 concludes the paper.

2.2 Optimization problem formulation

The following non-linear inequality-constrained optimization problem $\mathcal{P}$ is considered:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}), \\ \text{subject to} & \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \text{ and} \\ & \mathbf{x}_\ell \leq \mathbf{x} \leq \mathbf{x}_\text{u} \end{array} \tag{2.1}$$

with $\mathbf{x}$ the set of design variables, $f(\mathbf{x})$ the objective function, and $\mathbf{g}(\mathbf{x})$ the set of constraint functions. Design variable x_i belongs to the set of design variables $\mathbf{x}$, with $i = 1, \ldots, n$ and n the number of design variables. Three types of design variables are considered: continuous, integer, and discrete. In (2.1) the set of design variables $\mathbf{x}$ can be a mix of these types. The parameter sets $\mathbf{x}_\ell$ and $\mathbf{x}_\text{u}$ represent the lower and

upper bounds of the design space for each of the design variables in $\mathbf{x}$. Constraint g_j belongs to the set of constraint functions $\mathbf{g}$, with $j = 1, \ldots, q$ and q the number of constraint functions.

Simulation models are used to compute responses $\mathbf{r}$ for certain design variable values $\mathbf{x}$ and parameter values $\mathbf{p}$. Examples of such models are (non-linear) FEM models, discrete-event simulation models, or multi-body dynamical models. The responses of the simulation model are used to evaluate objective function $f(\mathbf{x})$ and constraint functions $\mathbf{g}(\mathbf{x})$. Part of the objective and constraint functions may also be explicit (or computationally inexpensive) functions, i.e. not determined by the simulation models. Objective function $f(\mathbf{x})$ can either be a simulation-based objective function $f_\mathrm{s}(\mathbf{r}(\mathbf{x}, \mathbf{p}))$, or an explicit objective function $f_\mathrm{e}(\mathbf{x})$. The same holds for each of the constraints: constraint g_j can be either simulation-based: $g_{\mathrm{s},j}(\mathbf{r}(\mathbf{x}, \mathbf{p}))$, or explicit: $g_{\mathrm{e},j}(\mathbf{x})$. The explicit (or computationally inexpensive) functions do not need to be approximated in each iteration of the SAO process.

The response of a simulation model can be deterministic or stochastic. Stochastic responses can be caused by stochastic design variables $\mathbf{X}$, stochastic parameters $\mathbf{P}$, stochasticity in the simulation model itself denoted by ω, or a combination. FEM-calculations with deterministic input $\mathbf{x}$ and $\mathbf{p}$ generally give deterministic output $\mathbf{r}(\mathbf{x}, \mathbf{p})$. In analysis for reliability or robustness, stochastic design variables $\mathbf{X}$ or stochastic parameters $\mathbf{P}$ may occur, which results in stochastic simulation responses $\mathbf{R}(\mathbf{X}, \mathbf{p})$ and $\mathbf{R}(\mathbf{x}, \mathbf{P})$, respectively. The estimated distribution of these responses is possibly needed to determine the measure of reliability or robustness. In discrete-event simulation, the stochastic responses $\mathbf{R}(\mathbf{x}, \mathbf{p}, \omega)$ are due to internal stochastic behavior in the simulation itself represented by variable ω; in this field objective function and constraints are usually formulated as expected values using deterministic design variables $\mathbf{x}$ and deterministic parameters $\mathbf{p}$ (see, e.g., Law and Kelton, 2000). The expected values and variances are estimated from a finite number of simulations for each design point. On the basis of these estimates and possibly corresponding confidence intervals, deterministic objective and constraint values that match (2.1) must be obtained.

2.3 Sequential approximate optimization method

A typical SAO method decomposes the optimization process into a sequence of cycles. In each cycle, an approximate optimization subproblem is defined within a search subregion, and solved. At the end of a cycle, certain stopping criteria determine whether the sequence is stopped or not. If the sequence is not stopped, the search subregion is moved and/or resized, and a new cycle is started.

2.3.1 Optimization sequence

A cycle in the SAO process is denoted by $c^{(k)}$ with k the running number of the cycle. Within one cycle, a number of steps can be distinguished. A common basic sequence is as follows. Cycle $c^{(k)}$ starts with initial design point $\mathbf{x}_0^{(k)}$ in search subregion $r^{(k)}$.

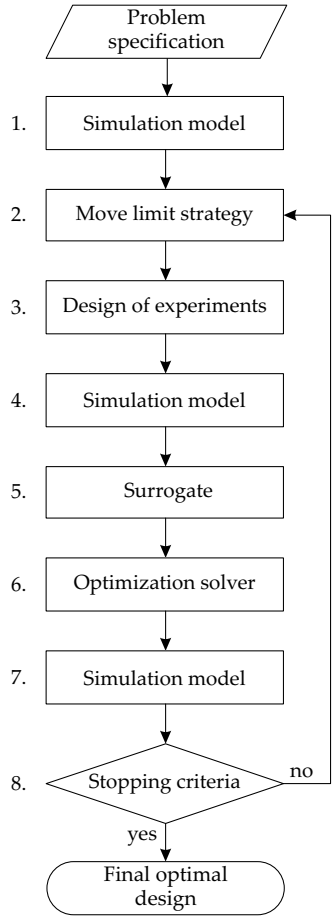

Figure 2.1: Example of a commonly used SAO sequence.

For this search subregion, an approximate optimization problem is formulated. Solving this approximate subproblem within search subregion $r^{(k)}$ gives an approximate optimum $\mathbf{x}_*^{(k)}$. Based on the optimization history during the cycles, the SAO method determines the new search subregion $r^{(k+1)}$ for the next cycle $c^{(k+1)}$ until the SAO process is stopped.

In the above mentioned SAO sequence, typically the following steps can be distinguished during cycle k:

1. compute the true responses in initial design point $\mathbf{x}_0^{(k)}$,

2. construct search subregion $r^{(k)}$ based on a move limit strategy,

3. plan a design of experiments in search subregion $r^{(k)}$,

4. compute the response values in the plan points using the extensive simulation model,
5. construct an approximation model based on the computed responses,
6. solve the approximate optimization problem within search subregion $r^{(k)}$,
7. compute the true responses in the approximate optimum $\mathbf{x}_*^{(k)}$,
8. stop if stopping criteria are met, otherwise start cycle $k+1$ and return to Step 2.

A graphical representation of these steps can be found in Figure 2.1. The rectangular boxes represent the modules that perform the independent computational steps. The arrows between the different modules represent the algorithm flow with corresponding data exchange. Step 2 refers to the move limit strategy. Steps 3–7 refer to the approximate optimization problem. Step 8 refers to the stopping criteria. The basic sequence shown in this figure is often used and can be found for example in Giunta and Eldred (2000), but other types of SAO sequences are possible as well (see, e.g., Pérez et al., 2002).

The approximate optimization problem, the stopping criteria, and the move limit strategy are further explained below.

2.3.2 Approximate optimization problem

In each cycle $c^{(k)}$ of the SAO method, an optimization subproblem is created. This subproblem is denoted by $\mathcal{P}^{(k)}$ which is defined within the bounds of search subregion $r^{(k)}$. Search subregion $r^{(k)}$ can be a region of any shape, but typically spherical or rectangular regions are used. For convenience of our discussion, rectangular regions are assumed. Such a rectangular region is defined by lower bounds $\mathbf{x}_\ell^{(k)}$ and upper bounds $\mathbf{x}_\mathrm{u}^{(k)}$. Approximate optimization subproblem $\mathcal{P}^{(k)}$ can then be stated as:

$$\begin{array}{ll} \text{minimize} & f_\mathrm{a}^{(k)}(\mathbf{x}), \text{ or} \\ & f_\mathrm{e}(\mathbf{x}), \\ \text{subject to} & \mathbf{g}_\mathrm{a}^{(k)}(\mathbf{x}) \leq \mathbf{0}, \text{ and/or} \\ & \mathbf{g}_\mathrm{e}(\mathbf{x}) \leq \mathbf{0}, \text{ and} \\ & \mathbf{x}_\ell \leq \mathbf{x}_\ell^{(k)} \leq \mathbf{x} \leq \mathbf{x}_\mathrm{u}^{(k)} \leq \mathbf{x}_\mathrm{u} \end{array} \tag{2.2}$$

Herein, simulation-based objective function $f_\mathrm{s}(\mathbf{x})$ and constraint functions $\mathbf{g}_\mathrm{s}(\mathbf{x})$ have been replaced by approximations $f_\mathrm{a}(\mathbf{x})$ and $\mathbf{g}_\mathrm{a}(\mathbf{x})$, respectively. These approximate relations are referred to as surrogate functions. The complete set of surrogate functions is called a surrogate denoted by a. For example, if we have one simulation-based objective function $f_\mathrm{s}(\mathbf{x})$ and one simulation-based constraint function $g_\mathrm{s}(\mathbf{x})$, and we approximate these functions by reciprocal linear regression relations, then for two design variables surrogate a becomes:

$$f_\mathrm{a}(\mathbf{x}) = \alpha_0 + \alpha_1 \cdot 1/x_1 + \alpha_2 \cdot 1/x_2 \tag{2.3}$$

$$g_\mathrm{a}(\mathbf{x}) = \beta_0 + \beta_1 \cdot 1/x_1 + \beta_2 \cdot 1/x_2 \tag{2.4}$$

In each cycle the unknown surrogate parameters (α_i and β_i in our example) have to be determined on the basis of simulation evaluations in the plan points of the present cycle and possibly previous cycles. The type of surrogate function (e.g. linear regression, Kriging, or radial basis), the amount of data points used, and the number of parameters to be estimated determine the parameter estimation approach. The aim is to obtain an approximation model $a^{(k)}$ that approximates the simulation response behavior within the search subregion as well as possible.

Approximation model $a^{(k)}$ together with explicit functions $f_{\mathrm{e}}(\mathbf{x})$ and $\mathbf{g}_{\mathrm{e}}(\mathbf{x})$ build approximate optimization problem $\mathcal{P}_{\mathrm{a}}^{(k)}$. A mathematical programming solver is applied to this approximate problem in search subregion $r^{(k)}$. The type of solver used strongly depends on the type of approximation functions and the type of design variables. E.g. solving a non-linear approximate optimization problem with continuous and integer design variables requires a mixed-integer NLP solver.

2.3.3 Stopping criteria

The subproblem optimization solver returns an approximate optimal design, denoted by $\mathbf{x}_*^{(k)}$. Simulation of approximate optimum $\mathbf{x}_*^{(k)}$ results in the objective value $f_{\mathrm{s}*}^{(k)}$ and constraint values $\mathbf{g}_{\mathrm{s}*}^{(k)}$ at the simulation level. Then, at the end of this cycle, it has to be decided if the sequence proceeds or stops. There may be various different reasons to stop, for instance: when during the last few cycles no significant reduction of the objective function is observed and the constraint violations satisfy the accuracy requirements. The stopping criteria are tied to the type of SAO approach, e.g. multi-point versus single-point. After the sequence has been completed, finally, the best design has to be selected. The final optimal design is often, but not necessarily, the last cycle optimal design found. This again relates to objective function value and constraint violations. In the case of stochastic functions confidence intervals may play a role as well.

2.3.4 Move limit strategy

If at the end of a cycle the stopping criteria have not been met, a new cycle is initiated. At the start of this new cycle $c^{(k)}$, the location and size of the new search subregion has to be determined. The move limit strategy takes care of this subregion placement. Assuming a rectangular subregion, for each of the design variables search subregion lower bound $x_{\ell,i}^{(k)}$ and search subregion upper bound $x_{\mathrm{u},i}^{(k)}$ have to be defined. The size of the search subregion then becomes $\Delta_i^{(k)} = x_{\mathrm{u},i}^{(k)} - x_{\ell,i}^{(k)}$. The move limit strategy should be able to resize and move the search subregion. The move limit strategy determines new values for parameters $\Delta_i^{(k+1)}$, at the end of each cycle $c^{(k)}$.

Relocation of the search subregion is often done in the direction of the last cycle optimal design. One can, for example, place the search subregion around this cycle optimal design, taking the cycle optimal design as the center point of the new search subregion. This is usually done in single-point-path SAO approaches. On the other

hand, several multi-point strategies also take the last cycle optimal design as corner point in the next search subregion (see, e.g., Toropov et al., 1993; Thomas et al., 1992).

Several other rules for resizing and moving the search subregion are possible (see, e.g., Wujek and Renaud, 1998). The move limit strategy comprises the complete set of rules regarding resizing and moving. Typically, these strategies are heuristic in nature and may differ for the various approximation methods.

2.4 Framework

The aim of the framework is to provide an open environment to specify and solve sequential approximate optimization problems. We first give a global overview of the layout of the developed framework. The layout shows how the framework can be divided into several modules with specific functionality. Next we discuss the class structure of the layout. The class structure is formalized in Unified Modeling Language (UML). UML diagrams show the relationship between the different modules. UML is explained in detail by Rumbaugh et al. (1999).

2.4.1 Layout

The layout of the proposed framework is designed such that it allows one to specify the optimization problem, the sequential approximate optimization sequence, as well as individual optimization steps. The framework consists of three basic layers: the problem layer, the sequence layer, and the routine layer. This is graphically represented in Figure 2.2. The three layers correspond with the optimization problem (problem layer), the SAO sequence (sequence layer), and the SAO steps (routine layer), respectively. The open environment is created by explicitly separating the SAO sequence from the individual SAO steps. Running the SAO sequence is represented by data, gathered during the cycles of the approximate optimization, whereas the SAO steps are 'simple' input-output functions (numerical routines) that carry out a computational task. These numerical routines are often Fortran, C, or Matlab implementations, either from an existing library or user-developed.

The problem layer of the framework holds the specification of the optimization problem and the simulation models. The specification of the optimization problem consists of (i) the formulation of the optimization problem and (ii) the specification how the objective and constraint functions have to be calculated. The formulation of the optimization problem is mathematically represented by (2.1). Objective function and constraints are computed through explicit relations, simulation responses, or a combination of those. Simulation responses $\mathbf{r}(\mathbf{x})$ are defined by the simulation model.

The second layer in the framework, the SAO sequence layer, has three functions: (i) it defines and controls the sequence of steps, (ii) it connects the problem layer and the routine layer, and (iii) it takes care of storing, transferring, and rearranging data. The sequence layer connects all the different modules by passing the data between the different numerical routines and the optimization problem. The SAO sequence defines the sequence of steps and specifies which routines are used in each

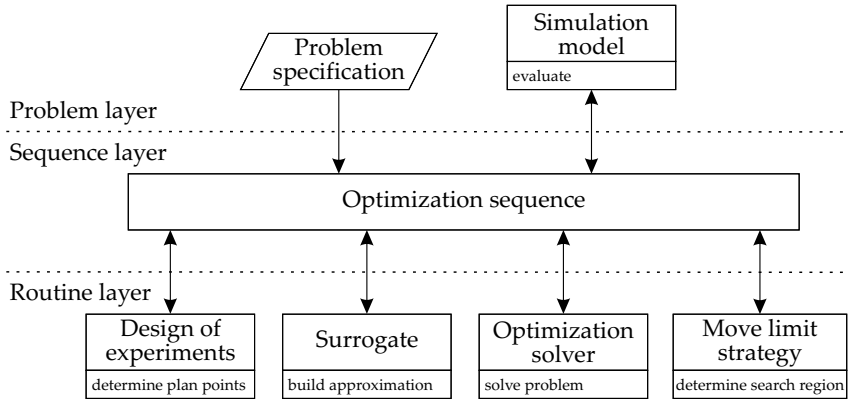

Figure 2.2: Framework layout consisting of three layers. Each layer has a specific task: (i) describe the problem formulation, (ii) describe the sequential approximate optimization sequence, and (iii) describe the individual computational routines.

of the steps, e.g. which design of experiments is used, which surrogate functions, which optimizer, etcetera. The framework enables one to specify different sequences using selected routines that perform the computational tasks. The sequence layer stores all relevant data generated during the optimization process. This data may be needed throughout the optimization process and is available for post-processing after the optimization process has ended. The SAO sequence itself does not perform computational tasks, except for simple calculations related to, e.g., stopping criteria. However, it may happen that more extensive calculations are needed to determine the next step in a particular sequence. In that case, a new module that performs these specific calculations may be added to the routine layer of the framework.

The third layer, the routine layer, includes all the routines which are used in the SAO steps specified by the optimization sequence. The routines are subdivided in separate modules. For each basic SAO step in the SAO sequence one module is available. The routines are treated as black-box input-output relations and do not contain any data of the optimization process itself. In Figure 2.2 each step is represented by a rectangular box with two parts. The upper part of the box is the interface between the routines and the sequence layer. The lower part of the box represents the routine that performs the specific task. When a number of different routines are available in a single module, the framework enables one to select one of these routines for the SAO approach.

For each of the modules, a collection of several different numerical routines is available. For example, a number of different routines for the design of experiments, such as full-factorial, Latin-hypercube, D-optimal, and random design of experiments is available. The user can add extra functionality by adding numerical routines to the framework, either in one of the existing modules or in newly developed modules. Adding a numerical routine to an existing module requires this routine to match with the input-output requirement as specified by the interface of the module. New modules should match with the input and output data structure as defined by

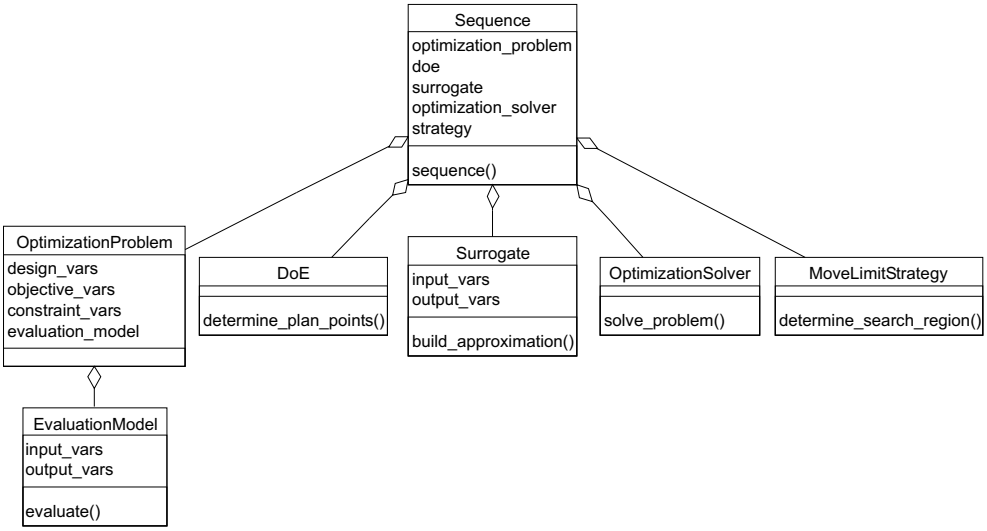

Figure 2.3: UML class diagram of the framework. Class Sequence aggregates all other classes.

the corresponding step in the SAO sequence.

2.4.2 Classes

The framework has been developed using an object-oriented class structure to create an extensible and well-structured environment, i.e. an open environment into which new functionality can easily be added. Each module of the framework has an equivalent class in the class structure. At least one instance of each of the classes of the framework is needed for an SAO run.

The UML diagram of the framework is shown in Figure 2.3. Each class in the diagram is represented by a box consisting of three compartments specifying the name, attributes, and methods of the class, respectively. Attributes are the object-oriented equivalent of data. Methods are the object-oriented equivalent of functions. The methods of the classes correspond to the functional relationships as shown in Figure 2.2.

The optimization sequence is represented by its equivalent class Sequence. Since the optimization sequence *uses* the modules, class Sequence is the main class of the framework. Five classes which all correspond to modules of the framework are aggregated[1]: OptimizationProblem, DoE, Surrogate, OptimizationSolver, and MoveLimitStrategy. The aggregated classes are also shown as attributes.

Class OptimizationProblem holds all the data required to specify the optimization problem. Besides the design, objective, and constraint variables, class OptimizationProblem also contains all the relations to evaluate the objective and constraint func-

[1]Aggregation is a means to express the "has a" relationship. A diamond-shaped connector is used in the UML diagram. An instantiation of the parent class contains an instantiation of the child class.

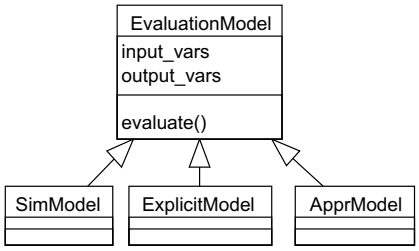

Figure 2.4: UML inheritance diagram of EvaluationModel.

tions. These relations can be represented by a computationally-expensive simulation model or explicit relations. All these type of relations are combined into class EvaluationModel.

The framework is able to specify different types of simulation models, design of experiments, surrogates, solvers, and move limit strategies. Below, the UML representations of the respective modules is explained. The inheritance[2] relationship is used to express that different type of modules can be selected within the framework.

Evaluation model

Figure 2.4 shows the inheritance diagram of class EvaluationModel. It shows three types of evaluation models: simulation models (SimModel), explicit models (ExplicitModel), and approximation models (ApprModel). All three types of evaluation models are able to determine one or a set of responses for a given design point. Attribute output_vars represents which responses the model is able to compute and attribute input_vars represents the design variables that are needed for this purpose. A simulation model, represented by class SimModel, is aggregated by class OptimizationProblem for determining the computational expensive responses $\mathbf{r}(\mathbf{x})$ in the 'true' optimization problem $\mathcal{P}$ of (2.1). The explicit functions $f_e(\mathbf{x})$ and $\mathbf{g}_e(\mathbf{x})$ defined in $\mathcal{P}$ are represented by class ExplicitModel, which are also aggregated by class OptimizationProblem. To solve the approximate optimization problem $\mathcal{P}_a^{(k)}$, class ExplicitModel is used together with the approximation functions represented by class ApprModel. In each cycle of the optimization process, new instances of class ApprModel are created by the surrogate functions. For this reason, class ApprModel is in fact a data entity instead of a framework entity and will be discussed in more detail in the next section.

The evaluation models are used to evaluate a design point in optimization problem $\mathcal{P}$ (using classes SimModel and ExplicitModel), as well as in approximate optimization problem $\mathcal{P}_a^{(k)}$ (using classes ApprModel and ExplicitModel). The result is an evaluation of the design point itself. This is represented by the evaluate() method, inherited by all three child classes. This means that the method is overloaded such

[2]Inheritance is a means to express the "is a" relationship. Arrow-shaped connectors are used in the UML diagram. The child class is a new version of the parent class.

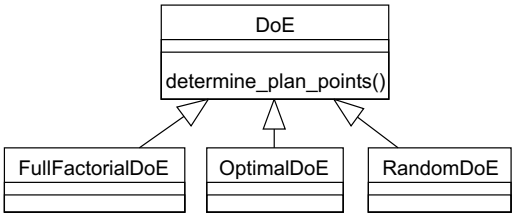

Figure 2.5: UML inheritance diagram of DoE.

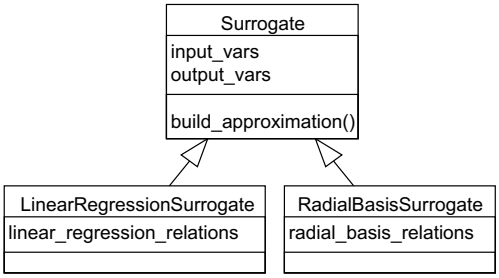

Figure 2.6: UML inheritance diagram of Surrogate.

that the functionality of each of the methods may be specified differently for each of the three model types.

DoE

Figure 2.5 shows an example of three inherited DoE classes: a full factorial design of experiments, a D-optimal design of experiments, and a random design of experiments. All three DoE classes inherit the method to determine the plan points in a specific search subregion. For each of the inherited DoE classes, the input-output relation of the method is the same, but the functionality differs.

Surrogate

In approximate optimization many types of surrogates are used. All these types of surrogates are inherited classes of class Surrogate. An example of a UML inheritance diagram of Surrogate is shown in Figure 2.6 which shows an example of two inherited classes: a surrogate based on regression functions and a surrogate based on radial basis functions. Class Surrogate has a method called build_approximation() to build approximation models based on evaluations of the simulation model. The attributes input_vars and output_vars are needed to link the evaluated plan points and responses of the simulation model to the approximation models. Furthermore, these attributes are needed during the approximate optimization step to determine which approximation model is needed to calculate a certain response and which design variables have to be provided.

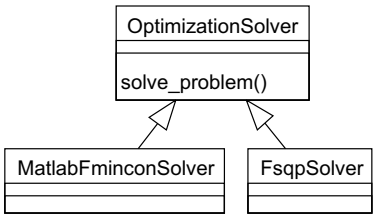

Figure 2.7: UML inheritance diagram of OptimizationSolver.

Optimization Solver

The framework enables the use of different types of optimization solvers. All optimization solvers inherit from the class OptimizationSolver. The solver has one method: solve_problem(). This method is used to find the optimal solution for a given approximate optimization problem. In Figure 2.7 an example of two inherited optimization solvers is shown which both have been implemented in the current framework. The MatlabFminconSolver class is an SQP solver for the minimization of nonlinear constrained optimization problems with continuous design variables in Matlab (Mathworks, 2002). The FsqpSolver class is an external SQP solver implemented in C++ (also available in Fortran) developed by Lawrence et al. (1997). The solve_problem() method provides an interface to optimization solvers in general from the framework side. To add another solver, one should create an interface on the solver side. In Section 2.6 we discuss in more detail such interfaces to external optimization solvers.

Move limit strategy

In sequential approximate optimization various move limit strategies are used. The move limit strategy determines at the start of each cycle the size and position of the new search subregion. In Figure 2.3 class MoveLimitStrategy represents the move limit strategy. Method determine_search_region() is used to determine the search subregion based on data of previous cycles of the optimization process. In Section 2.3 some move limit strategies were discussed. Each of these move limit strategies may be added to the framework by inheriting from class MoveLimitStrategy.

2.5 Data

The framework entities use data entities for the input/output relations of the modules. This section describes the structure of the data using UML representations.

2.5.1 Structure

At the start of the optimization sequence, the data of the optimization problem is passed to the sequence layer of the framework. This data includes: the design variables, the objective variables, the constraint variables, and the evaluation models.

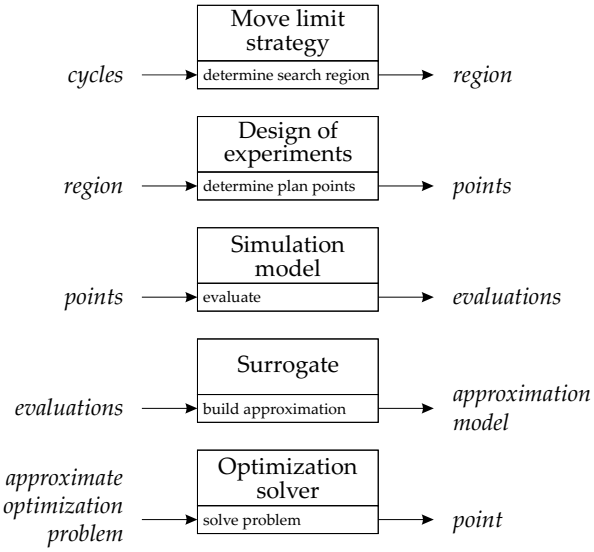

Figure 2.8: Functional relationships between input and output data.

Throughout the optimization process, new data is generated by the different modules of the routine layer. These modules are presented again in Figure 2.8, now illustrating the data that is passed from and to the sequence layer assuming the standard sequence of Figure 2.1. The data is addressed in italics. The following list discusses each of the relations.

- The move limit strategy determines a new search subregion based on data from previous cycles in the optimization process.
- A design of experiments is determined in a specific search subregion which results in a number of plan points.
- The simulation model yields response output by evaluating the plan points.
- The surrogate builds up an approximation model based on the simulation evaluations.
- The solver finds the optimal design point of an approximate optimization subproblem. The sequence layer generates this approximate optimization subproblem based on: the approximation models built in the previous step, the explicit functions, and the search subregion bounds.

Notice, this is just an example, as more data entities may be required for alternative sequences and/or alternative routines.

2.5.2 Classes

The UML diagram of the data is shown in Figure 2.9. Objects of the classes in this diagram are generated during one approximate optimization cycle $c^{(k)}$ and are stored in top-level class Cycle. An object of class Cycle is generated by the sequence layer at the start of a new cycle. During one cycle the following data is added to the object Cycle: search subregion $r^{(k)}$, plan points $p^{(k)}$, evaluations of the simulation model in the plan points, approximate optimization problem $\mathcal{P}_{\mathrm{a}}^{(k)}$, and the approximate optimal design represented by $\mathbf{x}_*^{(k)}$, $f_*^{(k)}$, and $\mathbf{g}_*^{(k)}$. During one cycle, approximate optimization problem $\mathcal{P}_{\mathrm{a}}^{(k)}$ is created by the sequence layer and solved by the approximate optimization solver. Class ApproximateOptimizationProblem is a representation of approximate optimization problem $\mathcal{P}_{\mathrm{a}}^{(k)}$. It contains design variables $\mathbf{x}$, objective variable f, and constraint variables $\mathbf{g}$, search subregion $r^{(k)}$, and approximation model $a^{(k)}$. For example, in the sequence presented in Figure 2.1, the following data objects are created during one cycle:

- First, an object of class Cycle is created. This object will store all data of the current cycle described below.
- An object of class Design is created that includes the cycle initial design point and the corresponding objective and constraint values.
- An object of class Region is created by class MoveLimit and represents the search subregion.
- A number of objects of class Point are created by class DoE and represent the plan points.
- A number of objects of class Evaluation are created by class SimModel and represent the simulation evaluations in the plan points. Besides objective and constraint values, the evaluations may include additional information, such as gradient information or other responses that can be used.
- An object of class ApprModel is created by class Surrogate and represents the approximations.
- An object of class ApproximateOptimizationProblem is created by class Sequence and represents the approximate optimization problem in the search subregion.
- An object of class Point is created by class Solver and represents the approximate optimal design in the search subregion.
- One or more objects of class Evaluation are created by class SimModel and represent the simulation evaluation of the approximate optimal design. These include optional simulation replications, which are needed in case of stochastic responses.
- An object of class Design is created that includes the approximate optimal design point and the corresponding objective and constraint values based on the evaluations in the approximate optimal design point.

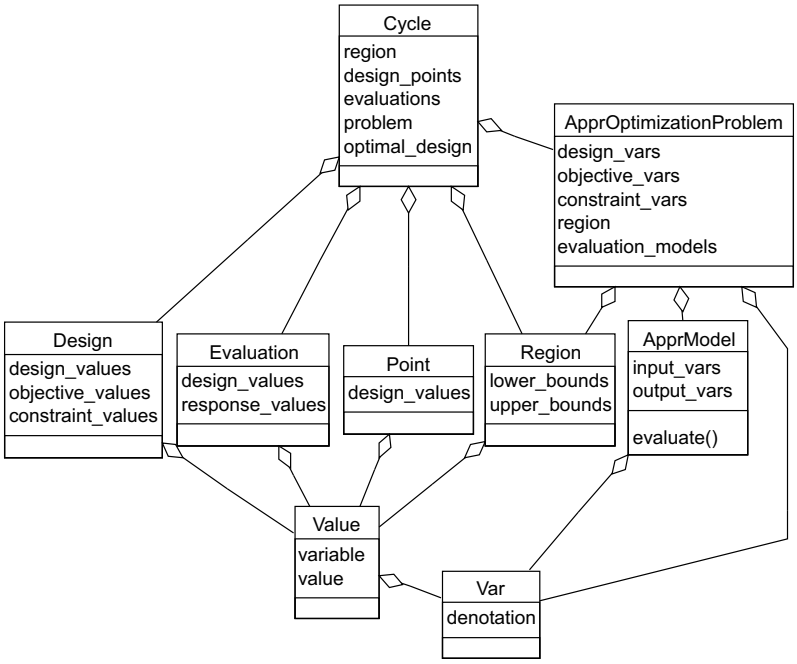

Figure 2.9: UML class diagram of the data. Objects of these classes are created during a cycle of the optimization sequence and stored in the corresponding object of class Cycle.

An important issue is how the data is presented in terms of the variables. The framework can handle design variables, responses variables, objective variables, and constraint variables. For the classes Design, Evaluation, Point, and Region the attributes are a set of values. Each of these values is the representation of a value of a certain variable. For example, class Region has two sets of values: the lower bound values and the upper bound values of each of the design variables x_i. This implies that each value representation is linked to a certain variable of the optimization problem. This is taken care of by class Value by defining two attributes containing: (i) the variable the value is linked to and (ii) the actual value represented by an array of one or more numbers. If additional information has to be stored, like gradients, additional attributes may be needed. For example, gradient information can be handled by class GradientValue, which is a new version of (or inherited from) class Value. Class GradientValue has an extra attribute that stores the gradient values for each design variable direction.

Figure 2.10 shows the inheritance diagram of the variables. For the design variables four different inherited variables are distinguished: the stochastic, continuous, integer, and discrete design variable. The stochastic design variable has an attribute which represents the stochastic distribution from which realizations of this design variable can be generated. For the response variables, two types are distinguished: deterministic and stochastic response variables. Each type of response variable can be subdivided into objective and constraint variables. In the formulation of the optimization problem three types of variables are defined: design, objective, and constraint variables. For design variables $\mathbf{x}$ the following information is available in the problem formulation: type of design variable (continuous, integer, discrete, uncertain), upper bounds $\mathbf{x}_u$, lower bounds $\mathbf{x}_\ell$, initial design values $\mathbf{x}_0$, and initial move limit width $\boldsymbol{\Delta}_0$. For objective variable f and for each of the constraint variables g_i one has to specify whether the variable is deterministic or stochastic. Response variables are needed whenever a simulation response has to be stored. Such a response may not be part of the optimization problem formulation, but can be used, e.g., for response surface building.

2.5.3 Approximation model

As discussed in Section 2.4, in each cycle, class Surrogate creates the approximation models represented by class ApprModel. Figure 2.11 shows an example of the inheritance diagram for the approximation model. This diagram shows the inheritance relationship of an approximation model based on response surfaces (regression) and an approximation model based on radial basis functions. These approximation models are created by the corresponding surrogates shown in Figure 2.6.

2.6 Framework implementation

To obtain an open and flexible environment, we adopted the dynamic programming language Python (Van Rossum and Drake, 2003a) for the implementation of the framework, as well as for the input specification of optimization problem and

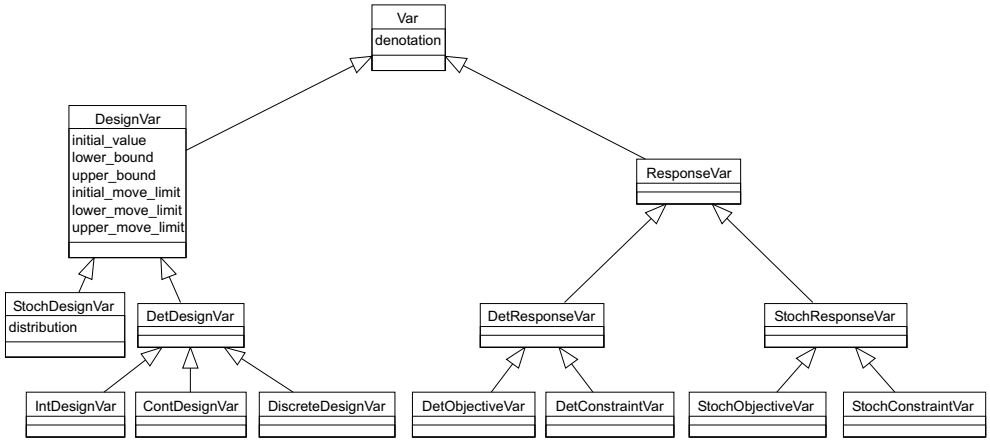

Figure 2.10: UML inheritance diagram of Var. The diagram has four types of design variables: one stochastic and three deterministic. The response variables are divided in deterministic and stochastic response variables.

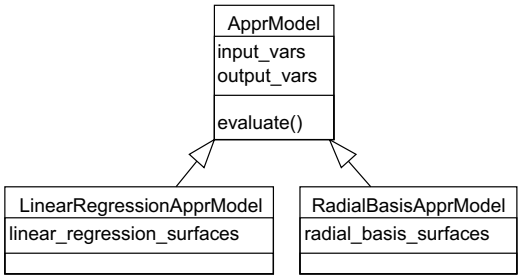

Figure 2.11: Example of UML inheritance diagram of ApprModel.

SAO approach. The developed framework contains a collection of Python classes, methods, and interfaces to external software to support a compact specification of the optimization problem and SAO approach. We selected Python because of its dynamic and highly expressive nature allowing full object-oriented programming with a clear syntax, and its excellent capabilities to extend with other software. Van Rossum and Drake (2003b) describe how external non-Python libraries can be connected to Python code by means of the Python Application Programming Interface (API).

Examples of extensions to the framework are the Matlab Fmincon optimization solver and the FSQP optimization solver. The Fmincon solver is part of Matlab (Mathworks, 2002). The solver is embedded in Python using the PyMat interface (Sterian, 1999). The FSQP solver has been developed by Lawrence et al. (1997) in C++. This solver is embedded in Python by writing an interface to the C++ code. Both the Fmincon and FSQP interface have a method to solve the approximate optimization problem. Another example of an extension is the use of Abaqus (HKS,

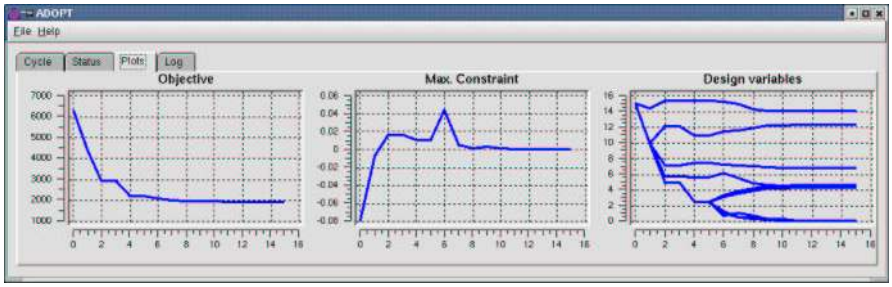

Figure 2.12: Example of a GUI showing the 10-bar truss optimization progress. The GUI is created using the Qt GUI widget-set.

2002) as FEM simulation software. In the next section Abaqus has been used to evaluate the response functions of a ten-bar truss.

For the FSQP solver, we used Python callback functions to specify the objective and constraint functions of the approximate optimization problem that FSQP has to solve. Callback enables the C++ FSQP implementation to call the objective and constraint functions specified in Python code. The objective and constraint functions do not need to be provided or translated into C++ code. The FSQP solver now uses the Python code directly for determining the objective and constraint values. Callback may be used for other optimization solvers as well, possibly also for other external routines, e.g. DoE or fitting routines. Through callback functions, Python specifications of functional relations in the (approximate) optimization problem can be used directly, with only little performance impact. Conversion of user-specified Python code into a different language (e.g. C++) is then not needed.

One can show the progress of the optimization via a graphical user interface (GUI). An example of such a GUI is shown in Figure 2.12. This GUI is created using the Python Qt GUI widget-set of Trolltech (2001). The figure shows three plots. The plots represent for each cycle optimal design: the value of the objective function, the value of the maximum constraint, and the values of each of the design variables, respectively. The results shown in these plots are from the ten-bar truss example discussed in the next section.

2.7 Illustration

Consider the optimization problem presented in Elishakoff et al. (1994). This optimal design problem is presented here to show the flexibility of the framework. We solve the optimization problem by means of a multi-point SAO strategy that combines several different routines, e.g. an external simulation code, a non-linear approximation model, and an external approximate optimization solver. Below follows a short description of the optimal design problem. In Subsection 2.7.1 and 2.7.2 this optimization problem is solved under deterministic loading and uncertain loading, respectively.

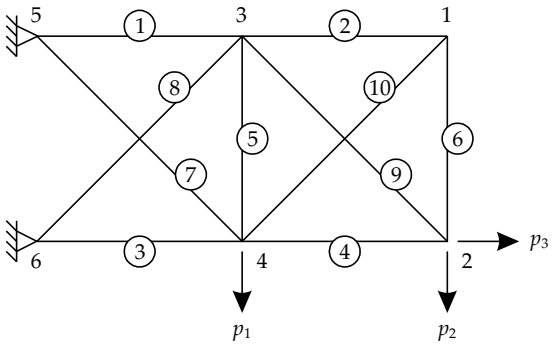

Figure 2.13: 10-bar truss structure (Elishakoff et al., 1994).

The optimal design problem concerns the ten-bar truss structure as shown in Figure 2.13, with a bay length L of 360 inch. The optimization problem is defined to find the cross-sectional areas that will minimize the mass of the structure subject to ten stress constraints (one for each bar) and one displacement constraint (vertical displacement of node 2). For stress constraint in bar i holds: $|\sigma_i| \leq 25$ ksi, except for bar 9, where $|\sigma_9| \leq 75$ ksi. The vertical displacement in node 2 is restricted to 5 inch. The lower bounds on the cross-sectional areas are 0.1 inch^2 for all bars. The material characteristics of the truss elements are: modulus of elasticity $E = 10^4$ ksi and material density $\rho = 0.10$ lb/inch^3. Stresses and displacements are computed here using the FEM package Abaqus (HKS, 2002). Gradients are not computed. The mass of the structure is represented by an explicit relation:

$$f = \rho L \left[\sum_{i=1}^{6} x_i + \sqrt{2} \sum_{i=7}^{10} x_i \right] \tag{2.5}$$

The ten-bar truss is subject to three uncertain loads p_1, p_2, and p_3. The nominal values of the loads are 100, 100, and 400 kips, respectively. The uncertain loads are bounded by $\pm 10\%$ of the nominal values. Elishakoff et al. (1994) considered the following optimization problems: (A) optimal truss under fixed nominal loads, (B) optimal truss under fixed highest load combination, and (C) optimal truss under uncertainty. For the highest load combination the upper bounds for each uncertain load were taken: 110, 110, and 440 kips, respectively.

2.7.1 Fixed loads

Fixed load problems (A) and (B) can be solved using the standard SAO sequence of Figure 2.1. The ten stress constraints and the single displacement constraint are approximated using response surface models with first order linear and reciprocal regression terms without interaction:

$$\tilde{g}_j(x) = \beta_{0,j} + \sum_{i=1}^{n} \beta_{i,j} x_i + \sum_{i=1}^{n} \beta_{i+n,j} \frac{1}{x_i} \tag{2.6}$$

Each response surface consists of 21 linear regression terms. The response surfaces are built using a star design of experiments evaluating points on three levels on the main axes. The experimental design includes the center point and the 'star' points at the center of each face of the design region.

The specification used to solve optimization problem (A) is shown in Figure 2.14. The definition of the simulation model starts on line 2. Simulation model class *Constraints* is used to calculate the stress and displacement constraints. The *evaluate*() method needs design values X as an input for which responses R have to be calculated. Both X and R are dictionaries containing the values of the design variables and constraint variables, respectively. Function *abaqus_nominal*() is a representation of the true Abaqus model (not presented here) used to calculate stresses S and displacements U based on design values X. This function also needs a second parameter, which is assigned on line 4 as a vector containing the fixed loads. The return values S and U are vector arrays. Array S contains 10 stresses (10 bars) and array U contains 1 displacement. These arrays are used to calculate responses R in lines 5–8. The objective function, the mass of the structure, is calculated by explicit model class *Mass* on lines 10–18.

The specification of the optimization problem formulation starts on line 19. All variables are instantiated and collected in list of design variables x, list of objective variable f, and list of constraint variables g. The design variables need additional arguments in the following order: name, initial design value, lower bound on design variable, upper bound on design variable, initial move limit width, lower bound on move limit width, and upper bound of move limit width. The models are instantiated and collected in list of models m on lines 26–28. The first model (m1) calculates all the constraints by means of the Abaqus FEM model, whereas the second model (m2) calculates the objective function by means of an explicit relation. Optimization problem p is instantiated on line 29.

On lines 31–41, the classes design of experiments doe, regression surrogate surrogate, optimization solver solver, and move limit strategy mvl are instantiated. These instances all refer to classes in the routine layer. Regression surrogate surrogate represents the approximate constraint functions. For this purpose, 11 different approximate relations are instantiated (for each constraint one relation) on lines 32–36, following (2.6). Here, terms_matlab and terms_python are used for the Matlab and Python representations, respectively, of the pure linear and reciprocal terms. The instantiations use linear regression terms specified as lists. For each design variable, the Python constructs on lines 35 and 36 add two linear regression terms each represented by a string. Iterator i is included in the strings by means of string formatting expression %. The Matlab representations are needed for the regression of the surrogate functions in Matlab. The Python representations are needed for the PyFsqpSolver to evaluate the corresponding approximation models.

In this example we used the standard sequence of the framework which required 12 cycles and 274 function evaluations for both optimization problem (A) and (B) to obtain the same results as reported by Elishakoff et al. (1994).

```
Problem layer
class ConstraintsModel(SimModel):    # simulation model for evaluation of constraints
    def evaluate(self, g, X, R):
        S, U = abaqus_nominal(X, [100.0, 100.0, 400.0])    # Abaqus FEM model
        for i in range(8): R[g[i]] = abs(S[i]/25.0) - 1    # stress constraints 1–8
        R[g[8]] = abs(S[8]/75.0) - 1    # stress constraint 9
        R[g[9]] = abs(S[9]/25.0) - 1    # stress constraint 10
        R[g[10]] = abs(U[0]/5.0) - 1    # displacement constraint
        return R
class MassModel(ExplicitModel):    # explicit model of mass objective following (2.5)
    def evaluate(self, f, X, R):
        rho = 0.1
        L = 360
        W = 0.0
        for i in [0, 1, 2, 3, 4, 5]: W = W + X[x[i]]
        for i in [6, 7, 8, 9]: W = W + sqrt(2) * X[x[i]]
        R[f] = rho * L * W
        return R
n = 10    # number of design variables
m = 11    # number of constraint functions
x = []    # design variables with corresponding attributes
for i in range(n): x.append(DesignVar('area', 15.0, 0.1, 20.0, 10.0, 0.1, 10.0)
f = [DetObjectiveVar('mass')]    # objective variable
g = []    # constraint variables
for i in range(m): g.append(DetConstraintVar('constraint'))
m1 = ConstraintsModel(x, g)    # simulation model
m2 = MassModel(x, f)    # explicit model
m = [m1, m2]
p = OptimizationProblem(x, f, g, m)    # optimization problem
Routine layer
doe = StarDoe()    # design of experiments
terms_matlab = ['1']    # linear regression terms in Matlab syntax following (2.6)
terms_python = ['1']    # linear regression terms in Python syntax following (2.6)
for i in range(n):
    terms_matlab = terms_matlab + ['x(:,%i)' % (i + 1),'1./x(:,%i)' % (i + 1)]
    terms_python = terms_python + ['x[%i]' % i,'1/x[%i]' % i]
rel = []    # linear regression relations
for i in range(m): rel.append(LinRegRelation(x, g[i], terms_matlab, terms_python)
surrogate = [RegressionSurrogate(x, g, rel)]    # surrogate
solver = PyFsqpSolver()    # approximate optimization solver
mvl = StandardMoveLimitStrategy()    # move limit strategy
Sequence layer
sequence = StandardSequence(p, surrogate, doe, solver, mvl)    # sequence following Fig. 2.1
sequence.start_sequence()    # starting sequence
```

Figure 2.14: Example of the Python specification of the ten-bar truss optimal design problem (A). This code can also be used to specify optimal design problem (B) by changing the load vector on line 4.

2.7.2 Uncertain loads

Optimization case (C) with uncertain loads is defined as an anti-optimization problem according to Elishakoff et al. (1994):

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}), \\ \text{subject to} \quad & \max_{\mathbf{p}} \; g_j(\mathbf{x}, \mathbf{p}) \leq \mathbf{0}, j = 1, \ldots, q, \\ & \mathbf{x}_\ell \leq \mathbf{x} \leq \mathbf{x}_\mathrm{u}, \text{ and} \\ & \mathbf{p}_\ell \leq \mathbf{p} \leq \mathbf{p}_\mathrm{u} \end{aligned} \tag{2.7}$$

which finds the optimal $\mathbf{x}$ values with minimum value for f such that the constraints g_j are satisfied for all values of $\mathbf{p}$. The anti-optimization basically consists of finding the worst value of $\mathbf{p}$ for each constraint g_j.

Several anti-optimization schemes for SAO are available (see Gurav et al., 2003). We apply the cycle based alternating method that performs an anti-optimization to determine the worst uncertainty settings. This class has a similar flow as the standard sequence of Figure 2.1, but performs an anti-optimization for the initial design and each cycle optimal design (in step 1 and 7). At the start of a new cycle the set of worst uncertainty values for each constraint is known. In step 4, evaluation of a plan point in the experimental design then requires that each constraint is evaluated for its corresponding worst uncertainty setting. Consequently, the approximate optimization step also accounts for the worst uncertainty setting for each constraint. Finally, the anti-optimization again computes new values for uncertainties $\mathbf{p}^{(k)}$ at the end of the cycle.

For the specification of optimization problem (C) we need some adaption of the Python specification of Section 2.7.1 to implement the cycle-based alternating method. Figure 2.15 shows the code that was added to the original code in Figure 2.14. In the problem layer the simulation model has been slightly changed. Since the loads are not fixed anymore, function *abaqus_uncertain* is used and the parameter that defined the loads on line 4 is omitted. Input parameter X now holds both the values of the design variables and the uncertainty variables. Uncertainties u are instantiated as three extra design variables. On line 26 the instantiation of the simulation model is slightly changed such that the input variables now includes both the design variables x and the uncertainty variables u.

For this example we developed a new anti-optimization sequence. This new sequence follows the cycle based alternating anti-optimization method as explained before and is represented by class *AntiOptimizationSequence*. The Python specification of this new anti-optimization sequence is not presented here. To use the anti-optimization sequence, the routine layer has been extended with instantiations needed for the anti-optimization. The anti-optimization itself is performed by means of the SAO method of Figure 2.1, using the following specified routines in the routine layer: two-level full-factorial design of experiments, pure linear regression approximations for each constraint, the Matlab linear programming algorithm to solve the linear approximate optimization problem, and the standard move limit strategy.

```
Problem layer
class UncConstraintsModel(SimModel):    # simulation model for evaluation of constraints
    def evaluate(self, g, X, R):
        S, U = abaqus_uncertain(X)    # Abaqus FEM model
        ⋮
u = []    # uncertainties and corresponding attributes
u.append(DesignVar('load p1', 100.0, 90.0, 110.0, 20.0, 1.0, 20.0))
u.append(DesignVar('load p2', 100.0, 90.0, 110.0, 20.0, 1.0, 20.0))
u.append(DesignVar('load p3', 400.0, 360.0, 440.0, 40.0, 2.0, 40.0))
m1 = UncConstraintsModel(x + u, g)    # simulation model
        ⋮
Routine layer
        ⋮
doe_anti = FullDoe(2)    # design of experiments used in anti-optimization
rel_anti = []    # relations used in anti-optimization
for i ∈ range(11): rel_anti.append(PureLinearRelation(u, g[i]))
surrogate_anti = [RegressionSurrogate(u, g, rel_anti)]    # surrogate used in anti-optimization
solver_anti = MatlabLinprogSolver()    # optimization solver for anti-optimization
mvl_anti = StandardMoveLimitStrategy()    # move limit strategy in anti-optimization
Sequence layer
sequence = AntiOptimizationSequence(p, u, surrogate, doe, solver, mvl, surrogate_anti,
                                    doe_anti, solver_anti, mvl_anti)
sequence.start_sequence()
```

Figure 2.15: Example of the Python specification of the ten-bar truss optimal design problem (C) using the cycle-based alternating method with anti-optimization.

To reduce the computational effort in the anti-optimization step, Elishakoff et al. (1994) showed that for this example the solution of the anti-optimization is given by one of the eight vertices in the uncertainty domain. The eight vertices are the 8 different combinations of minimal and maximal uncertainty parameters for the three uncertain load vectors. It is sufficient for the anti-optimization to simulate these eight uncertainty combinations for fixed **x** and select for each constraint g_j the uncertainty set that maximizes its constraint value.

We have also implemented this method of Elishakoff et al. (1994). Class *AntiOptimizationSequence* needs a new method that searches for the worst uncertainties without the full anti-optimization. Method *anti_sequence*() is specified for this purpose. We developed class *AntiTenBarTrussSequence* shown in Figure 2.16. The eight vertices are evaluated using a two-level full-factorial design for fixed design point X coming from the outer optimization loop. The worst uncertainties are stored in dictionary U which represents the worst uncertainty for each of the constraints.

The two anti-optimization sequences required both 15 cycles to obtain the same results reported by Elishakoff et al. (1994). The number of function evaluations equals 9163 FEM analyses for the full anti-optimization and 1766 FEM analyses for

Problem layer

```
# Elishakoff's anti-optimization method
class AntiTenBarTrussSequence (AntiOptimizationSequence):
    # user-defined method to determine the worst uncertainties
    def anti_sequence (self, pa, X, U):
        # determining the eight vertices
        pl = FullDoe (2).generate_planpoints (pa.design_region ())
        # the simulation model of the approximate optimization problem
        e = pa.evaluator ()
        # the constraints of the approximate optimization problem
        g = pa.constraints ()
        for j ∈ pl:
            # evaluate vertice j for fixed value X
            R = e._evaluate (j, X)
            for i ∈ g:
                # if a simulation results in a worse constraint value, this
                # constraint value and corresponding uncertainty is saved
                if R[i] > U[i].value ():
                    U[i] = (j, R[i])
        return U
        ⋮
```

Sequence layer

```
43 sequence = AntiTenBarTrussSequence (p, u, doe, solver, mvl)
44 sequence.start_sequence ()
```

Figure 2.16: Example of the Python specification of the ten-bar truss optimal design problem (C) using the method of Elishakoff et al. (1994).

Elishakoff's method. The optimization progress of problem (C) using Elishakoff's method is shown in Figure 2.12.

2.8 Conclusion

The sequential approximate optimization framework is an open environment for the specification and implementation of SAO techniques. The structure of the framework distinguishes three basic layers: an optimization problem layer, an SAO sequence layer, and a numerical routines layer. The framework starts from some predefined SAO sequence classes as well as a toolbox of numerical routines to carry out basic steps in the SAO sequence, e.g. related to design of experiments, approximation building, and optimization solvers. The framework gives an easy access to the contents of each three layers. On this basis, the framework enables one to redefine or implement new SAO sequences, and allows the use of other (third-party) numerical routines currently not available from the framework toolbox.

The framework has been implemented in Python. The Python language is compact and highly expressive, which is advantageous for the specification of the opti-

mization problem, the SAO sequence, as well as for user modifications in the framework. The specification directly results in an implementation due to the fact that Python is used for both the specification of optimization problem and sequence as well as the implementation of the framework itself. We can use Python language constructs in the specification of optimization problem and optimization approach, e.g. higher data types, such as lists, tuples, arrays, classes, as well as, iterators, file parsing capabilities, and functions. We also take advantage of Python in interfacing the framework with external numerical routines and simulation software.

We followed an object-oriented approach to obtain a flexible and extensible framework. The key feature of the framework is the ability to add new functionality by adding user-defined routines. The only requirement is that the new routine has to meet the input and output requirements of the module. Within the new routine, the functionality of the base classes of the framework and more general Python constructs can be used. New modules can also be introduced based on the existing base classes of the framework.

At present, the tool holds basic numerical functionality for standard SAO approaches, e.g. design of experiments and linear regression. Several Matlab routines have been included. The ten-bar structure optimal design problem illustrated the flexibility of the framework. Two design cases were considered: deterministic and uncertain loading. The latter resulted in a nested optimization scheme. In both cases the optimization problem and the SAO approach could be completely specified. The second case required only small modifications on the basis of the specification of the first case. The optimization process itself showed good results by finding the same optimum as reported in the literature. In this example, the framework was connected to three external software packages, being Matlab, Abaqus, and the FSQP solver written in C++.

Chapter 3

Characterization of Operational Time Variability

Operational time variability is one of the key parameters determining the average cycle time of lots. Many different sources of variability can be identified such as machine breakdowns, setup, and operator availability. However, an appropriate measure to quantify variability is missing. Measures such as Overall Equipment Effectiveness (OEE) used in the semiconductor industry are entirely based on mean value analysis and do not include variances.

The main contribution of this paper is the development of a new algorithm that enables estimation of the mean effective process time t_e and the squared coefficient of variation c_e^2 of a multiple machine workstation from real fab data. The algorithm formalizes the effective process time definitions as known in the literature. The algorithm quantifies the claims of machine capacity by lots, which include time losses due to down time, setup time, and other irregularities. The estimated t_e and c_e^2 values can be interpreted in accordance with the well-known $G/G/m$ queueing relations. Some test examples as well as an elaborate case from the semiconductor industry show the potential of the new effective process time algorithm for cycle time reduction programs.

3.1 Introduction

Equipment in semiconductor manufacturing is subject to many sources of variability. An important source is machine down time, which occurs due to highly complex and technologically advanced semiconductor manufacturing processes (Uzsoy et al., 1992). Many other corrupting operational influences are also present, such as batch-

Reproduced from: Jacobs, J. H., Etman, L. F. P., Van Campen, E. J. J., and Rooda, J. E. (2003). Characterization of flow time variability using effective process times. *IEEE Transactions on Semiconductor Manufacturing*, 16(3):511–520.

ing, hot lots, rework, setup, and operator availability. All together, they introduce a substantial amount of variability in the interarrival and operational times of the lots during their flow through the fab.

Queue times are mainly influenced by variability and utilization. High utilization is necessary in the semiconductor industry in order to maximize productivity and minimize costs. In combination with large variability, high utilization leads to large cycle times for the lots. IC manufacturers are under high pressure nowadays to reduce cycle times and improve delivery performance. Lu et al. (1994) developed scheduling policies that attempt to reduce various fluctuations in the flow of the lots in order to reduce mean and variance of cycle time. Park et al. (2001) describe the operating curve as a means to evaluate the trade-off between cycle time, throughput, and work-in-progress. Variability plays a central role in this trade-off. Schömig (1999) stated that the corrupting influence of variability on the cycle time is often overlooked, and semiconductor industry should aim at reducing variability to provide low cycle times. Therefore, identification and reduction of the main sources of variability are key actions to improve upon the compromise between throughput and cycle time.

Unfortunately, in the semiconductor industry no measures for operational time variability are used. The Overall Equipment Effectiveness (OEE) has been introduced by SEMI (Ames et al., 1995). This measure is based on mean values with respect to availability, productivity, and yield. It includes for example mean time between failure and mean time to repair to characterize machine down time, but it fails to include variances. Hopp and Spearman (2001, Section 8.4) show with a simple example that, besides the average capacity, the fluctuations of capacity in time should also be included to make the correct conclusion on how well a machine is performing. Taking into account only average capacity may lead to the wrong conclusion.

A suitable measure that quantifies the total process time variability is still missing. Such a measure would be highly valuable in variability reduction programs. Sturm et al. (1999) observed that it is impossible to measure each individual source of variability. Instead, they measured cycle time distributions at workstations, and used these in their simulation model. However, in these distributions the effects of utilization and variability are combined. Another approach is proposed by Hopp and Spearman (2001). They introduce the so-called effective process time, and describe it as the time seen by lots from a logistical point of view. Basically, the effective process time includes all time losses due to failure, setup, and any other source of variability. A similar description is given by Sattler (1996) who defined the effective process time as all cycle time except waiting for another lot. It includes waiting for machine down time and operator availability and a variety of other activities.

Sattler (1996) noticed that her definition of effective process time is difficult to measure. The same difficulty holds for the description given by Hopp and Spearman (2001). But the basic idea of the effective process time to include time losses does give a starting point for computing effective process time realizations of lots when a list of events is available with arrival and departure times of lots. Since the semiconductor industry is highly automated, this track-in and track-out data is generally available. We propose a new method to actually compute effective process

times from such a data set. In this way we are able to estimate the mean and variance of the effective process time of a workstation. This gives the desired quantification of operational time variability. The approach is illustrated using real fab data from a Philips Semiconductors wafer fabrication facility.

3.2 Performance measurement

Wafer fabs combine uncertain yields and unreliable machines in a re-entrant process flow. In order to improve machine productivity, SEMI (Ames et al., 1995) defined the Overall Equipment Effectiveness (OEE). OEE separates machines productivity into three basic corrective action categories: availability, performance, and quality. Availability efficiency is the fraction of time that a machine is in a condition to perform its intended function. Performance efficiency is the fraction of machine uptime that a machine is processing actual units at theoretically efficient rates. Finally, quality efficiency is the theoretical production time for effective units divided by the theoretical production time of actual units. Typically, OEE includes several sources of variability such as down times and monitoring times. However, OEE is only based on mean values.

Besides mean process time, the average queue time is determined by utilization and variability. Hopp and Spearman (2001) use the following approximation for average queue time of wafers in a $G/G/m$ queueing system, where m denotes the number of identical machines:

$$t_q = \frac{c_a^2 + c_e^2}{2} \cdot \frac{u^{(\sqrt{2(m+1)}-1)}}{m(1-u)} \cdot t_e \tag{3.1}$$

with the utilization defined as:

$$u = \frac{t_e}{t_a m} \tag{3.2}$$

The first term of (3.1) represents the variability which is the sum of the squared coefficients of variation of the interarrival times c_a^2 and the process times c_e^2. The squared coefficient of variation is defined as the quotient of variance and the mean squared. Thus, $c_a^2 = \sigma_a^2/t_a^2$, and $c_e^2 = \sigma_e^2/t_e^2$, where t_a and t_e are the mean interarrival time and mean process time, respectively. Hopp and Spearman (2001) use the effective process time paradigm: t_e and c_e^2 include the effects of operational time losses due to machine down time, setup, rework, and other irregularities. Compared to the theoretical process time t_0, this typically means $t_e > t_0$ and $c_e^2 > c_0^2$. In accordance with Hopp and Spearman (2001) we call $c_e^2 < 0.5$ lowly variable, $0.5 < c_e^2 < 1.75$ moderately variable, and $c_e^2 > 1.75$ highly variable.

Equation (3.1) clearly identifies the contribution of utilization and variability. Cycle time increases linearly with the squared coefficients of variation of interarrival times and effective process times, and increases nonlinearly with utilization. To reduce the mean waiting time, there are two possible courses of action. The first is to reduce the loss of capacity due to irregularities. This gives a smaller mean effective

process time t_e, which also means a lower utilization. This part is covered by performance measures such as OEE, and focuses on the improvement of bottleneck workstations with high utilizations. If the mean capacity loss cannot be further reduced, the second action is to reduce the variation of the irregularities, giving a smaller variability term, c_e^2. The OEE fails to cover this term. Equipment with large operational variability can have large effect on cycle time even if they are not bottlenecks.

An important property is that variability propagates through the fab. The departure flow of a workstation determines the arrival flow to the next workstation in the flow line. Variability in the departure flow of a workstation is determined by utilization, variability in arrivals, and variability in processing. The following linking equation gives an approximation of this relation. The squared departure coefficient of variation c_d^2 can be estimated by (Hopp and Spearman, 2001; Buzacott and Shanthikumar, 1993):

$$c_d^2 = 1 + (1 - u^2)(c_a^2 - 1) + \frac{u^2}{\sqrt{m}}(c_e^2 - 1) \tag{3.3}$$

This means that for low utilizations, the flow variability of the departing wafers equals the variability of the arriving flow to the workstation, while for high utilizations, the flow variability of the departing wafers is proportional to the effective process time variability. To be cost effective, wafer fabs operate at high machine utilizations. Thus, reducing operational time variability at one workstation will positively influence the arriving wafer flow to its successors.

Equation (3.1) implies that the mean effective process time and the corresponding squared coefficient of variation are two fundamental process parameters with respect to cycle time performance. To use these parameters as performance measures, t_e and c_e^2 have to be determined from actual fab data. However, the description as given by Hopp and Spearman (2001) that the effective process time is *the time seen by lots from a logistical point of view* does not define how effective process times should actually be measured from such a data set.

3.3 How to measure effective process time?

We have formalized the Effective Process Time (EPT) definition and propose a new algorithm to compute EPTs of machines in a workstation from real-time fab data. By workstation we mean one or more machines that perform a similar operation and that share a single queue. The EPT definitions of Hopp and Spearman (2001), and Sattler (1996) include the theoretical process time as well as setup time, breakdown, operator availability, and all other operational times due to variability effects. For the cycle time of a lot it is of no importance whether the lot is waiting for an operator or waiting for a machine that is being monitored. Generally stated, EPT can be defined as the total amount of time a lot could have been, or actually was, processed on a machine. So, the EPT is the total amount of time a lot *claims* capacity of a machine, even if it is not yet being processed.

The new algorithm is developed such that it enables calculation of EPT from a list of events. This list of events consists of the arrival and departure times of the

time	lot	event
0	1	Arrival
1	2	A
2	1	Depart
3	3	A
5	2	D
6	3	D
7	4	A
9	4	D

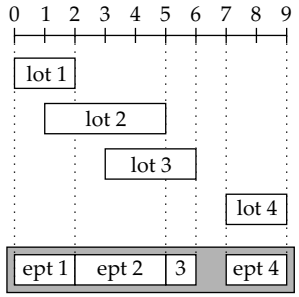

Figure 3.1: Single machine: FIFO dispatching.

lots at a certain workstation, and the machine identification number the lot has been processed on. We start with investigating the EPT definition for a single-machine workstation with First-In-First-Out (FIFO) dispatching. This EPT definition is extended to include other dispatching policies as well. Finally, the EPT definition is generalized to a multiple machine workstation.

3.3.1 Single machine, FIFO dispatching

Consider a workstation with FIFO dispatching that consists of a single queue and a single machine. This single machine setup is used to formalize the conceptual idea that the EPT is the total amount of time a lot could have been or actually was processed on a machine. The event history with respect to arrival and departure can be visualized by a Gantt chart. An example is presented in Figure 3.1. The Gantt chart shows four lots which were processed in FIFO order on a single machine. The first lot arrived at $t = 0$ and departed at $t = 2$. The arrival and departure times of the other lots are depicted in the same way. Actual process times are not needed for determining the EPTs, and are therefore not depicted. The resulting EPTs are shown in the gray box at the bottom of Figure 3.1. The following paragraph explains how these EPTs have been determined.

Initially, no lots are present in the workstation, i.e. no lots are queued and no lot is in process. Since at $t = 0$ the first lot arrives at the workstation, this lot immediately claims capacity of the machine, independent of whether it is queued for a while or processed immediately. Before the first lot departs, a second lot arrives at $t = 1$. Since a FIFO dispatch policy is used, the first arrived lot still claims capacity of the machine. When the first lot has finished processing, and departs from the machine, the total amount of time this first lot has claimed capacity is called a realization of effective process time. From this point of time ($t = 2$), the second lot now claims capacity of the machine until the lot departs. It does not make a difference if new lots are arriving, like the third lot at $t = 3$. Therefore, the second EPT realization is the time between the departure of the first lot and the second lot.

In general, for each lot that is processed next on the machine it holds that a realization of effective process time is calculated as follows: EPT is the total amount of time the lot was queued or processed between the departure of the previous lot and

time	lot	event
0	1	Arrival
1	2	A
2	1	Depart
3	3	A
5	3	D
6	2	D
7	4	A
9	4	D

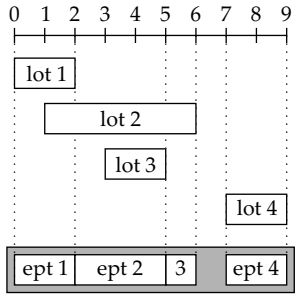

Figure 3.2: Single machine: general dispatching.

its own departure. When enough individual EPTs have been determined, a complete EPT distribution arises. For the machine, an estimate of mean effective process time t_e and coefficient of variation c_e^2 can be calculated from the complete set of individual EPTs.

3.3.2 Single machine, general dispatching

If a single machine is considered, the lots in the system claim capacity of this particular machine. From the machine's point of view it does not matter which lot claims its capacity. If two lots are queued, capacity of the machine is claimed from the time point the first lot has arrived in the queue. No matter when the second lot arrives, capacity of the machine is claimed until one of the two lots departs. We assume that the order in which the lots are processed does not affect the EPT calculation. As a consequence, the EPT does not depend on the schedule, but only on the arrival and departure times of the lots. Thus, during the period that at least one lot is present in the workstation, the capacity of the machine is claimed until a lot departs from the workstation. The EPT calculation does not take into account *which* lot arrives or departs. This corresponds with observations that dispatch rules that do not use information on the individual process times of lots have no influence on the mean cycle time (see Buzacott and Shanthikumar, 1993, Section 3.6).

In Figure 3.2 an event history and Gantt chart of a non-FIFO schedule is shown to illustrate how the above mentioned rationale for general dispatching affects the EPT definition. When lot 1 leaves, lot 2 is already available in the queue, but does not start processing immediately. Lot 2 is kept in queue, but already starts claiming capacity of the machine, since the machine is kept idle. Assume that lot 3 is a hot lot and arrives while lot 2 is still kept in queue. Due to the priority of lot 3, this lot is processed first on the machine. For this reason, at the arrival of lot 3, this hot lot claims capacity of the machine. But, for the machine it does not make any difference which lot is processed next. During the complete time period between the departure of lot 1 and the departure of lot 3, capacity of the machine was continuously claimed by any lot. Therefore, the complete time period is a single EPT realization. It is of no concern whether this realization is based on the presence of a single lot or multiple lots. Notice that the Gantt chart of Figure 3.2 has equal arrival and departure times

```
n := 0
loop
    read τ, ev
    if ev = "A" then
        if n = 0 then s := τ                (i)
        elseif n > 0 then skip              (ii)
        endif
        n := n + 1
    elseif ev = "D" then
        write τ − s
        n := n − 1
        if n = 0 then skip                  (iii)
        elseif n > 0 then s := τ            (iv)
        endif
    endif
endloop
```

Figure 3.3: Algorithm SM – single machine.

compared to Figure 3.1. Although the schedules produce the lots in different orders, the EPT calculation delivers equal EPT realizations.

Summarizing, the machine does not need to know which lot is claiming capacity. The EPT is the total amount of time a single lot or different lots are claiming capacity of the machine until a lot departs. Thus, whenever there is no lot in the workstation (queue empty and machine idle), capacity of the machine is not claimed. These time periods do not belong to EPT. But as soon as a new lot arrives, the next EPT realization will be the time between this arrival and the next departure of any lot, not necessarily the first newly arrived. With lots present in the single machine workstation, the EPT is the time between two departures of two lots. This holds until no more lots are present in the workstation.

Algorithm SM – single machine workstation

An algorithm to calculate EPTs is proposed in Figure 3.3. The algorithm considers a single machine workstation and a general dispatch rule. It is assumed that at the start of the period the workstation is empty. When an event occurs, the algorithm reads the current time of the event τ and the type of the event ev. An event ev can be either an arrival of a lot ("A") or a departure of a lot ("D"). The number of lots present in the workstation is denoted by n.

If a lot arrives, the workstation can be in two different states: (i) the workstation is empty; that means the number of lots n present in the system equals 0, or (ii) the workstation is not empty, and thus $n > 0$. In the first case ($n = 0$), capacity of the machine is not claimed until the lot arrives at time τ. From this point the capacity of the machine is claimed until a lot departs. Therefore, the start of EPT is set at $s := \tau$. In the second case, if a lot arrives and the workstation is not empty ($n > 0$), the start of EPT calculation has already been set by a lot that arrived earlier. In this

case, nothing has to be done, represented by the skip statement. Finally, the number of lots present in the system has to be updated to the new value: $n := n + 1$.

If a lot departs, a realization of EPT can be calculated which equals the time between the departure time τ and the time the start of EPT was set: s. Thus, the realization of EPT equals $\tau - s$. This value is written. Afterwards, the number of lots present in the system has to be updated: $n := n - 1$. Now again two different states can occur: (iii) the workstation is empty, or (iv) the workstation is not empty. If the workstation is empty, capacity of the machine is not claimed by a new lot and the start of the EPT is not set. If the workstation is not empty, capacity of the machine is immediately claimed by one of the lots still left in the system and s is set to the event time τ.

In the practical case that lots are already present in the workstation when the algorithm is started, then the first departure of a lot cannot result in an EPT realization, since s has not been properly set. But, after this first departure, s can be properly set, and the algorithm can continue with all the other lots.

3.3.3 Multiple machines

The EPT algorithm is generalized to cover multiple machines with general dispatching. The workstation now consists of a single queue which feeds a number of parallel machines. Again we follow the concept that a departure of a lot from a machine yields a new EPT realization. It holds for each of the machines in the workstation. The number of machines for which capacity is claimed (m_e) should be equal to the minimum of the number of machines (m) and the number of lots (n) present in the workstation:

$$m_e = \min(m, n) \tag{3.4}$$

Usually if two lots are present in the workstation they will be processed by two different machines. Then it is clear that capacity of both machines is claimed ($m_e = 2$). But there are some other possibilities where it is less clear from which machine capacity is claimed. Imagine, for example, the following situation of a workstation with 2 machines. A lot is processed on the first machine and another lot is waiting to be processed on this first machine too. Capacity of this first machine can not be claimed twice, but according to (3.4) capacity is still claimed for two machines. One could say that the second lot claims capacity of the second machine now, since it is the only available machine left. However, the actual value of the next EPT realization depends on whether or not a third lot will arrive before the first lot has finished and on which machine this third lot will be processed. This is explained in the following two paragraphs.

Consider the scenario that either no new lot arrives before the first one is finished, or that a new lot arrives that will also be processed on the first machine. In both of these situations, where the second machine stays idle, we follow (3.4) and assume that the capacity of the second machine is claimed as long as at least two lots are present in the system. So, upon departure of a lot, we want the multiple machine EPT algorithm to compute the EPT realization according to the time this lot has claimed

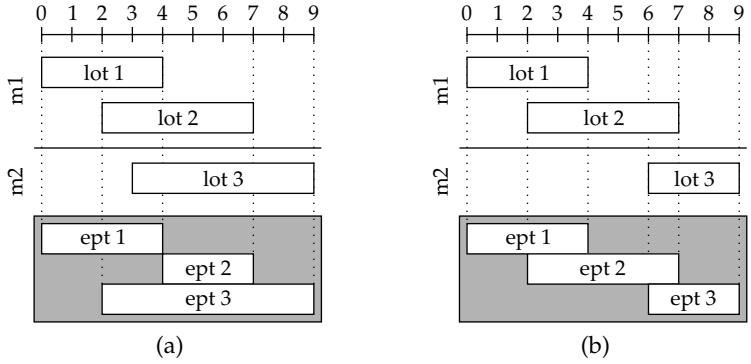

Figure 3.4: Multiple machines.

capacity on some machine, either the first one, or the second one, or both, as long as the claim is a continuous one. Consequently, an EPT realization in the multiple machine case cannot always be assigned to one particular machine.

We need the assumption of a continuous claim of capacity. This is explained by slightly changing the previous example. Consider again the two machines. One machine is processing a lot and capacity of this machine is claimed. Another lot is queued and will be processed in the future on the first machine too. As in the previous example, this second lot claims capacity of the second machine. Now imagine that the next lot that arrives will not be processed on the first machine, but will be processed on the second machine which has been idle so far. This third lot may arrive either before or after the first lot departs from the first machine, as shown in Figure 3.4(a) and 3.4(b), respectively. If it arrives before the first lot is finished but after the second lot has arrived (Figure 3.4(a)), then you might end up with a discontinuous claim of capacity and discontinuous EPTs if start and end times are not properly chosen: at time $t = 2$ lot 2 claims capacity from idle machine 2; then at $t = 3$ lot 3 arrives and takes over this claim of capacity of machine 2; for lot 2 the claim would end, and be resumed at $t = 4$ when machine 1 becomes available again. Instead, we describe it as: at time $t = 2$ a lot claims capacity from idle machine 2; at $t = 3$ another lot arrives and capacity of machine 2 is still being claimed; at $t = 4$ lot 1 departs which means the start of a new claim on machine 1; the respective EPTs run as long as the claims continue without interruption. The same consideration holds for the Gantt chart in Figure 3.4(b) with the difference that directly after the arrival of lot 2, this lot continuously claims capacity of the machines until it departs. In that case, the total time lot 2 is present in the workstation is a single EPT realization. Figure 3.4 illustrates the EPT realizations we obtain in this way at times $t = 7$ and $t = 9$ for both situations. Consequently, an EPT realization in the multiple machine situation cannot always be assigned to one particular lot. We already observed that for the single machine workstation.

Summarizing, for multiple machines capacity can be claimed in two different ways: (i) capacity is claimed by presence of a lot that will be processed on a certain machine. This means that it is clear which lot claims which machine, (ii) capacity is

claimed but it cannot be assigned to one specific lot or machine. In case of multiple idle machines it is even impossible to determine which machine is claimed and which not.

Algorithm MM – multiple machine workstation

The proposed EPT algorithm for multiple machine workstations is presented in Figure 3.5. It is a generalization of Algorithm SM. Algorithm MM uses list ts, which is initially empty (a list is a vector of variable length). The algorithm uses ts to temporarily store all start values of new EPT realizations. From the time points in ts, capacity of machines is claimed. Besides list ts array s is also used to store start values of EPT realizations (an array is a vector of fixed length). The start values stored in s belong to capacity claims that can be assigned to specific machines. Thus, element $s[i]$ is the start time that capacity of the i-th machine is claimed. From a certain time point when a value for $s[i]$ can be set, its value is taken from list ts. In list ts all start values remain that cannot yet be assigned to a particular machine. This ensures continuous capacity claims. Finally, a third variable is used in Algorithm MM: array nt. Element $nt[i]$ of array nt equals the number of lots present in the workstation that are or will be dispatched into the i-th machine. The sum of $nt[i]$ over all machines equals the number of lots n present in the workstation.

Algorithm MM again triggers on an event and determines besides the actual time τ and the event ev also the machine number i. In case of an arrival event, i is the number of the machine that the arrived lot will be processed on in the future. In case of a departure, i is the machine the lot was processed on.

At an arrival, Algorithm MM distinguishes four cases by combining two boolean expressions, $n \otimes m$ and $nt[i] \otimes 0$, where $\otimes$ denotes a relational operator:

- $n \otimes m$: If the number of lots is below the number of machines ($n < m$), a new EPT realization has to be started and the start time is added to the rear of list ts (cases (i) and (ii)). If the number of present lots is larger than or equal to the number of machines in the workstation ($n \geq m$), capacity of all machines is claimed already (cases (iii) and (iv)). Therefore, no new EPT realization has to be started.

- $nt[i] \otimes 0$: When a lot arrives for processing on machine i, an EPT start value can only be assigned to this specific machine if no other lot is already waiting for this machine or being processed on this machine (cases (i) and (iii)). So, if $nt[i] = 0$ is true, then $s[i]$ is set to the head value of list ts, and the head of list ts is removed.

At a departure, the opposite holds:

- $n \otimes m$: a new EPT realization has to start directly only if $n \geq m$ to claim the machine that has become available (cases (vii) and (viii)). Then, the corresponding start value (τ) is added to list ts or set in $s[i]$.

```
n := 0; nt := zeros(m); ts := emptylist()
loop
    read τ, ev, i
    if ev = "A" then
        if n < m ∧ nt[i] = 0 then                    (i)
            ts := append(ts, τ)
            s[i] := head(ts); ts := tail(ts)
        elseif n < m ∧ nt[i] > 0 then                (ii)
            ts := append(ts, τ)
        elseif n ≥ m ∧ nt[i] = 0 then                (iii)
            s[i] := head(ts); ts := tail(ts)
        elseif n ≥ m ∧ nt[i] > 0 then                (iv)
            skip
        endif
        n := n + 1; nt[i] := nt[i] + 1
    elseif ev = "D" then
        write τ − s[i]
        n := n − 1; nt[i] := nt[i] − 1
        if n < m ∧ nt[i] = 0 then                    (v)
            skip
        elseif n < m ∧ nt[i] > 0 then                (vi)
            s[i] := head(ts); ts := tail(ts)
        elseif n ≥ m ∧ nt[i] = 0 then                (vii)
            ts := append(ts, τ)
        elseif n ≥ m ∧ nt[i] > 0 then                (viii)
            s[i] := τ
        endif
    endif
endloop
```

Figure 3.5: Algorithm MM – multiple machines.

- $nt[i] \otimes 0$: when a lot departs from machine i, a new EPT start value has to be assigned to this machine, only if other lots are waiting for this machine ($nt[i] > 0$). This new EPT start value $s[i]$ depends on n: if $n < m$, $s[i]$ becomes the head of ts (case (vi)), but if $n \geq m$, all idle machines are already claimed and $s[i]$ can be directly set to the actual time τ leaving ts in its original state (case (viii)).

Summarizing, if an event occurs, the algorithm distinguishes eight different cases as denoted between parenthesis in Figure 3.5. In general, if a new EPT realization has to be started, the corresponding time is added to the rear of list ts. When it becomes possible to assign a new EPT start value to one specific machine, this value is obtained from list ts by taking the oldest element of ts and updating ts. The only exception is case (viii) when all capacity has already been claimed. Then the EPT start value of the corresponding machine is set equal to the current event time τ to express that only for this machine a new EPT realization starts whereas for all the other machines capacity is still continuously claimed.

3.4 Examples

The examples presented in this section are used to validate the EPT definition and to show how the EPT can be interpreted. The first example shows two Gantt charts that illustrate the eight cases of Algorithm MM. The other examples are based on a discrete-event simulation model of a workstation with multiple machines that suffer from one or more different sources of variability. The discrete-event model is implemented using the specification and simulation software χ (Rooda and Vervoort, 2003), and explicitly includes each different source of variability. Running the model gives an estimate of the expected cycle time CT (sum of waiting and processing time) and generates a list of arrival and departure events. Fifty simulation replications of 200,000 lots each are carried out for each experiment in the examples. The EPT realizations are calculated from the list of events using Algorithm MM. This gives estimates for the mean effective process time t_e and coefficient of variation c_e^2. These t_e and c_e^2 values are used to replace the original machine processing specifications in the discrete-event model by an "EPT-based" Gamma distribution. Running this EPT-based meta model gives another estimate of the expected cycle time CT^*. The original CT and meta model CT^* should have a similar behavior to ensure that t_e and c_e^2 are good measures for capacity loss and variability. That is, reduction of process time variability in the original model should be represented by a reduced c_e^2 value with an accordingly decreased CT^* value in the meta model. The examples show that CT and CT^* have approximately equal values if the original discrete-event model generates EPT realizations that can always be assigned to one of the machines. The original model then has an effective process time behavior that corresponds with a $G/G/m$ queueing system. If lots are not always processed on the first available machine, the $G/G/m$ representation by the meta model still appears sufficiently accurate to correctly interpret the t_e and c_e^2 values.

3.4.1 Two Gantt charts

The Gantt charts in Figure 3.6 show two different processing schedules of four lots. The corresponding EPT realizations as computed by Algorithm MM are shown in the gray boxes. In the Gantt chart of Figure 3.6(a) a lot always claims the machine it will be processed on. Algorithm MM can directly assign the EPT to a particular machine implying that only cases (i), (iv), (v), and (viii) occur and that list ts stays empty. In this way, all EPT realizations can be assigned to one of the machines.

In the Gantt chart of Figure 3.6(b) for some reason lots are not processed on the first available machine. These lots start capacity claims on machines they will not be processed on. For example, machine 2 is idle from time $t = 1$ till time $t = 3$. Lot 2 could have been processed by this idle machine, but this does not happen. A similar situation occurs from $t = 7$ till $t = 8$: Lot 4 is available for processing on machine 1, but machine 1 is held idle. This forces Algorithm MM to go through all eight different cases. Cases (ii), (iii), (vi), and (vii) arise whenever capacity of an idle machine is claimed by a queued lot which will however not be processed on this particular machine.

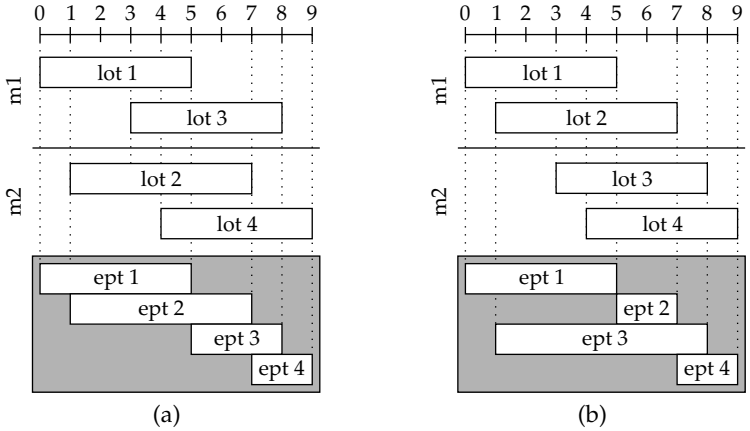

Figure 3.6: Multiple machines.

Table 3.1: Unreliable machines: down time during processing.

t_f/t_r	r_a	t_e	c_e^2	CT	CT^*
0.8/0.2	1.0	1.000	0.330	1.229	1.229
	1.4	1.000	0.330	1.649	1.649
	1.8	1.000	0.330	3.856	3.887
8.0/2.0	1.0	1.000	1.050	1.319	1.341
	1.4	1.000	1.052	1.954	1.988
	1.8	1.000	1.050	5.368	5.381
16.0/4.0	1.0	1.000	1.846	1.401	1.465
	1.4	0.999	1.845	2.261	2.338
	1.8	1.000	1.850	6.930	7.026

3.4.2 Unreliable machines

Consider a workstation with two identical unreliable machines, a single infinite buffer, and Poisson arrival of lots. The mean theoretical process time of the machines is $t_0 = 0.8$ with a coefficient of variation $c_0^2 = 0.25$. However, a machine may break down and be temporarily unavailable for further processing. Two different cases are distinguished: (i) a machine only breaks down during processing, and (ii) a machine may break down at any time – called general down – even if the machine is idle. The down behavior is modeled by exponential failure and repair time distributions, with a mean time between failure t_f and a mean time to repair t_r. It is assumed that if a down occurs during processing of a lot, the lot is finished after repair and takes the remaining process time. In the simulation experiments t_f and t_r are varied in such a way that the availability always has a constant value of $A = t_f/(t_f + t_r) = 0.8$.

Table 3.1 shows the simulation results of the workstation with machines that only fail during processing. The EPT realizations that result correspond with the Gantt chart of Figure 3.6(a). The workstation is simulated at different mean time between

Table 3.2: Unreliable machines: general down time.

t_f/t_r	r_a	t_e	c_e^2	CT	CT^*
0.8/0.2	1.0	1.013	0.326	1.251	1.252
	1.4	1.008	0.327	1.675	1.685
	1.8	1.003	0.329	3.895	3.957
8.0/2.0	1.0	1.063	1.073	1.485	1.494
	1.4	1.038	1.079	2.158	2.247
	1.8	1.013	1.065	5.583	6.183
16.0/4.0	1.0	1.082	1.920	1.689	1.724
	1.4	1.038	1.945	2.624	2.836
	1.8	1.014	1.895	7.365	8.560

failure t_f and mean time to repair t_r levels and at various throughput levels (arrival rates r_a). The three selected t_f and t_r combinations give a low, moderate, and high variability workstation. The arrival rates of 1.0, 1.4, and 1.8 lots per time unit yield workstation utilizations of 0.5, 0.7, and 0.9, respectively. The t_e and c_e^2 values are obtained from Algorithm MM. For each throughput level the mean effective process time is estimated to be 1.0. This is correct since $t_e = t_0/A$. The squared coefficient of variation c_e^2 increases for increasing values of t_f and t_r. This reflects the previously mentioned statement of Hopp and Spearman (2001) that, for equal availability, machines with frequent but short outages are to be preferred to machines with infrequent but long outages. The estimated mean cycle time of the original model, CT, and the cycle time of the EPT based meta model, CT^*, are approximately equal for all throughput levels and t_f and t_r values. The inaccuracies of the computed mean cycle time values for the highest utilization levels are at most 1.5% based on a 95% confidence interval.

Table 3.2 shows the simulation results of the second case with general down time machines. Now a schedule such as in Figure 3.6(b) may arise. The workstation is simulated at the same t_f, t_r, and r_a levels as before. The difference with the first case is that the mean effective process time is not constant anymore as function of the throughput level. For small throughput levels the mean effective process time becomes larger than 1.0. This is caused by the effect that all the machines in an empty workstation can be down when a new lot enters the queue. Then this lot cannot start processing immediately (Adan and Resing, 2001). The c_e^2 values are not constant either. The effective process time behavior of the original workstation model does not resemble a $G/G/m$ queueing system anymore, and, as a consequence, the mean cycle time time of the original model, CT, shows larger deviations from the cycle time of the meta model, CT^*. The meta model approximation is however accurate enough to correctly interpret t_e and c_e^2 using $G/G/m$ queueing relation (3.1).

3.4.3 Unequal machines

In many practical cases, machines in a workstation are not completely identical and may differ in the mean or variance of the process times. Therefore consider a work-

Table 3.3: Unequal machines: a fast and a slow machine.

$t_0(1)/t_0(2)$	r_a	t_e	c_e^2	CT	CT^*
0.9/1.125	1.0	1.004	0.266	1.222	1.227
	1.4	1.002	0.266	1.616	1.628
	1.8	1.001	0.266	3.711	3.705
0.75/1.5	1.0	1.038	0.404	1.262	1.314
	1.4	1.020	0.406	1.642	1.775
	1.8	1.006	0.406	3.689	4.245
0.60/3.0	1.0	1.171	1.202	1.424	1.840
	1.4	1.084	1.236	1.747	2.724
	1.8	1.024	1.250	3.797	7.434

Table 3.4: Unequal machines: machines with difference in variability of the process times.

$c_0^2(1)/c_0^2(2)$	r_a	t_e	c_e^2	CT	CT^*
0.25/1.00	1.0	1.001	0.620	1.271	1.275
	1.4	1.000	0.620	1.780	1.785
	1.8	1.000	0.623	4.435	4.441
1.00/2.00	1.0	1.000	1.493	1.403	1.408
	1.4	1.001	1.494	2.188	2.193
	1.8	1.000	1.500	6.221	6.277

station with two unequal machines. Both machines have Gamma distributed process times with means $t_0(1)$ and $t_0(2)$, and squared coefficients of variation $c_0^2(1)$ and $c_0^2(2)$. Two cases are studied: (i) the machines have equal coefficients of variation $c_0^2(1) = c_0^2(2) = 0.25$, but different mean process times, and (ii) the machines have equal mean process times $t_0(1) = t_0(2) = 1.0$ but different coefficients of variation. The FIFO dispatch rule that is used in the simulation study does not account for the difference in capacity or variability. If the two machines are both idle, they have an equal probability to start processing a lot.

The first case considers a fast and a slow machine in parallel. Different values are assigned to $t_0(1)$ and $t_0(2)$, but the mean capacity of the machines is kept at 2.0 lots/hour. Table 3.3 shows the results of three different settings for the process time of the fast machine $t_0(1)$ and the slow machine $t_0(2)$. For increasing difference in capacity of the machines, the variability of the effective process time of the workstation increases. However, the increase of cycle time CT is much less compared with the increase of c_e^2. The reason is that EPT realizations are measured for the workstation as a unit without distinction between the machines. The difference in process times is included as variability and causes c_e^2 to increase, while for a correct interpretation c_e^2 should stay 0.25. As a consequence, the meta model overestimates the effect of unequal capacity on the cycle time. This means that we should be careful with the interpretation of the c_e^2 if there is a large capacity difference in the machines of a workstation. Also a second effect can be observed; for larger capacity differences,

the estimated mean effective process time tends to rise above the 1.0 value for decreasing throughput. This is due to the absence of a suitable dispatching rule in the model to account for the fact that the first machine is faster than the second. For low throughput capacity is actually lost due to unnecessary idle time of the fast machine. This is reflected in an increase of t_e.

The second case considers a workstation with two machines with equal capacity rates set at 1.0 lot/hour, but with different variability in the process times. Table 3.4 shows the results for two settings of the variability, machines with low and moderate variability, and machines with moderate and high variability. The estimated c_e^2 values equal the averages of the two squared coefficients of variability of the process time of the machines. For both the original and the EPT based meta model the estimated cycle times are approximately equal. The c_e^2 correctly represents the variability in the workstation.

3.5 Case study

A case study from Philips Semiconductors is used to illustrate the potential of the EPT as a performance measure for cycle time reduction. The Philips wafer fabrication facility is a multi-product multi-process fab with more than 400 machines. Over 1.5 million track-in and track-out events were extracted from the Manufacturing Execution System (MES), covering a period of half a year in 1998. A track-in is the start of a process step of a lot on a machine. The track-out is the end of this process step. The track-out data was converted to arrival and departure events, assuming that a track-out of a lot implies an arrival at the next process step. This includes transport time due to material handling in the effective process time realizations. The data was filtered and checked for inconsistency, e.g. the track-in of a lot should always have a corresponding track-out. Using this data, the mean effective process time t_e and the squared coefficient of variation c_e^2 were computed for fourteen workstations each consisting of single-lot machines that process just one lot at at time.

The computed t_e and c_e^2 values are presented in Figure 3.7(a) and 3.7(b) and compared with the nominal process time t_0 and its variation c_0^2. The nominal process time only includes the time a lot has been actually in process. If the t_e of a workstation is larger than t_0 this means that a considerable amount of capacity is lost due to irregularities. Similarly, the c_e^2 can be compared with c_0^2, revealing how much additional variability is present. For most workstations the natural variability c_0^2 is very small due to the highly automated and controlled processes in semiconductor manufacturing. Figure 3.7(b) shows that there is one workstation, workstation N, with a c_0^2 larger than 1.0. This workstation performs process capability measurements. Measurement time depends on the maturity of the process flow and can vary between 4 hours and 40 hours. For all other workstations, the c_e^2 is significantly larger than c_0^2, showing how much variability is due to irregularities such as down time, setups, and operator availability.

Figures 3.7(c) and 3.7(d) show respectively the arrival coefficients of variation and the utilizations of these workstations, respectively. Figure 3.7(e) shows the cycle time factors of the workstations as retrieved from the MES represented by the solid

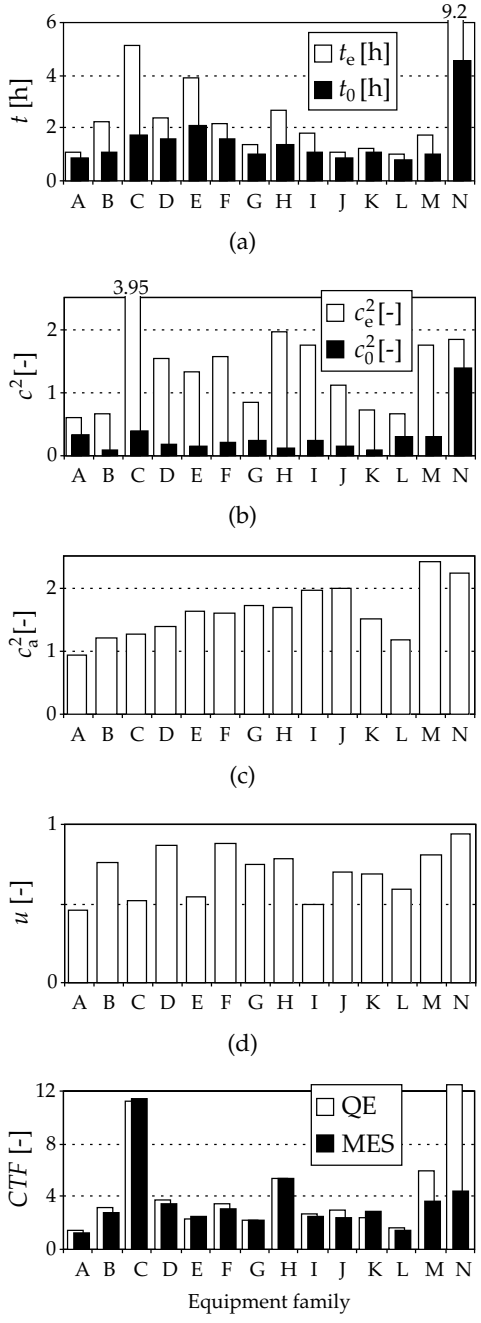

Figure 3.7: Case study MOS4YOU. (a) Mean process times. (b) Process time coefficients of variation. (c) Arrival coefficients of variation. (d) Utilization. (e) Cycle time factors.

bars in the figure. The cycle time factor is defined as the quotient of cycle time and nominal process time. It is an often used indicator in the semiconductor industry to measure the contribution of waiting time to the cycle time. From Figure 3.7(e) we can observe that workstation C, H, and N have a cycle time factor, CTF, larger than 4.0, indicating a very bad cycle time performance. Figure 3.7(a), (b), (c), and (d) can be interpreted using (3.1) and give more detailed information about this bad performance. In Figure 3.7(e) the cycle time factor estimations using (3.1) based on t_e, c_e^2, c_a^2, and u are represented by the white bars. On the basis of the data presented in the figures, (3.1) yields for most of the workstations cycle time estimations that lie within 15% of the real mean cycle time values observed in the fab (obtained from the MES). This confirms that the interpretation using (3.1) is indeed valid.

Workstation C has a t_e value almost three times as large as t_0, and a large c_e^2 of almost 4.0. The arrival coefficient of variation has a reasonable value of $c_a^2 = 1.3$. The combination of a high t_e and c_e^2 value causes the large cycle time, even though the utilization is only 50%! So the problem is not shortage of capacity, but capacity loss and variability due to irregularities. Closer examination reveals that workstation C consists of a single machine that is not used very frequently. Lots are queued until the queue length has reached a considerable length. Then all queued lots are processed in succession. This way of processing causes a substantial amount of variability in the operational times, and the c_e^2 rises. The problem of workstation H mainly lies with a combination of a significant capacity loss ($t_e/t_0 \approx 2$) and a high c_e^2 value. This is caused by the long down periods that occurred at this workstation during the period of data collection. In addition, workstation H has a highly variable arrival flow of lots ($c_a^2 \approx 2$). Finally, workstation N combines a very high t_e with a utilization near one. We already mentioned that workstation N is an exceptional station where the time of measurement is highly variable and the process time may be shortened in busy periods. This implies that the process times and queue length are correlated. As a result, the CTF observed in practice is significantly lower than predicted through (3.1) using the estimated t_e and c_e^2 values.

Variability can be reduced in a number of ways. Examination of the production processes has to reveal the possible causes for irregularities. These causes may differ for each of the workstations. In general, the capacity loss and process time variability due to irregularities has to be minimized. One always has to try to arrive at short and frequent irregularities instead of long and infrequent irregularities to minimize variability in the effective process times. For example, one long period of maintenance should be divided in more and shorter periods of maintenance. Furthermore, lots should be processed in a steady working pace, and should be processed at an idle machine without accumulation.

3.6 Conclusion

A new algorithm is proposed that enables estimation of the mean effective process time and the corresponding squared coefficient of variation from manufacturing execution system data. The required data includes arrival and departure times of lots at the workstations, and corresponding identification numbers of the machines on

which the lots have been processed. For our case study we were able to obtain this information from track-in and track-out data of the machines.

With this algorithm available, all four key parameters determining the cycle time of a workstation can be measured: mean effective process time t_e, squared coefficient of variation of the effective process time c_e^2, the squared arrival coefficient of variation c_a^2, and the utilization u (see (3.1) and (3.2)). The case study shows how, using these four parameters, the main causes of large cycle times can be identified, and appropriate actions defined. Without estimates for t_e and c_e^2 this reasoning is not possible, and one can only rely on a cycle time based measure for each machine, such as the cycle time factor, which however does not give a clue on the true cause of any large cycle time observed.

The examples show that the EPT algorithm correctly computes the mean effective process time t_e and squared coefficient of variation c_e^2 if the workstation yields EPT realizations that can always be assigned to one particular machine in the workstation. The mean cycle time of the EPT-based discrete-event meta model corresponds with the cycle time observed at the workstation. The $G/G/m$ cycle time queueing equation (3.1) can be used as an explicit relation to interpret t_e and c_e^2. In many practical cases, lots are not always processed on the first available machine. Upon arrival these lots have to start capacity claims on other machines than they will be processed on to represent the loss of capacity. The proposed algorithm accounts for this. The EPT-based meta model then becomes less accurate. As a consequence, the computed t_e and c_e^2 values are difficult to validate with respect to the estimated cycle times. The workstation example with general down times shows, however, that the $G/G/m$ representation is still sufficiently accurate to correctly interpret the t_e and c_e^2 values.

In this paper the EPT algorithm has been specifically developed for single-lot machine workstations: only machines that process one lot at a time are considered. It is furthermore assumed that the machines in a workstation do not have large differences in processing times since this troubles the interpretation of the c_e^2. The single-lot assumption is a serious restriction: many machines in semiconductor manufacturing process more than one lot at a time, such as lithography machines or furnaces. A generalization of the EPT algorithm to include these types of machines is being investigated. For the moment, EPT promises to be a very powerful tool in cycle time monitoring and improvement for semiconductor manufacturing.

Chapter 4

Quantifying Variability of Batching Equipment

Process time variability is one of the main elements that determine the cycle time of a lot. Several sources of variability can be distinguished, for example machine breakdowns, setup times, and operator availability. However, identification and measurement of all individual sources is almost impossible. Therefore, in previous work a new method has been developed to measure Effective Process Times (EPT) for single-lot machines. The EPT includes the various sources of variability. Typical parameters of a workstation that can be computed from the measured EPT realizations are the mean effective process time and the corresponding squared coefficient of variation. With actual values of the mean and variation coefficient of the EPT available, a useful tool for cycle time reduction programs arises. In this paper the EPT quantification approach is further generalized towards batching equipment, which are commonly present in semiconductor industry. The proposed method adds a new transformation algorithm that enables one to use the previously developed single-lot algorithm also for batch machine workstations. Discrete-event simulation examples are used to validate the generalized EPT approach. A case study from semiconductor industry is used to illustrate its application on an operational data set of furnace workstations.

4.1 Introduction

Semiconductor wafer fabs are complex manufacturing systems containing various types of equipment in which numerous sources of variability occur. These sources of variability have a negative influence on the cycle time performance. Variability

Reproduced from: Van Bakel, P. P., Jacobs, J. H., Etman, L. F. P., and Rooda, J. E.: Quantifying Variability of Batching Equipment using Effective Process Times, *IEEE Transactions on Semiconductor Manufacturing*, submitted.

is caused by stochastic process times and process disturbances such as machine failures, maintenance, setup times, and operator availability.

Given a certain installed production capacity, increasing the throughput of the fab causes a non-linear increase of the mean cycle time. To improve the compromise between throughput and cycle time, variability must be reduced. The effect of variability on the cycle time performance can be described by means of the effective process time concept, as introduced by Hopp and Spearman (2001). They define the Effective Process Time (EPT) as a single 'replacement' process time distribution for a workstation, in which all process disturbances are included. A similar idea is proposed by Sattler (1996). The mean effective process time, denoted by t_e, relates to the effective capacity of a machine. The squared coefficient of variation of effective process time c_e^2, which equals variance σ_e^2 divided by t_e^2, corresponds to the effective variability. These two parameters can be interpreted using a $G/G/m$ queueing equation to approximate the mean cycle time φ, for instance (Hopp and Spearman, 2001):

$$\varphi = \frac{c_a^2 + c_e^2}{2} \cdot \frac{u^{(\sqrt{2(m+1)}-1)}}{m(1-u)} \cdot t_e + t_e \tag{4.1}$$

Equation (4.1) clearly identifies the contribution of utilization and variability to the mean cycle time. The first term contains the squared coefficients of variation of inter arrival times c_a^2 and effective process times c_e^2, respectively. The second term contains the utilization u, which is defined as the quotient of the mean arrival rate r_a and the effective installed capacity:

$$u = \frac{r_a t_e}{m} \tag{4.2}$$

where m represents the number of machines in the workstation.

In practice, it is difficult to calculate workstation EPT parameters t_e and c_e^2 by means of analytical relations such as presented in Hopp and Spearman (2001). These relations require that all contributing disturbances are known and have been quantified. Sturm et al. (1999) observed that it is almost impossible to measure each individual source of variability. Furthermore, the analytical relations are subject to rather restrictive assumptions, which may not always be valid in practical situations. To circumvent these difficulties, in Chapter 3 a new method has been proposed to compute the EPT parameters from actual fab data on arrival and departure events of lots at the workstation. This method does not require the identification of the individual sources of variability. The effective process time definition is formalized by computing so-called EPT-realizations from the arrival and departure events: each departing lot results in a single EPT realization. These EPT realizations yield an EPT distribution, from which parameters t_e and c_e^2 are computed.

Several types of equipment can be found within a semiconductor manufacturing environment. For instance, we can distinguish single-lot equipment, i.e. machines that process a single-lot at a time, batching equipment, machines processing multiple lots simultaneously, and conveyor-like machines, e.g. track-scanner lithography equipment. The effective process time algorithm as proposed in Chapter 3 has been

specifically developed for workstations containing single-lot machines. The objective of this paper is to generalize the EPT quantification method to include batching equipment, such as furnaces.

The paper is organized as follows. First, the single-lot method to measure effective process times as proposed in Chapter 3 is revisited in Section 4.2. We distinguish between workstations obeying the so-called non-idling assumption and workstations violating this assumption from an effective-process-time perspective. This distinction based on the EPT non-idling assumption is then used to make the generalization towards batching equipment in Section 4.3. The new EPT batching algorithm is validated in Section 4.4, by means of discrete-event simulation examples. A case study from Philips Semiconductors is presented in Section 4.5 to illustrate the use of the EPT batching algorithm. Section 4.6 concludes the work.

4.2 EPT calculation for single-lot machines

Hopp and Spearman (2001) describe the effective process time as "the time seen by lots from a logistical point of view". EPT therefore includes all time losses due to any source of variability. This EPT concept is similar to a description given by Sattler (1996): "all cycle time except waiting for an other lot".

In Chapter 3 a new method has been proposed to actually measure the effective process time. They use data from the fab, containing arrival and departure events of lots at a workstation. This data is generally available in highly automated industries such as the semiconductor industry. They consider a single-lot workstation, which contains one or more machines that perform a similar operation and that share a single queue with an unlimited storage capacity. The arrival of a lot in the queue and the departure of a lot from a machine are taken as input data to determine EPT realizations. An EPT realization is defined as "the time a lot was in process plus the time a lot, not necessarily the same one, could have been in process on one of the machines" (see Chapter 3). Thus, each lot delivers an EPT realization. All together, these realizations form an EPT distribution from which mean effective process time t_e and squared coefficient of variation c_e^2 are computed.

In the present paper we show that the single-lot EPT method of Chapter 3 can also be used to compute EPT realizations of batching equipment. The extension towards batching starts from the observation that for multi-machine workstations satisfying the so-called EPT non-idling assumption, each EPT realization can be designated to one specific lot and to one specific machine in the workstation. This is explained below.

4.2.1 EPT non-idling

Consider a single-lot machine workstation. Equation (4.1) can be used for interpretation of EPT parameters t_e and c_e^2. This equation describes how several key parameters of a workstation affect the mean cycle time of a lot. The equation is based on a set of assumptions, one of these is the non-idling assumption (Buzacott and Shanthikumar, 1993). This assumption states that a machine cannot be held idle while a

time	lot	event
0	1	Arrival
1	2	A
2	1	Departure
4	2	D

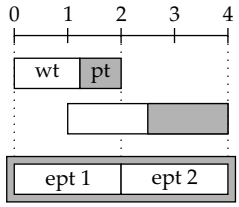

Figure 4.1: One single-lot machine: EPT non-idling schedule.

lot is waiting to be processed. Since (4.1) is expressed in terms of EPT parameters t_e and c_e^2, the non-idling assumption has to be obeyed in terms of the effective process time. This means that the EPT realizations have to obey the non-idling assumption. Note that this is a less strict assumption than the assumption that the workstation itself may not be kept idle while a lot is queued.

Single machine

Figure 4.1 presents an example of a list of arrival events in the buffer ("A") and departure events from the machine ("D"), which occur at event time τ in a single machine workstation. These events are depicted in a Gantt chart. In the Gantt chart, details are drawn regarding the actual process time (shaded part, pt) and the waiting time (unshaded part, wt). The EPT realizations that follow from these arrival and departure events according to Chapter 3 are drawn at the bottom of the Gantt chart. Both lots have a certain amount of waiting time before they are processed, while the machine is idle. Thus, the machine itself disobeys the non-idling assumption. However, from an EPT point of view, the workstation does obey the non-idling assumption: for both lots holds that a new EPT capacity claim, i.e., an EPT realization, is started (i) whenever the arriving lot arrives at an empty workstation or (ii) the departing lot leaves a non-empty workstation.

If the workstation obeys the EPT non-idling assumption, the definition of an EPT realization can be written as: the time a lot was in process plus the time *this* lot could have been in process. In accordance with Chapter 3, this definition is formalized by algorithm 1SLM-n in Figure 4.2.

In algorithm 1SLM-n, initially, the number of lots in the workstation n equals 0 ($n := 0$). The algorithm reads events containing the event time τ and event value ev. The algorithm starts an EPT realization after the arrival of a lot ("A") in case of an empty workstation (i), or after the departure of a lot ("D") if the workstation still contains at least one lot (iv). This is done by assigning the EPT start time s to the event time τ. An EPT realization ends at the departure of a lot and equals $\tau - s$, which is written as output. Since the workstation contains a single machine, only one lot can claim capacity at a time. Note, that for the EPT calculation the start times of actual processing are not needed. Therefore, the actual time of processing is not plotted anymore in the Gantt charts that will follow.

```
n := 0
loop
    read τ, ev
    if ev = "A" then
        if n = 0 then s := τ                (i)
        elseif n > 0 then skip              (ii)
        endif
        n := n + 1
    elseif ev = "D" then
        write τ − s
        n := n − 1
        if n = 0 then skip                  (iii)
        elseif n > 0 then s := τ            (iv)
        endif
    endif
endloop
```

Figure 4.2: EPT algorithm 1SLM-n: one single-lot machine, EPT non-idling.

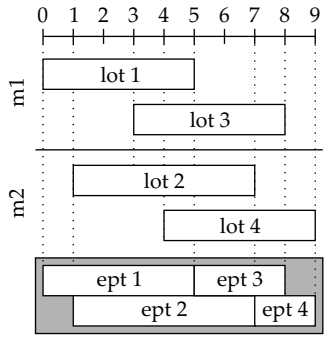

Figure 4.3: Multiple single-lot machines: EPT non-idling schedule.

Multiple machines

Consider a multiple machine workstation that still satisfies the EPT non-idling assumption. In Figure 4.3 such a workstation is illustrated by an example of a small schedule of four lots processed by a two-machine workstation. From an EPT point of view, the machines are never held idle while lots are waiting to be processed. The workstation obeys the EPT non-idling assumption: after a lot departure, the next lot waiting in the queue immediately starts a new capacity claim on the machine that has just finished processing. The lot that starts the new claim will also be processed on this particular machine.

Similar to the single machine case, for a multiple machine workstation obeying the EPT non-idling assumption holds that an EPT realization equals the process time of a lot plus the time *this* lot could have been in process. The corresponding EPT realization can be assigned to the machine the lot has been processed on, which

```
n := zeros(m)
loop
    read τ, ev, i
    if ev = "A" then
        if nt[i] = 0 then st[i] := τ          (i)
        elseif nt[i] > 0 then skip            (ii)
        endif
        nt[i] := nt[i] + 1
    elseif ev = "D" then
        write τ − st[i]
        nt[i] := nt[i] − 1
        if nt[i] = 0 then skip                (iii)
        elseif nt[i] > 0 then st[i] := τ      (vi)
        endif
    endif
endloop
```

Figure 4.4: EPT algorithm mSLM-n: multiple single-lot machines, EPT non-idling.

implies that, in the EPT non-idling case, EPT realizations can be determined for each machine individually. This observation is used to derive the multi-machine EPT non-idling algorithm mSLM-n from the single-machine algorithm presented in Figure 4.2.

Algorithm mSLM-n is shown in Figure 4.4 and applies the single machine algorithm 1SLM-n for each of the m machines. This is done by adding machine number i to the arrival and departure event data: the machine a lot has been or will be processed on. The number of lots is counted per machine using array nt. The EPT start times are also set for each machine individually using array st.

Example

Consider a workstation with two identical unreliable machines, a single infinite First-In-First-Out (FIFO) buffer, and Poisson arrival of lots ($c_a^2 = 1.0$). The machines have a Gamma distributed natural process time with $t_0 = 0.8$ and squared coefficient of variation $c_0^2 = 0.25$. It is assumed that a machine may breakdown only when a lot is in process, which implies that the EPT non-idling assumption is obeyed. The time between breakdowns of a machine is modeled by an exponential distribution, with mean process time between failure t_f. Repair of the machine is also represented by an exponential distribution, with mean time to repair t_r. After repair, the remaining process time of the lot in process is finished. The two machines in the workstation have equal t_f and t_r values, and their distributions are independent of each other.

Three combinations of equal availability $A = t_f/(t_f + t_r) = 0.8$ are selected, corresponding to three levels of variability in the range of frequent and short breakdowns to infrequent and long breakdowns. The following two equations can be used to compute EPT parameters t_e and c_e^2 in this example (Hopp and Spearman, 2001):

$$t_e = \frac{t_0}{A} \tag{4.3}$$

Table 4.1: Two unreliable single-lot machines, EPT non-idling.

	analytical		Alg. mSLM-n		exp.	approx.	
t_f/t_r	t_e	c_e^2	t_e	c_e^2	φ_s	φ_{qe}	φ_m
0.8/0.2	1.000	0.330	0.999	0.330	1.642	1.658	1.650
8.0/2.0	1.000	1.050	0.998	1.037	1.928	1.999	1.960
16.0/4.0	1.000	1.850	0.999	1.838	2.242	2.399	2.338

and

$$c_e^2 = c_0^2 + (1 + c_r^2)A(1 - A)\frac{t_r}{t_0} \tag{4.4}$$

To demonstrate algorithm mSLM-n, discrete-event simulation experiments are performed using an arrival rate of $r_a = 1.4$ (which corresponds to a utilization of $u = 0.7$). For each combination of t_f and t_r, a simulation run of 20,000 lots is repeated with a minimum of 30 replications until the 95% confidence intervals on the estimated values for t_e, c_e^2, and cycle time φ_s are smaller than $\pm 1\%$ of the estimated mean values. Table 4.1 shows that the measured values from algorithm mSLM-n for t_e and c_e^2 correspond with the values analytically obtained from (4.3) and (4.4). The long but infrequent breakdowns yield a high c_e^2 value while the short but frequent outages give a small c_e^2. The t_e is not affected here.

The measured EPT values can be used to estimate mean cycle time φ_s using either queueing approximation (4.1) or a so-called discrete-event meta model representation of the original workstation. The cycle time estimates are also included in Table 4.1: φ_{qe} from the queueing approximation, φ_m from the discrete-event simulation meta model, and φ_s from the original simulation model. The discrete-event meta model has a similar model structure of a buffer and machines, compared to the original discrete-event model. However, the processing of lots, thus the natural process time and breakdowns, is replaced by an EPT-based Gamma distributed process time. Table 4.1 shows that φ_{qe} and φ_m correspond well to φ_s. The small deviations in φ_{qe} are explained by queueing approximation (4.1), which is not exact for an $M/G/2$ workstation (Buzacott and Shanthikumar, 1993). The small deviations in φ_m are explained by the representation of the EPTs by a Gamma distribution in the discrete-event meta model.

4.2.2 EPT general situation

The example in Section 4.2.1 shows that, if the EPT non-idling assumption is obeyed, algorithm mSLM-n determines valid values for t_e and c_e^2. With these values available it is possible to estimate the mean cycle time by means of (4.1) or by an EPT-based discrete-event meta model. However, in practical situations the EPT non-idling assumption may not always be satisfied. This may occur when, for example, the first machine of a two-machine workstation is kept idle while the queued lots are processed by the second machine. According to the definition of an EPT realization, the capacity of the first machine should be claimed by a queued lot. By making this

claim, *the time a lot could have been in process on one of the machines* is added to an EPT realization. However, the lot that should start this claim for the first machine will not be processed by that machine. Therefore, the EPT non-idling assumption is not obeyed.

Recall our definition of an EPT realization being the time a lot was in process plus the time a lot, not necessarily the same one, could have been in process on one of the machines. Thus whenever *a* lot could have been in process, a claim for capacity has to be made and added to an EPT realization. In Chapter 3 is proposed to start EPT realizations in two cases: (i) when new lots arrive in a workstation with less work than installed machine capacity, and (ii) when lots depart from a workstation which has still sufficient work to keep all machines busy. Such a start of an EPT realization is assigned to a particular machine as soon as a lot (not necessarily the same lot that started the claim) is processed on that machine. This idea has been implemented in algorithm MM of Chapter 3. This algorithm starts EPT realizations in a temporary list *ts*, which are assigned to a machine directly or upon later arrival or departure of a lot. However, if the EPT non-idling assumption is obeyed, then this list *ts* is not used and algorithm MM reduces to the EPT non-idling algorithm mSLM-n presented in Figure 4.4.

4.3 EPT calculation for batch machines

In this section the method to measure the effective process time is generalized towards batching equipment. In semiconductor industry several types of equipment can be found with batching characteristics, for example furnaces. A batch machine processes multiple lots in a single batch simultaneously. The number of lots in a single batch is denoted with batch size k. The size of a batch has a certain maximum due to machine limitations: the upper bound k_u. Several loading policies can be followed, for example a fixed-batch policy (batches of equal size) or a variable-batch policy (batch sizes in a certain range $[k_\ell, k_u]$ with k_ℓ the lower bound and k_u the upper bound on the batch size). Chang et al. (1998) use a "greedy" loading policy for a variable-batch machine. This policy implies that a batch is formed as soon as a machine becomes available if the number of queued lots l, not yet assigned to a batch, is at least k_ℓ. As many lots as possible are accumulated in a batch, which results in a batch size equal to:

$$k = \begin{cases} l & \text{if } k_\ell \leq l \leq k_u, \\ k_u & \text{if } l > k_u. \end{cases} \tag{4.5}$$

A batch machine is capable of processing different product classes, i.e. each lot may have its own recipe according to which it has to be processed. The loading policy has to account for these differences in recipes: only lots of the same recipe can be grouped in a batch. In case of multiple recipes, the greedy loading policy groups as many lots of the same recipe as possible in a batch.

To measure effective process times of batch machines, the observation is used that a single-lot machine is in fact a special case of a batch machine. In a batch machine workstation batches of a certain size are queued in the buffer and processed on a

machine. In case of a fixed-batch size of 1, the batch machine workstation equals the single-lot machine workstation. The only difference between the cycle time of a batch machine workstation and the cycle time a single-lot machine workstation is the waiting time to form a batch, for batch sizes larger than 1. The waiting time to form a batch should not be included in the definition of an EPT realization: an EPT realization equals the time a batch was in process plus the time a batch could have been in process on one of the machines. This definition results in an EPT realization for each batch. From these realizations an EPT distribution is obtained, with a mean effective process time t_e and corresponding squared coefficient of variation c_e^2.

The measured EPT batch parameters t_e and c_e^2 can be interpreted using the following $G/G^k/1$ queueing approximation for the mean cycle time of a lot in a single fixed-batch machine, single recipe workstation (Hopp and Spearman, 2001):

$$\varphi = \frac{k-1}{2ku} \cdot t_e + \frac{c_a^2/k + c_e^2}{2} \cdot \frac{u}{1-u} \cdot t_e + t_e \tag{4.6}$$

In this equation the utilization is defined as $u = (r_a \cdot t_e)/k$. The fixed batch size is denoted with k. The first term of (4.6) equals the mean waiting time to form a batch. The remainder of the equation is similar to (4.1) for a single-lot machine, except for the c_a^2/k term. This part describes the waiting time in the queue of the formed batches and the batch processing on the machine, including the revised arrival pattern of batches corresponding to c_a^2/k. Thus, the cycle time of a lot is divided into two parts: waiting time to form a batch, and waiting time and process time of the formed batch.

The forming of batches is modeled as a transformation of the arrival stream of lots into an arrival stream of batches. This concept of transformation is used to measure the effective process time. This paper proposes a transformation algorithm that transforms the arrival events of lots into arrival events of batches. Subsequently, the single-lot EPT algorithm MM of Chapter 3 is used to determine the EPT realizations from the batch arrivals and departures. These realizations build an EPT distribution, from which batch parameters t_e and c_e^2 are computed.

Again, the distinction is made between workstations that obey the batch non-idling assumption and workstations that do not, the general batching situation. Batch non-idling implies that if a batch can be formed, i.e. enough lots of a certain recipe are available, this batch will be processed on the next available idle machine. This is a similar assumption as the EPT non-idling assumption, but now for batches. The batch non-idling assumption implies that if a batch has been formed, a capacity claim starts if at least one machine isn't already claimed. The batch that starts the new claim will also be processed on this particular machine. For the forming of batches the batch non-idling assumption implies that the greedy loading policy is followed, accounting for the difference in recipes. Thus a batch of lots of similar recipe is formed as soon as possible of a size according to (4.5).

In this paper only batch machines are considered which obey the batch non-idling assumption with respect to the forming of batches. This assumption relates to the transformation algorithm. The batch machines though may violate the EPT non-idling assumption regarding the capacity claims by *formed* batches; this is covered by the single-lot EPT algorithm presented in the previous section. Thus, capacity loss

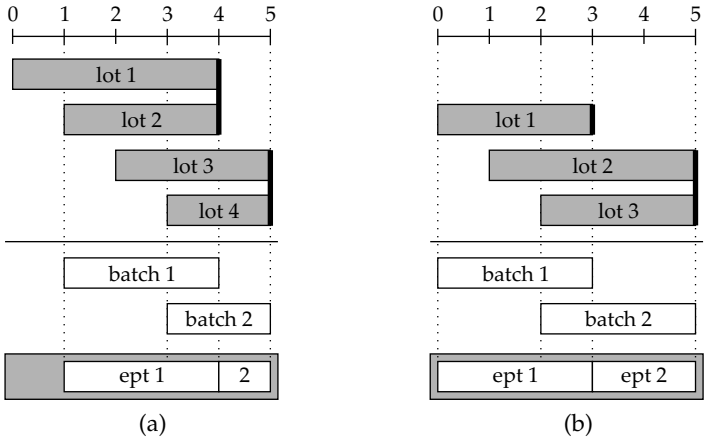

Figure 4.5: One-machine single-recipe schedules. (a) Fixed-batch. (b) Variable-batch.

due to holding machines idle while batches are queued, is included in the EPT by the single-lot algorithm. Capacity loss due to disobeying the greedy loading policy to form the batches is not taken into account by the transformation algorithm and therefore not included in the EPT calculation.

4.3.1 Single machine, single recipe

In this subsection the transformation from lot arrival events into batch arrival events is illustrated and formalized for single-machine single-recipe workstations. Such a workstation is considered to be the most elementary batch machine workstation. First, such a workstation is illustrated using two small schedules. Then the transformation is formalized by a transformation algorithm.

In Figure 4.5 two schedules are depicted: (a) a fixed-batch machine schedule and (b) a variable-batch machine schedule. Both schedules show the arrival events of lots and departure events of batches, as well as the transformed schedule: arrival and departures of batches. The EPT realizations are determined from these batch events.

The events of schedule 4.5(a) are presented in Table 4.2. The transformed events are determined as follows: each time the number of lots l in the queue not yet assigned to a batch equals the fixed batch size k, a batch arrival event is generated. Since there is only one recipe, no distinction has to be made between these lots. Batch departure events in the transformed schedule are equal to the departure events in the original schedule.

The variable-batch machine processes batches of size 1 and 2, as depicted in schedule 4.5(b). The batch arrival events are now generated when the number of unassigned lots l equals the batch size that is processed first on this machine. Thus, first the arrival event of a batch of size 1 is determined, then the arrival event of a batch of size 2, see Table 4.3.

For both schedules it holds that the EPT realizations are determined by single-lot

Table 4.2: List of events for fixed-batch schedule in Figure 4.5

original events				transformed events		
τ	*ev*	*k*	lot(s)	τ	*ev*	batch
0	A		1			
1	A		2	1	A	1
2	A		3			
3	A		4	3	A	2
4	D	2	1, 2	4	D	1
5	D	2	3, 4	5	D	2

Table 4.3: List of events for variable-batch schedule in Figure 4.5.

original events				transformed events		
τ	*ev*	*k*	lot(s)	τ	*ev*	batch
0	A		1	0	A	1
1	A		2			
2	A		3	2	A	2
3	D	1	1	3	D	1
5	D	2	2, 3	5	D	2

```
l := 0
read ks
loop
    read τ, ev
    if ev = "A" then
        l := l + 1
        if l < head(ks) then skip
        elseif l = head(ks) then
            write τ, "A"
            l := 0
            ks := tail(ks)
        endif
    elseif ev = "D" then
        write τ, "D"
    endif
endloop
```

Figure 4.6: Transformation algorithm 1VBM: one batch machine, single recipe.

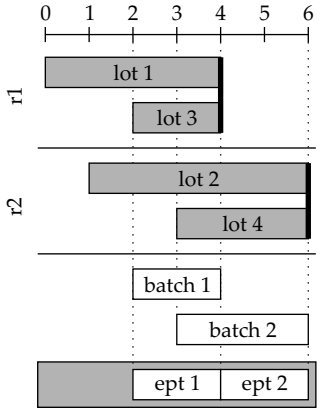

Figure 4.7: One-batch-machine multiple-recipe schedule.

algorithm 1SLM-n, for which the transformed batch events are taken as input. These realizations satisfy the definition that an EPT realization equals the time a batch was in process plus the time this batch could have been in process.

The transformation for a single variable-batch machine workstation is formalized by algorithm 1VBM, as depicted in Figure 4.6. Initially, the number of lots l which are unassigned to a batch is set to 0 and list *ks* is read. List *ks* contains all batch sizes that will be processed on the machine, in time order. For schedule 4.5(b) list *ks* is equal to: $[1, 2]$, which implies that first a batch of size 1 and second a batch of size 2 is processed.

Algorithm 1VBM reads events, which contain event time τ and event value *ev*. After the arrival of a lot, l is raised by 1 ($l := l + 1$). Variable l is compared with the next batch size that will be processed on the machine, which corresponds to the head element of list *ks*. If l is less than the head element of list *ks*, the skip statement is executed: the next batch has not been formed yet. If l equals the head element of list *ks*, a batch arrival event is written, using the current event time τ. After a batch has arrived, the number of unassigned lots is set to 0 and list *ks* is updated: the head element is removed from *ks* by assigning the tail of *ks*. After the departure of a batch, a departure event is written immediately.

4.3.2 Single machine, multiple recipes

The method to measure the effective process time for single-machine single-recipe batching workstations as presented in the previous subsection, is extended to single-machine multiple-recipe workstations. It is assumed that the greedy loading policy accounts for the differences in recipes. Thus, only lots with equal recipes are grouped together in a batch. Therefore, a batch of a certain recipe is formed as soon as enough lots of that recipe have arrived. Using the non-idling property, algorithm 1VBM for single-machine single-recipe workstations can be applied to each recipe separately in single-machine multiple-recipe workstations.

```
lt := zeros(r)
read kst
loop
    read τ, ev, j
    if ev = "A" then
        lt[j] := lt[j] + 1
        if lt[j] < head(kst[j]) then skip
        elseif lt[j] = head(kst[j]) then
            write τ, "A" j
            lt[j] := 0
            kst[j] := tail(kst[j])
        endif
    elseif ev = "D" then
        write τ, "D", j
    endif
endloop
```

Figure 4.8: Transformation algorithm 1VBM-r: one batch machine, multiple recipes.

This is illustrated using a single-machine two-recipes schedule in Figure 4.7. The Gantt chart presents a single-machine schedule where two batches of fixed size 2 are processed. Arriving lots can either be of recipe 0 or recipe 1, and can only be grouped together per recipe. The transformation of lot-based events into batch-based events should account for this difference in recipe: the transformed events have to be determined per recipe separately. The EPT realizations are subsequently determined by applying single-lot algorithm 1SLM to the transformed events.

Applying algorithm 1VBM for each recipe results in algorithm 1VBM-r, presented in Figure 4.8. In algorithm 1VBM-r, array lt stores the number of unassigned lots per recipe, which is initially set to 0 for each recipe. The number of recipes is equal to r. Array kst contains for each recipe a list of batch sizes that will be processed of that particular recipe in the original time order. Index j is used in the arrays to obtain an element corresponding to recipe j. After an event is read, the recipe number j corresponding to the lot arrival or batch departure is known. Then, for this recipe, it is subsequently checked if a batch-arrival event or batch-departure event can be generated.

4.3.3 Multiple machines, multiple recipes

With algorithm 1VBM-r available, the effective process time can be measured for a single-machine multiple-recipe batching workstation. This transformation algorithm determines the time a batch arrives, i.e. the event time that a batch of a certain recipe can be formed. In this subsection the transformation algorithm is generalized towards a multiple-machine multiple-recipe workstation. The new transformation algorithm is called mVBM-r, and is depicted in Figure 4.9.

Multiple machine algorithm mVBM-r is obtained by applying algorithm 1VBM-r (Figure 4.8) to each machine, using the non-idling assumption again. This is simi-

```
ltt := zeros(m, r)
read kstt
loop
    read τ, ev, i, j
    if ev = "A" then
        ltt[i, j] := ltt[i, j] + 1
        if ltt[i, j] < head(kstt[i, j]) then skip
        elseif ltt[i, j] = head(kstt[i, j]) then
            write τ, "A", i, j
            ltt[i, j] := 0
            kstt[i, j] := tail(kstt[i, j])
        endif
    elseif ev = "D" then
        write τ, "D", i, j
    endif
endloop
```

Figure 4.9: Transformation algorithm mVBM-r: multiple batch machines, multiple recipes.

lar as described in Section 4.2: applying single-machine algorithm 1SLM-n to each machine results in multiple-machine algorithm mSLM-n.

To apply algorithm 1VBM-r to each of the m machines in a workstation, the following variables have to be changed or added. Array lt is replaced by matrix ltt, array of lists kst is replaced by matrix of lists $kstt$, and index i is added denoting the corresponding machine. Variable ltt contains the numbers of unassigned lots for each machine, for each recipe. Variable $kstt$ contains the lists of batch sizes for each machine, for each recipe. Elements of these variables are denoted by a double index: i representing the i-th machine and j representing the j-th recipe.

Algorithm mVBM-r requires that variable $kstt$ is known in advance. This means that variable $kstt$ must be obtained from the available event data before the EPT realizations can be computed. Element $kstt[i, j]$ represents the list of batch sizes that will be processed on machine i for recipe j, sorted in time order.

4.4 Simulation study

In this section a simulation study is performed, to validate the proposed algorithms and approach to measure the effective process times of batch machines. A similar discrete-event simulation experiment is performed as in the example of Section 4.2, but now for workstations with batch machines. A discrete-event simulation model of a certain workstation is implemented using the χ language (Rooda and Vervoort, 2003). Simulation runs of 20,000 batches are carried out until the accuracy of the estimated values for t_e, c_e^2, and φ_s has reached a certain level. This level is defined as the 95% confidence interval on the estimated parameter, which has to be smaller than $\pm 1\%$ of the estimated value. Again, the minimum number of repetitions is set to 30.

The modeled workstation contains explicitly different sources of variability, for

Table 4.4: Single machine, single recipe, downs during processing, fixed-batch size $k = 4$.

t_f/t_r	r_a	t_e	c_e^2	φ_s	φ_{qe}	φ_m
0.8/0.2	2.0	1.001	0.331	1.912	2.041	1.910
	2.8	1.000	0.330	2.052	2.211	2.054
	3.6	1.000	0.330	3.847	4.025	3.858
8.0/2.0	2.0	1.002	1.058	2.257	2.407	2.219
	2.8	0.999	1.046	2.879	3.043	2.821
	3.6	1.000	1.048	7.089	7.268	7.075
16.0/4.0	2.0	1.001	1.856	2.657	2.807	2.576
	2.8	1.000	1.843	3.806	3.978	3.730
	3.6	1.000	1.849	10.614	10.872	10.505

example the natural process time and breakdowns. In a simulation experiment mean cycle time φ_s and mean batch size $\overline{k}$ are estimated, and arrival events of lots and departure events of batches are generated. Algorithm mVBM-r (Figure 4.9) is used to transform the events into single-lot events. Subsequently, algorithm MM of Chapter 3 is used to determine the EPT realizations, from which the EPT parameters t_e and c_e^2 are computed. In the examples of this simulation study we assume non-idling, which means that algorithm MM can be replaced by algorithm mSLM-n (Figure 4.4).

After the EPT calculation, a simulation experiment is performed using a so-called meta model. The meta model is obtained by replacing the process time specifications in the original discrete-event model by a Gamma distributed process time with mean t_e and squared coefficient of variation c_e^2, thus an EPT-based process time distribution. From the meta model, response φ_m is determined in a simulation experiment. The meta model value for the mean cycle time is compared with the value from the original model. Ideally, the estimate for φ_m should be comparable to the estimate for φ_s. Also, both values should show the same behavior, according to queueing approximations (4.1) and (4.6). This means that if, for example, the variability in the original model is increased, φ_s and c_e^2 should increase, and thus also the meta model value φ_m should increase. This validation of the EPT parameters t_e and c_e^2 determines if t_e and c_e^2 are accurate measures for the effective capacity and variability in a batch machine workstation.

4.4.1 Single-machine single-recipe workstation

Consider a single batch machine workstation, which has a fixed-batch policy of $k = 4$. All batches are processed according to the same recipe. Breakdowns occur similar to the example in Section 4.2.1: the machine can breakdown during processing, with a mean process time between failure t_f; then a repair is needed, with a mean time to repair t_r. After a repair the remaining process time of a batch is finished. The three combinations for t_f/t_r (from Section 4.2.1) all yield an availability of 80%, but different variability: the breakdowns vary from frequent and short to infrequent and long. Both t_f and t_r are exponentially distributed. Also the natural process time is a source of variability: a Gamma distribution is implemented with $t_0 = 0.8$ and

$c_0^2 = 0.25$.

In Table 4.4 the results are presented. For each variability level three arrival rates: $r_a = 2.0$, 2.8, and 3.6 are considered, resulting in utilizations $u = 0.5$, 0.7, and 0.9, respectively, since t_e equals 1.0. The measured values for t_e and c_e^2 equal those for the single-lot case in Example 4.2.1 in Section 4.2. Again, longer and less frequent breakdowns result in a higher c_e^2. Table 4.4 presents three values for the mean cycle time: the experimental value φ_s and two approximations: analytic approximation φ_{qe} from (4.6) and meta model approximate φ_m. The values for φ_{qe} deviate between 2.4% and 7.8% from the experimental values φ_s. The meta model values φ_m show a smaller deviation: between 0.1% and 3.0%. From this experiment it is concluded that parameters t_e and c_e^2 are accurate measures to represent the effective capacity and variability of the single batch machine, single recipe workstation.

4.4.2 Single-machine multiple-recipes workstation

The influence of multiple recipes is considered for a single batch machine workstation. Recipes are added to one specific experiment from the previous subsection: breakdowns during processing with t_f/t_r equal to $8.0/2.0$ and arrival rate r_a equal to 2.8. Considering the number of recipes, three situations are tested: 2, 3, and 4 recipes. In each case lots arrive in a Poisson arrival stream with arrival rate r_a, and with an equal chance a lot has one of the possible recipes. The natural process time is equal for each recipe, and is Gamma distributed with $t_0 = 0.8$ and $c_0^2 = 0.25$. Thus, the introduction of recipes influences only the forming of batches in this experiment. The number of recipes r are varied for a fixed-batch policy ($k = 4$) and a variable-batch policy ($[k_\ell, k_u] = [2, 4]$). The results are depicted in Table 4.5.

The measured values for EPT parameters t_e and c_e^2 confirm our expectations: the EPT values are the same for each recipe and equal to the single recipe case (Table 4.4, $t_f/t_r = 8.0/2.0$). The batching policy does not affect the parameters t_e and c_e^2, although the batching policy does influence the mean cycle time. The variable-batch policy gives lower mean cycle time values than the fixed batch policy. Furthermore, increasing cycle time is seen due to an increasing number of recipes: more lots are queued before a batch of a certain recipe can be formed. The mean cycle time is accurately estimated by the meta model: compared to φ_s of the original model, the meta model approximate φ_m shows a maximum deviation of 1.4% for the fixed-batch cases and 3.6% for the variable-batch cases. The meta model has the same arrival stream of lots with the same recipes as in the experiment. The greedy loading policy is applied in the meta model.

Next consider the case where each recipe has a completely different natural process time distribution. In Table 4.6 the mean process times t_{0r} for each recipe are depicted, for experiments with 2, 3, and 4 recipes. The squared coefficient of variation c_0^2 is equal to 0.25 for all process times. Similar to the case of "multiple recipes with equal process times", the lots arrive in a Poisson arrival stream with a certain arrival rate r_a, and with an equal chance a lot needs one of the possible recipes. Thus, this distribution is not influenced by the difference in process times.

The measured effective process time parameters t_e and c_e^2 are depicted for each

Table 4.5: Single machine, multiple recipes, equal process times, downs during processing.

k	r	t_e	c_e^2	$\overline{k}$	φ_s	φ_m
[4]	2	1.002	1.053	4.0	3.510	3.472
	3	0.999	1.055	4.0	4.098	4.043
	4	1.000	1.045	4.0	4.684	4.619
[2, 4]	2	0.999	1.048	3.08	3.458	3.366
	3	1.001	1.052	3.04	3.790	3.663
	4	1.001	1.052	3.01	4.116	3.967

Table 4.6: Multiple recipes, unequal process times.

	t_{0A}	t_{0B}	t_{0C}	t_{0D}
$r = 2$	0.5	1.5		
$r = 3$	0.5	1.0	1.5	
$r = 4$	0.5	0.5	0.5	2.5

Table 4.7: Single machine, multiple recipes, unequal process times, downs during processing.

		r_A		r_B		r_C		r_D				
k	r	t_e	c_e^2	t_e	c_e^2	t_e	c_e^2	t_e	c_e^2	$\overline{k}$	φ_s	φ_m
[4]	2	0.626	1.540	1.874	0.676					4.00	7.815	7.719
	3	0.625	1.530	1.250	0.887	1.875	0.675			4.00	8.279	8.217
	4	0.625	1.530	0.626	1.531	0.626	1.535	3.122	0.507	4.00	10.376	10.378
[2,4]	2	0.625	1.529	1.874	0.675					3.63	8.160	7.963
	3	0.626	1.528	1.249	0.890	1.875	0.678			3.61	8.378	8.261
	4	0.625	1.524	0.625	1.530	0.626	1.542	3.124	0.506	3.66	10.688	10.558

recipe in Table 4.7. They all correspond to the analytical values that can be computed using (4.3) and (4.4) with $t_f/t_r = 8.0/2.0$. Again we see that the batching policy has no influence on the computed t_e and c_e^2 values.

Table 4.7 shows the mean batch size $\overline{k}$ and the mean cycle time φ_s measured in the experiments. An increase in cycle time is seen for an increasing number of recipes, and in this particular case the fixed-batch policy results in a slightly lower cycle time than the variable batch policy. The mean cycle time φ_m obtained from the meta model is also presented in Table 4.7. All EPT parameters in Table 4.7 are used in the meta model: each recipe has an EPT-based Gamma distributed process time, and the arrival stream of lots is equal to the arrival stream in the experiments. The results show only small deviations between the mean cycle time from the meta model and the original model: a maximum deviation of 2.4%.

This experiment shows the possibility to create an EPT-based meta model which estimates the mean cycle time accurately for a single batch machine workstation with multiple recipes with unequal process times. In the next subsection the step is made to a multiple machine workstation.

Table 4.8: Multiple machines, multiple recipes, downs during processing, variable-batch size: $[k_\ell, k_u] = [2, 4]$.

	m	r_a	r	$\overline{k}$	φ_s	φ_m
equal	2	5.6	2	3.03	2.115	2.158
process			3	2.99	2.265	2.288
times	3	8.4	2	2.99	1.689	1.737
			3	2.96	1.793	1.839
different	2	5.6	2	3.60	4.609	4.629
process			3	3.59	4.725	4.726
times	3	8.4	2	3.58	3.395	3.468
			3	3.57	3.490	3.518

4.4.3 Multiple machines, multiple-recipes

The single-machine multiple-recipes experiments with variable-batch policy $[k_\ell, k_u] = [2, 4]$ and number of recipes $r = 2$ and $r = 3$ are repeated for a workstation with two and three identical, parallel batch machines. Again, in case of equal process times, each recipe has a Gamma distributed process time with $t_0 = 0.8$ and $c_0^2 = 0.25$. For the cases with unequal process times, Table 4.6 gives the mean natural process time for each recipe, each with $c_0^2 = 0.25$. The arrival rate of the Poisson arrival stream is raised to 5.6 and 8.4 for the 2 and 3 machine workstation, respectively. The exponential failure and repair behavior is now implemented for each machine individually, independent of the other machine.

In Table 4.8 the results for the average batch size $\overline{k}$ and the mean cycle time φ_s from the experiments and the meta model approximate φ_m are presented. The values for t_e and c_e^2 for each recipe are omitted in this table, since these are equal to the values in Table 4.7, measured in the single machine experiment. The conclusions are the same for this multiple batch machine workstation experiment as for the single batch machine workstation experiment: parameters t_e and c_e^2 correctly quantify the effective process time and these parameters can be used to estimate the mean cycle time by means of a discrete-event meta model. The mean cycle time estimated by this meta model (φ_m) has a maximum deviation of 2.8% in case of equal process times and 2.2% in case of unequal process times.

4.5 Case

The case study is taken from Philips Semiconductors MOS4YOU wafer fabrication facility. We discuss nine workstations consisting of furnaces, which in the following text and figures are referred to as capitals A through I. The number of furnaces in each of the workstations are: 2, 6, 6, 2, 6, 5, 6, 2, and 1, respectively. The furnaces of MOS4YOU are typically batching equipment. Each furnace has a maximum capacity of 6 lots that can be processed in one batch. Track-in and track-out data were observed for a time period of six months from January 1st, 2000 until July 1st,

2000. This resulted in 3.6 million track-in and track-out events for all workstations. These track-in and track-out events are used for the analysis of the effective process times, natural process times, variation coefficient of arrival times, utilization, and cycle time factors of the batch machines. A track-out of a lot is assumed to be the arrival at the next process step. This includes transport time due to material handling in the effective process time realizations.

For all nine workstations the following parameters are determined based on the track-in and track-out data set: mean cycle time φ, mean natural process time t_0, natural squared coefficient of variation c_0^2, mean effective process time t_e, effective squared coefficient of variation c_e^2, mean inter arrival time t_a, and squared arrival coefficient of variation c_a^2. The effective process time realizations are determined using transformation algorithm mVBM-r and EPT algorithm MM of Chapter 3. Mean batch size $\overline{k}$ is determined as well as the proportion of processed recipes for each of the workstations. Some workstations just process three different recipes, while others more than ten.

Figure 4.10 shows all the results derived from the analysis of the track-in and track-out data. Figure 4.10(a) shows natural process time t_0 and effective process time t_e for each of the workstations. In all cases the effective process times are higher than the natural process times due to capacity losses. The highest ratio t_e/t_0 is observed for workstation F. In Figure 4.10(b) the natural process time squared coefficient of variation c_0^2 and effective process time squared coefficient of variation c_e^2 are shown. The furnaces have a very low c_0^2 value due to the automated production process. The variation in natural process times is caused completely by small variations in process times among the different recipes. The c_e^2 is quite a bit higher, but for almost all workstations c_e^2 is below 0.35 which is still lowly variable. One exception is workstation D. One single EPT realization of 298 hours resulted in a very high c_e^2 for workstation D. If this outlyer is removed from the data set, t_e becomes 7.60 hours and c_e^2 becomes 0.21. This long EPT realization is caused by a long down period of one of the machines, which consumed most of the time of the EPT realization.

Inter arrival time coefficient of variation c_a^2 is very high for all of the furnace workstations as shown in Figure 4.10(c). This is caused by the wet benches which are the main supplier of the furnaces. The wet benches produce lots in batches of size 2, which results in high c_a^2 values for the inter arrival times of the lots. In Figure 4.10(d) the utilization of the furnaces are shown. The utilization of almost all furnaces is between 60% and 80%. Workstations A and D have a utilization close to 90%. The utilization is defined here as the fraction of time the furnace is effectively processing a batch, even if this batch consists of a single lot: $u = t_e/(t_a \cdot \overline{k})$ with $\overline{k}$ the average batch size of production lots processed by the furnace.

The cycle time factor (CTF) for each of the workstations is shown in Figure 4.10(e) as obtained from the MES. The CTF is defined as the quotient of cycle time and natural process time. The CTFs are all moderately low (below 3.0). A simulation meta model is used to approximate the CTFs. The meta model consists of one buffer and given number of identical batch machines each with a maximum capacity of 6 lots. The buffer follows a FIFO dispatching policy and uses a greedy batch loading policy with a minimum batch size of 1. The process times are Gamma distributed

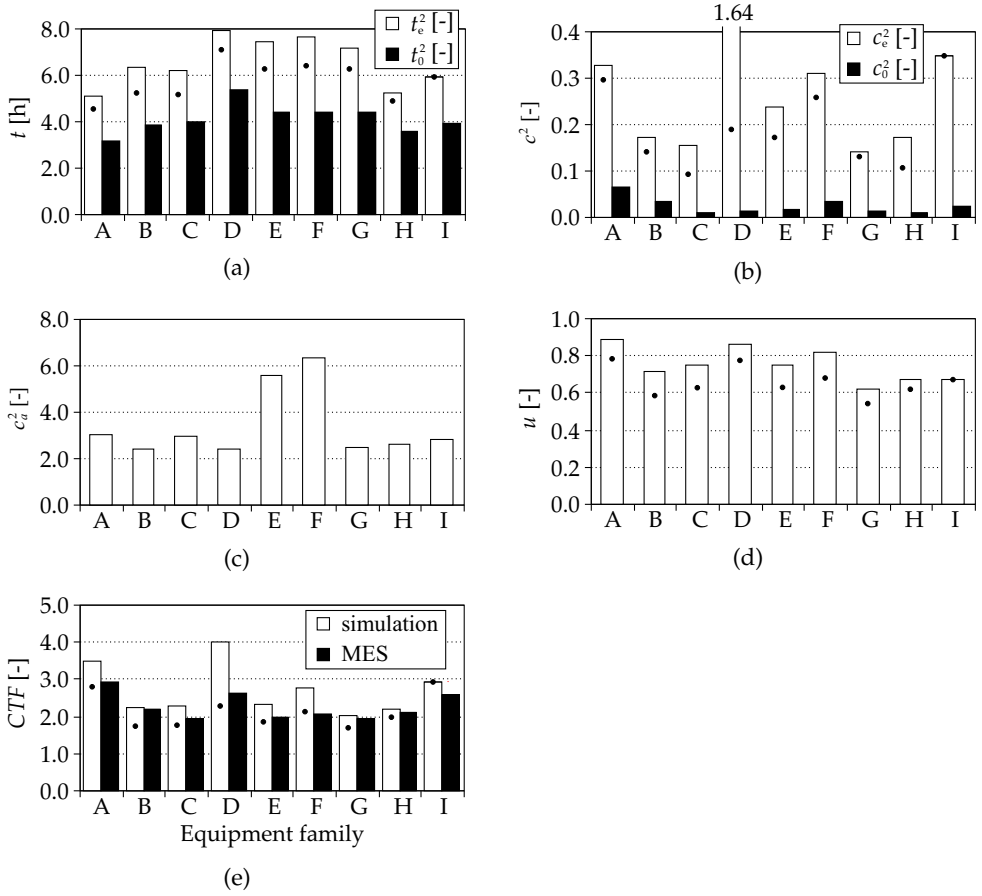

Figure 4.10: Case study MOS4YOU. (a) Mean process times. (b) Process time coefficients of variation. (c) Arrival coefficients of variation. (d) Utilization. (e) Cycle time factors.

with parameters t_e and c_e^2. The lots arrive at the buffer in batches of 2 lots each with the same recipe to be processed. This arrival pattern is according to the measured departure pattern of the wet benches with parameters t_a and c_a^2. The recipe of the arriving lot is determined by a Bernoulli chance equal to the proportion of each of the processed recipes. The mean cycle times in the meta models are determined by running a χ discrete-event simulation model (Rooda and Vervoort, 2003) with 50 runs of 20,000 lots each. The 95% confidence intervals on the estimated mean cycle times are smaller than $\pm 0.2\%$ of the estimated mean values. The simulation meta model mean cycle time estimates are close to the real fab's cycle time. For most workstations the CTF value estimated from the meta model is slightly larger than the CTF values measured from the fab.

Finally, we replaced EPT algorithm MM of Chapter 3 by EPT non-idling algorithm mSLM-n (Figure 4.4) and recalculated the EPT realizations. For each worksta-

tion the new t_e and c_e^2 value is shown in Figure 4.10(a) and 4.10(b) as a small dot in the white bar of the original EPT values. The new t_e values decreased between 7.5 to 17.5%, except for workstation I, as shown in Figure 4.10(a). This implies that some capacity is lost by holding batch machines idle while formed batches are queued. Station I is a single-machine workstation which by definition does not violate the EPT non-idling assumption and therefore gives an equal t_e value. For c_e^2 larger differences were observed for some workstations, see Figure 4.10(b); the 'non-idling' c_e^2-values of workstations C, E, and H are 30% to 40% smaller than the ones computed using algorithm MM. The c_e^2-value of station D decreased to 0.19, which corresponds with the previous observation on the long machine down. EPT algorithm MM corrects for this capacity loss, while algorithm mSLM-n does not account for violations of the EPT non-idling assumption. We also recalculated the corresponding CTF meta model estimates and observed that for most workstations they are below instead of above the measured CTFs from the fab (shown as small dots in the white bars of Figure 4.10(e)). For stations A, C, D, E, and F the CTF estimates based on algorithm mSLM-n are even slightly more accurate than the CTF estimates based on algorithm MM. This suggests that violations of the EPT non-idling assumption (such as long machine downs) should be accounted for if they occur frequently, but the correction proposed in algorithm MM may in some cases overestimate its contribution to the effective capacity and variability.

4.6 Conclusion

An algorithm is proposed for the calculation of effective process time realizations for batching equipment. The required data includes the arrival and departure times of the lots processed at the workstation for the considered time period, the corresponding machine identifications the lots have been processed on, as well as lists of batch sizes sorted in time order for each machine and for each recipe. The EPT batch algorithm is a combination of a new transformation algorithm and the single-lot EPT algorithm. The transformation algorithm is used to transform lot arrivals into 'batch arrivals'. Subsequently, the algorithm of Chapter 3 is used to calculate the EPT realizations for the batches. The transformation starts from the concept of EPT non-idling to calculate the batch arrivals for each individual machine and recipe.

A discrete-event simulation study was setup to validate the proposed EPT quantification approach for batch machine workstations. The simulation experiments obey the EPT non-idling assumption in both batch forming and batch processing. The simulation study showed that the batch effective process times were correctly measured. For the single-machine single-recipe case both the analytical queueing approximation and the discrete-event simulation meta model provided accurate mean cycle time estimations compared to the mean cycle time obtained from the original discrete-event model. Also in the multi-recipe and multi-machine cases accurate cycle time estimations were obtained using the discrete-event meta model both for fixed and variable batch sizes.

The developed batching EPT algorithm has been applied in a case study at the Philips MOS4YOU wafer fab in Nijmegen, The Netherlands. The furnace worksta-

tions were considered. Using the new algorithm, all key parameters determining the cycle time performance could be calculated: the mean effective process time t_e, the corresponding squared coefficient of variation c_e^2, the squared coefficient of variation of lot arrivals c_a^2, and the effective utilization u. The observed variability in the batch processing was low; c_e^2 values lower than 0.35 were found for almost all workstations. For one workstation a larger value was found due to one long down period of the station. The computed workstation t_e values were at most 1.5 times the corresponding raw processing times, which resulted in utilizations between 60% and 90%. The measured c_a^2 values appeared to be relatively high (values of 2 or more). This is due to the batch arrivals from the preceding wet benches. However, the batch forming at the furnaces reduces the effect of these irregular arrivals on the cycle time performance, i.e. c_a^2 is divided by batch size k according to (4.6).

The proposed approach clearly separates the batch forming policy from the effective process times at the workstation. The simulation experiments showed that for fixed and variable batch sizes exactly the same t_e and c_e^2 values are obtained. A bad batching policy can still spoil the cycle performance even if the capacity losses and effective variability are low. For cycle time analysis the batching policy, the workstation utilization, and the variability should be taken into account. From the case study we observed that this rationale could possibly also provide new insights for the single-lot algorithm: replacing EPT algorithm MM of Chapter 3 by EPT non-idling algorithm mSLM-n of Figure 4.4 excludes all violations of the EPT non-idling assumption, for example caused by long machine downs, from the EPT realizations. All capacity losses due to these violations then have to be modeled as being part of the dispatching policy at the workstation.

Chapter 5

Flow Time Approximation Method for Simulation Optimization

This paper proposes a new approximation method to describe product flow times in simulation optimization problems of open queueing networks with fixed routings. Semiconductor wafer fabrication is an example of such a queueing network. The approximations are built from the outcome of simulation runs according to a design of experiments in search subregions. The proposed approximations will be used in a Sequential Approximate Optimization (SAO) approach and provide accurate approximations of flow time performance in each iteration of the optimization process. The idea of the newly developed approximation method is the incorporation of queueing physics of each individual station in the approximation model of the total product flow time. The approximation requires station utilization and variability as response output from the simulation model. These responses are determined using the concept of effective process time. Effective process time was proposed in previous work to quantify utilization and variability of stations in an operating factory. Here, the idea is used in the simulation environment. The newly developed flow time approximation model has been incorporated in a multi-point sequential approximate optimization approach. Test examples considered optimization problems of a four-station flow line and a twelve-station re-entrant flow line, subject to the number of machines as (integer) design variables. For the twelve-station flow line also time to repair and setup time are treated as (continuous) design variables.

5.1 Introduction

This paper presents a new method for simulation-based optimization of discrete-event manufacturing systems regarding the flow time performance. The paper assumes that the manufacturing system can be modeled as an open queueing network

of coupled stations and is represented by a discrete-event simulation model. Each station consists of a number of parallel machines that share a single infinite queue.

Simulation optimization is a means to find optimal settings for parameters in a computationally expensive simulation model. Optimization is performed to find the parameter values that minimize (or maximize) a performance criterion subject to constraints. Simulation optimization is reviewed in journal papers of Fu (1994), Carson and Maria (1997), and Azadivar (1999), and is covered in text books of, e.g., Law and Kelton (2000) and Gosavi (2003). Three major classes of techniques for simulation optimization can be distinguished: (i) gradient based search methods, (ii) Response Surface Methodology (RSM), and (iii) stochastic methods. The gradient based methods determine the response function gradient. This gradient is used to compute a new search direction iteratively. RSM based methods use a sequence of approximation models, that are built in a subdomain of the design space. Stochastic methods, such as genetic algorithms and simulated annealing, combine design exploration with random search strategies.

The proposed method relates to the RSM (see, e.g., Myers and Montgomery, 2002). Basic RSM is a means to obtain an approximation of the objective function for the entire design space. For optimization purposes *sequential* RSM is used to obtain a local approximation in a smaller subdomain of the design space. This local approximation is used to determine the steepest-descent direction. In this direction a line search method is applied to find the optimal design in this direction. This design is starting point for a new local approximation that determines a new steepest-descent direction. Within the discrete-event simulation optimization literature, RSM is usually considered for *unconstrained problems*. Recent work of Angün et al. (2003) introduced RSM for *constrained problems* for simulation-based optimization.

RSM is also commonly applied in structural optimization. See, e.g., Barthelemy and Haftka (1993). Structural optimization considers optimization with a Finite-Element-Method (FEM) model in the loop. In this field, RSM relates to the approximation building of the FEM-based objective and constraint functions. RSM is used in two different ways: (i) to build global approximations of the objective function and constraints, replacing the original optimization problem by an approximate one, and (ii) to generate a sequence of approximate optimization subproblems in smaller subregions. Following the second approach, Toropov et al. (1993, 1996) introduced a multi-point Sequential Approximation Optimization (SAO) strategy. In each iteration, this method builds response surface approximations of objective and constraints using a restricted number of simulation evaluations in a smaller search subregion of the design space. The resulting approximate optimization subproblems are explicit and can be easily solved using a suitable mathematical programming algorithm. A move limit strategy is used to successively adjust the location and size of the search subregions. Multi-point strategies were further developed and successfully applied in work of others (see the overview in Chapter 2).

Abspoel et al. (2001) developed a multi-point SAO technique for simulation optimization of manufacturing systems. The objective and constraint functions were approximated using pure linear approximation models based on D-optimal design of experiments. Integer design variables are considered. Integer design variables

are very typical for manufacturing systems: think of the number of machines, the number of operators, and batch sizes. Abspoel et al. (2001) observed that the non-linearity of the flow time response plays an important role in the approximation. The proposed pure linear approximations are not very well suited in larger search subregions. Due to the integer design variables, the search subregion cannot be reduced beyond the integer restrictions, which can lead to inaccurate approximations in the smallest possible search subregion. Gijsbers (2002) extended this method towards more general linear regression models. These regression models are advantageous in approximating the physics that exists in manufacturing system responses. He proposed a linear regression model for approximating the flow time in which the number of machines per station is treated as a design variable. The regression model terms included one asymptote resembling the station utilization asymptote.

The aim of this paper is to develop a regression approximation model that relates both to utilization and to variability. The regression approximation model of Gijsbers (2002) already included the utilization effects when the number of machines is treated as a design variable, but did not include variability. It furthermore assumes that between stations no correlation exists. The newly developed approximation method defines regression models for utilization and variability for each station separately taking into account design variables of other stations too. The flow time approximation is a pure linear regression model with one extra regression term that is based on both the utilization and variability approximations. The utilization and variability approximations account for the effect design variables may have on the flow time. This allows to include various types of integer and continuous design variables, such as: the number of machines, the number of operators, the mean time to repair, and setup times.

The key feature of the approximation method is the use of Effective Process Time (EPT) in the approximation building. EPT is a measure to quantify capacity and variability in stations of an operating factory (see Chapter 3). Here, the idea is used to determine EPT parameters from the simulation model. The approximations for utilization and variability are built on the computed EPT parameters from simulation evaluations. Since capacity and variability are the main drivers for flow time, the use of effective process time in the flow time approximations is essential here.

This paper is organized as follows. Section 5.2 defines the optimization problem. Section 5.3 discusses the sequential approximate optimization approach. In Section 5.4 the approximation for a single station is discussed. Section 5.5 explains how utilization and variability approximations can be obtained for each separate station in the network using effective process times. Section 5.6 describes how the approximate optimization problem can be built for a single station and shows how these approximations can be combined into a response surface for the complete product flow. Section 5.7 shows which optimization sequence is followed and Section 5.8 discusses some additional implementation issues. The method is tested in Section 5.9 using two optimization examples of a four-station flow line and a twelve-station flow line. This paper concludes in Section 5.10.

5.2 Optimization problem

Consider a manufacturing system that produces discrete products. We assume that the manufacturing system is modeled as an open queueing network of p stations represented by a discrete-event simulation model. Each station, denoted with j, consists of a number of machines, denoted with m_j. All machines within one station share a single queue. The queue has infinite storage capacity, i.e. blocking does not occur. The manufacturing system produces q different product types. Each product flow, denoted with k, has fixed routing and fixed flow rate.

The optimal design problem for a manufacturing system is defined such that some explicit cost function is minimized subject to flow time constraints. The flow time of each product flow is determined by means of a discrete-event simulation model (see Figure 5.1). The stochastics that occur in the discrete-event simulation are represented by variable ω. The flow time estimate $\Phi_k(\mathbf{x}, \omega)$ for each product flow k is a stochastic response.

The optimization problem is defined as follows:

$$\begin{array}{ll} \text{Minimize} & f(\mathbf{x}) \\ \text{subject to:} & E[\Phi_k(\mathbf{x}, \omega)] \leq \Phi_{\mathrm{c},k}, \quad k = 1, \ldots, q \\ & x_{\ell,i} \leq x_i \leq x_{\mathrm{u},i}, \quad i = 1, \ldots, n \end{array} \tag{5.1}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$ is the vector of n design variables, and $x_{\ell,i}$ and $x_{\mathrm{u},i}$ are the lower bounds and upper bounds on the design variables, respectively. Objective function $f(\mathbf{x})$ represents the (explicit) cost function. The constraints relate to the expected value of the mean flow time of each product flow. Flow time $\Phi_k(\mathbf{x}, \omega)$ represents the sample mean flow time of product flow k determined by a single simulation run. The expected value of the mean flow time for all products is estimated by a given number of simulation replications. The expected value of the mean flow time for product k is bounded by $\Phi_{\mathrm{c},k}$.

The mean product flow time $\Phi_k(\mathbf{x}, \omega)$ is the sum of individual station flow times in the simulation model, determined by the product routing. This is illustrated in Figure 5.2 which contains four stations and two different product flows. Denote the flow time of station j by φ_j, with $j = 1, \ldots, p$. The total flow times of the product flows in the example of Figure 5.2 become:

$$\begin{bmatrix} \Phi_1 \\ \Phi_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}}_{R} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \varphi_4 \end{bmatrix} \tag{5.2}$$

or in the general case:

$$\Phi_k(\mathbf{x}) = \sum_{j=1}^{p} R_{kj} \varphi_j(\mathbf{x}) \tag{5.3}$$

where R is the routing matrix. Element R_{kj} contains the number of re-entrant arrivals of product flow k for station j. In this paper it is assumed that R is a given fixed

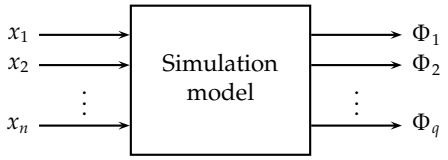

Figure 5.1: Discrete-event simulation model.

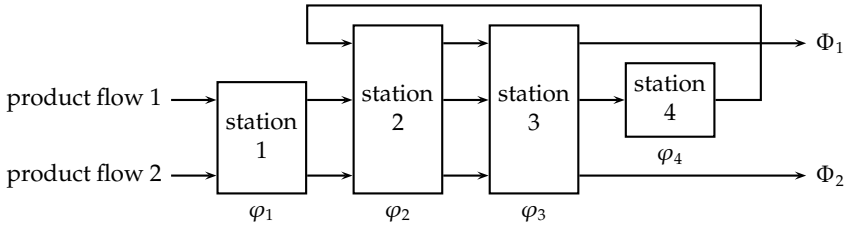

Figure 5.2: Re-entrant flow line with two product flows.

matrix. Following the optimization problem of (5.1), the flow time constraints in Figure 5.2 relate to product flow 1 and product flow 2, bounded by $\Phi_{c,1}$ and $\Phi_{c,2}$, respectively.

The mean flow rate (or throughput) of the manufacturing system is known, since the routing of the products and the flow rate of each product flow are given. For each station j the mean arrival rate $r_{a,j}$ is given by:

$$r_{a,j} = \sum_{k=1}^{q} R_{kj} r_{A,k} \tag{5.4}$$

where $r_{A,k}$ is the mean interarrival rate of product flow k into the queueing network. The interarrival time of product flow k is distributed with mean $t_{A,k} = 1/r_{A,k}$ and Squared Coefficient of Variation (SCV) $c^2_{A,k} = \sigma^2_{A,k}/t^2_{A,k}$ where $\sigma^2_{A,k}$ is the variance of the interarrival times.

Design variables can be continuous or integer. Examples of continuous variables are: arrival distribution parameters $t_{A,k}$ and $c^2_{A,k}$ for each product flow k, process time distribution parameters $t_{e,j}$ and $c^2_{e,j}$ for each station j where $c^2_{e,j}$ is the SCV of the process time distribution. Other continuous design variables could be parameters related to process disturbances such as time to failure and time to repair. Integer design variables may be the number of machines m_j for each station j, or the number of operators. For batching equipment, the batch size may also be an integer design variable. In this paper operational rules included in the simulation model, such as scheduling and dispatching, are assumed to be fixed and are not considered as design variables.

The objective function of the optimization problem is mostly related to cost of investment, like the cost of equipment or the cost of operators. It is assumed that the objective function is explicitly known.

5.3 Optimization approach

For solving the optimization problem of (5.1), a multi-point Sequential Approximate Optimization (SAO) approach is followed. An introduction to SAO techniques can be found in Haftka and Gürdal (1991). The basic principle of an SAO approach is that a sequence of approximate optimization problems is formulated and solved, iteratively. The SAO approach carries out several cycles (or iterations). In each cycle, a number of carefully selected designs is simulated in a search subregion of the design space. Based on the simulation responses, approximations for objective and constraint functions are built. These explicit approximations replace the simulation responses in a so-called approximate optimization problem. This approximate optimization problem is solved within the search subregion by a mathematical optimization solver to find an approximate optimum. This new optimum is the basis for determining a new search subregion in the next cycle. The sequence stops whenever certain stopping criteria related to the improvement during the last few cycles are met.

The simulations that determine flow time $\Phi_k(\mathbf{x}, \omega)$ are computationally expensive. Therefore, within the SAO approach, approximations are needed for these flow time constraint functions to define the approximate optimization problem in each cycle. The approximate optimization problem in cycle (r) is defined as follows:

$$\begin{array}{ll} \text{Minimize} & f(\mathbf{x}) \\ \text{subject to:} & \tilde{\Phi}_k^{(r)}(\mathbf{x}) \leq \Phi_{\mathrm{c},k}, \quad k = 1, \ldots, q \\ & x_{\ell,i} \leq x_{\ell,i}^{(r)} \leq x_i \leq x_{\mathrm{u},i}^{(r)} \leq x_{\mathrm{u},i}, \quad i = 1, \ldots, n \end{array} \tag{5.5}$$

In each cycle, constraint approximation functions $\tilde{\Phi}_k^{(r)}(\mathbf{x}, \omega)$ are used to find an optimum in the search subregion. The constraints $\tilde{\Phi}_k^{(r)}(\mathbf{x})$ are tightened compared to (5.1) to account for the uncertainty from stochastic constraint evaluations. This is further explained in Section 5.6. The rectangular search subregion is represented by lower bounds $x_{\ell,i}^{(r)}$ and upper bounds $x_{\mathrm{u},i}^{(r)}$, also called move limits. The width of the search subregion in design variable direction i is given by: $\Delta_i^{(r)} = x_{\mathrm{u},i}^{(r)} - x_{\ell,i}^{(r)}$. The search subregion is moved and resized at the end of each cycle by changing the values for the move limits.

Each cycle (r) in the SAO approach starts with defining a rectangular search subregion. Rectangular search subregions are placed around iterate $\mathbf{x}_0^{(r)}$. This iterate is the accepted optimal design of the previous cycle $(r-1)$ and is used as starting design in cycle (r). At the start of the SAO sequence the initial iterate $\mathbf{x}_0^{(0)}$ is given. Each iterate $\mathbf{x}_0^{(r)}$ ($r = 0$, 1, 2, etcetera) is simulated M times to evaluate the point regarding feasibility of the flow time constraints. Afterwards, a number of N carefully selected design points is included in the design of experiments in the search subregion. The simulations of the N design points in the experimental design and the M simulation replications in the center point are the basis for the flow time approximation building. The simulation results are used to approximate the flow time constraint functions which build the approximate optimization problem of cycle (r).

This approximate optimization problem is solved by a mathematical optimization solver to find approximate optimal design $\mathbf{x}_*^{(r)}$ in the search subregion. The approximate optimal design is also simulated M times to check objective improvement and feasibility of $\mathbf{x}_*^{(r)}$ compared to $\mathbf{x}_0^{(r)}$ and prior iterates. If the new approximate optimal design indeed shows improvement, this design is selected to be the new iterate of the next cycle.

The check whether a new approximate optimum $\mathbf{x}_*^{(r)}$ is feasible or not is based on M simulation replications. Let α be the significance level of the hypothesis tests to reject a design. Based on the original optimization problem of (5.1), we have one (explicit) objective function and q stochastic flow time constraint functions. For stochastic constraints, the upper bound of the $100(1-2\alpha)\%$ confidence interval on the mean product flow time is used to determine feasibility (Montgomery and Runger, 2003), which replaces the flow time constraints in (5.1) by:

$$\overline{\Phi}_k(\mathbf{x}, M) + t_{\alpha,M-1}\sqrt{\frac{S_k^2(\mathbf{x}, M)}{M}} \leq \Phi_{\mathrm{c},k} \tag{5.6}$$

for all $k = 1, \ldots, q$, where $\overline{\Phi}_k(\mathbf{x}, M)$ is the sample mean and $S_k^2(\mathbf{x}, M)$ the sample variance of the flow time of product flow k based on M replications. In this paper it is assumed that a design is feasible if all constraints are feasible separately, i.e. for all $k = 1, \ldots, q$ holds that $\Phi_k(\mathbf{x}, M) \leq \Phi_{\mathrm{c},k}$. This is further explained in Section 5.6.

5.4 Flow time approximation for single station

For fast convergence and robustness of the SAO method, this paper introduces good quality response surfaces to approximate the flow time constraints. The newly proposed response surfaces are able to approximate the non-linearities present in the flow time responses with high accuracy. The new response surfaces should be able to approximate the flow time constraints for a mix of the continuous and integer design variables mentioned in Section 5.2.

The basis for the new approximation method comes from queueing theory. Several analytical relations of the mean flow time φ are known (see for an overview Buzacott and Shanthikumar, 1993). The mean flow time of a $G/G/m$ station can for instance be given by the following explicit (but not completely exact) expression (Hopp and Spearman, 2001):

$$\varphi = \frac{c_\mathrm{a}^2 + c_\mathrm{e}^2}{2} \cdot \frac{u^{(\sqrt{2(m+1)}-1)}}{m(1-u)} \cdot t_\mathrm{e} + t_\mathrm{e} \tag{5.7}$$

where utilization u is defined as:

$$u = \frac{t_\mathrm{e}}{t_\mathrm{a} m} \tag{5.8}$$

with t_e the mean effective process time, c_e^2 the SCV of the effective process times, t_a the mean interarrival time, c_a^2 the SCV of the arrival times, and m the number of

parallel machines in the station. Note that in (5.7) parameters t_e and c_e^2 represent the mean and SCV of the *effective* process time distribution of the station, respectively. Effective Process Time (EPT) includes process times as well as all irregularities that affect the station capacity.

All quantities in (5.7) may relate to one or more design variables $\mathbf{x}$. The number of machines m can be a design variable. Other design variables, such as mean time to repair, may affect t_e, c_e^2, and c_a^2. This means that if $\mathbf{x}$ changes, parameters t_e, c_e^2, and c_a^2 will also change. The number of machines m is often a design variable itself. Parameter t_a is a given value, since the flow rate for each product flow is fixed. Although (5.7) holds specifically for $G/G/m$ queueing stations, it shows the following main properties, which hold for many other types of stations, e.g. batching stations, as well:

(i) The flow time increases in a non-linear fashion for increasing utilization $u(\mathbf{x})$.

(ii) The flow time rises to infinity for $u(\mathbf{x})$ close to one:

$$\lim_{u(\mathbf{x})\uparrow 1} \varphi(\mathbf{x}) = \infty \tag{5.9}$$

This is referred to as the utilization asymptote.

(iii) The flow time behaves linear with squared coefficient of variation $c^2(\mathbf{x})$. The variability of the station is presented here by $c^2(\mathbf{x})$, which equals the variability term in (5.7):

$$c^2(\mathbf{x}) = \frac{c_a^2(\mathbf{x}) + c_e^2(\mathbf{x})}{2} \tag{5.10}$$

Based on the above mentioned flow time characteristics, we suggest a linear regression model to approximate the flow time of a single station. The new approximation extends the pure linear regression model with one additional linear regression term that contains the queueing physics:

$$\tilde{\varphi}(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \beta_{n+1} \frac{\tilde{c}^2(\mathbf{x})}{1 - \tilde{u}(\mathbf{x})} \tag{5.11}$$

The pure linear model part consists of a constant term β_0, and one linear term $\beta_i x_i$ for each design variable, $i = 1, \ldots, n$. The additional linear regression term

$$\beta_{n+1} \frac{\tilde{c}^2(\mathbf{x})}{1 - \tilde{u}(\mathbf{x})}$$

has to describe the main utilization and variability effects as mentioned above. This new term relates to the MMA type of approximation presented by Svanberg (1987). Main difference is that we will insert expressions approximating the variability as a function of $\mathbf{x}$ and utilization as a function of $\mathbf{x}$. This is further explained in the next section.

5.5 Utilization and variability

The flow time approximation of (5.11) depends on design variables $\mathbf{x}$, utilization $u(\mathbf{x})$, and variability $c^2(\mathbf{x})$:

$$\tilde{\varphi} = \tilde{\varphi}(\mathbf{x}, u(\mathbf{x}), c^2(\mathbf{x}))$$

In order to build the approximations in (5.11), surrogate functions (approximations) for utilization $u(\mathbf{x})$ and variability $c^2(\mathbf{x})$ have to be defined in terms of design variables $\mathbf{x}$, since $u(\mathbf{x})$ and $c^2(\mathbf{x})$ may not be explicitly known or difficult to obtain. In some cases, the utilization and variability can be determined by means of analytical relations as explained in Hopp et al. (2002). However, these analytical expressions for t_e and c_e^2 are based on rather restrictive assumptions, which may not always hold to obtain accurate $u(\mathbf{x})$ and $c^2(\mathbf{x})$ expressions for the stations in the simulation model. If $u(\mathbf{x})$ and $c^2(\mathbf{x})$ cannot be computed analytically, they have to be approximated based on simulation responses from the discrete-event model. An approach to do this is suggested below.

In general, a discrete-event simulation model does not return u and c^2 as a response output. Equations (5.8) and (5.10) have to be used. These equations show that, for a certain station, u and c^2 depend on five quantities. Utilization function $u(\mathbf{x})$ depends on (i) mean effective process time $t_e(\mathbf{x})$, (ii) mean interarrival time $t_a(\mathbf{x})$, and (iii) number of machines m. Variability function $c^2(\mathbf{x})$ depends on (iv) SCV of the interarrival times c_a^2 and (v) SCV of the effective process times c_e^2. These five quantities determine the utilization and variability of the station for a given design $\mathbf{x}$. The next paragraphs of this section discuss how the five quantities can be obtained as a response from discrete-event simulations.

The interarrival time t_a can easily be calculated, since for each product flow the flow rate is given. Given a fixed routing matrix, t_a can be calculated analytically using (5.4). The same holds for the number of machines m, which is a given number for each simulation run (and which is often a design variable itself). The variation of interarrival times c_a can also be obtained easily by sampling the interarrival times during the simulation run (which also provides an estimate for t_a that can be compared with the known analytical value).

The main difficulty lies with t_e and c_e. Values for t_e and c_e cannot be simply measured during the discrete-event simulation. The idea is to use the effective process time approach proposed in Chapter 3 for estimating t_e and c_e during a simulation run. The EPT quantification approach enables to compute t_e and c_e values for stations from arrival and departure data in an operational factory. Here we use this approach within the simulation environment instead of the real-life factory.

Consider the station depicted in Figure 5.3. This figure shows a $G/G/1$ station. Products arrive in the buffer B_∞ with infinite capacity. The arrival times are distributed with mean t_a and coefficient of variation c_a. If products are queued in the buffer, machine M will process them one by one. After being processed the products leave the station.

The Effective Process Time (EPT) of a single product is defined as the time the product was in process plus the time this product *could have been* in process (see Chapter 3). The EPT is in general larger than the raw process time, because the

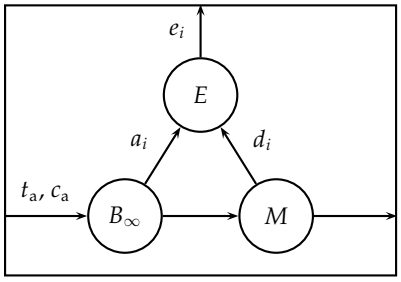

Figure 5.3: Effective process time approach: measuring EPT realizations from a discrete-event manufacturing system.

machine doesn't necessarily start processing immediately when products are available in the queue. This can happen due to process disturbances arising from various sources, such as: machine downs, operator availability, and setups. If for every departing product the individual EPT realization is measured, an EPT distribution emerges for this station. With this EPT distribution available, the sample mean and sample variance can be used to estimate t_e and c_e^2.

In Chapter 3 a method has been introduced that can be used to measure EPT realizations at a station. The basis for the EPT measurements are the actual interarrival and interdeparture times of the products. Assume a station with a single machine that processes one product at a time based on First-Come-First-Serve (FCFS). That is, products arriving at the station, are processed based on FCFS dispatching, and depart from the station without overtaking. The arrival time of the i-th product is denoted by a_i and the departure time of the i-th product is denoted by d_i (see also Figure 5.3). The effective process time e_i of the i-th product can now be calculated by (Kock et al., 2004):

$$e_i = d_i - \max(a_i, d_{i-1}) \tag{5.12}$$

which equals the total time product i was in the station (either waiting in queue or being processed on the machine) after product $i-1$ left the station. For other stations with overtaking of products, the more general EPT algorithm as proposed in Chapter 3 can be used. The idea of the EPT measurements is the same: an external observer at the station, E in Figure 5.3, calculates EPT realizations e_i based on arrival times a_i and departure times d_i.

Using the above described EPT approach in a simulation based setting, response values for u and c^2 for each station can be obtained for each design point evaluated during the SAO sequence. These response values are used to build the flow time approximations in (5.11). This approximation requires functions for utilization and variability, denoted here with: $\tilde{u}(\mathbf{x})$ and $\tilde{c}^2(\mathbf{x})$. For each SAO cycle the utilization and variability approximations are built from the available M simulation replications in $\mathbf{x}_0^{(r)}$ and N additional design of experiments points in the search subregion. So, besides mean flow time response φ_j, the discrete-event simulation model also delivers estimates for t_e, c_e^2, and c_a^2 for each simulation run. Based on these three quantities

and values for t_a and m, the corresponding u and c^2 values can be obtained using (5.8) and (5.10).

Linear regression approximations are built for u and c^2. For a given design of experiments (representing the N design points for which simulation runs are carried out in addition to the M replications available for the current iterate $\mathbf{x}_0^{(r)}$), a response surface model can than be fitted through the utilization and variability data resulting in approximation functions $\tilde{u}(\mathbf{x})$ and $\tilde{c}^2(\mathbf{x})$ which are valid in the corresponding search subregion. Typical response surface models that are often used are linear or polynomial functions. In this paper we approximate the utilization and variability functions by the following linear regression models:

$$\tilde{u}(\mathbf{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n} \alpha_{i+n} \frac{1}{x_i} \tag{5.13}$$

$$\tilde{c}^2(\mathbf{x}) = \gamma_0 + \sum_{i=1}^{n} \gamma_i x_i + \sum_{i=1}^{n} \gamma_{i+n} \frac{1}{x_i} \tag{5.14}$$

Besides the constant regression terms α_0 and γ_0, for each design variable x_i we introduced just one pure linear term ($\alpha_i x_i$ and $\gamma_i x_i$) and one pure reciprocal term (α_{i+n}/x_i and γ_{i+n}/x_i). The use of both linear regression terms is motivated as follows. Capacity related design variables, such as number of machines and number of operators, have a reciprocal effect on utilization, while operational time related design variables, such as setup times and mean time to repair, behave linear with utilization. The same rationale can be used for variability. The proposed linear regression models (5.13) and (5.14) are able to catch both the linear and reciprocal behavior.

5.6 Building the approximate optimization problem

Consider the approximate optimization problem of (5.5). The approximate product flow time $\tilde{\Phi}_k(\mathbf{x})$ is bounded by $\tilde{\Phi}_{c,k}(\mathbf{x})$. The approximate product flow time $\tilde{\Phi}_k(\mathbf{x})$ is a sum of flow times of the individual stations $\tilde{\varphi}_j(\mathbf{x})$ given by (5.3). The flow time approximation of station j is represented by a linear regression model which is based on the approximation functions for utilization $\tilde{u}_j(\mathbf{x})$ and variability $\tilde{c}_j^2(\mathbf{x})$. To construct the approximate optimization problem in each cycle the following procedure is used:

(i) For each station j, with $j = 1, \ldots, p$:

  (a) construct approximation functions for utilization $\tilde{u}_j(\mathbf{x})$ and variability $c_j^2(\mathbf{x})$ introduced in Section 5.5, and

  (b) construct flow time approximation $\tilde{\varphi}_j(\mathbf{x})$ introduced in Section 5.4.

(ii) For each product flow k, with $k = 1, \ldots, q$:

  (c) derive product flow time approximation $\tilde{\Phi}_k(\mathbf{x})$ from the station flow times $\tilde{\varphi}_j(\mathbf{x})$ following (5.3) in Section 5.2, and

(d) determine approximate constraint bound $\tilde{\Phi}_{c,k}$.

The two main steps in this procedure are explained in further detail below.

5.6.1 Workstation flow time approximation

First step (a) is to construct approximation $\tilde{u}(\mathbf{x})$ and $\tilde{c}^2(\mathbf{x})$ for each station in the network (subscript j is omitted here for brevity). Parameters α_i and γ_i ($i = 1, \ldots, 2n$) in (5.13) and (5.14) are determined through linear regression based on the N simulation evaluations of the design of experiments in the search subregion. The M simulation replications at the center point are used to determine (and fix) the constant terms in the regression models (α_0 and γ_0). That is, for $\mathbf{x} = \mathbf{x}_0$ the approximations will equal the mean of the M simulated responses u and c^2 obtained through the procedure described in Section 5.5. For the utilization and variability models this gives for the constant terms:

$$\alpha_0 = \overline{u}(\mathbf{x}_0, M) - \sum_{i=1}^{n} \alpha_i x_{0,i} - \sum_{i=1}^{n} \alpha_{i+n} \frac{1}{x_{0,i}} \tag{5.15}$$

$$\gamma_0 = \overline{c}^2(\mathbf{x}_0, M) - \sum_{i=1}^{n} \gamma_i x_{0,i} - \sum_{i=1}^{n} \gamma_{i+n} \frac{1}{x_{0,i}} \tag{5.16}$$

and for the regression models:

$$\tilde{u}(\mathbf{x}) = \overline{u}(\mathbf{x}_0, M) + \sum_{i=1}^{n} \alpha_i (x_i - x_{0,i}) + \sum_{i=1}^{n} \alpha_{i+n} (\frac{1}{x_i} - \frac{1}{x_{0,i}}) \tag{5.17}$$

$$\tilde{c}^2(\mathbf{x}) = \overline{c}^2(\mathbf{x}_0, M) + \sum_{i=1}^{n} \gamma_i (x_i - x_{0,i}) + \sum_{i=1}^{n} \gamma_{i+n} (\frac{1}{x_i} - \frac{1}{x_{0,i}}) \tag{5.18}$$

Representing in regression native form:

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \boldsymbol{\epsilon} \tag{5.19}$$

where $\mathbf{y}$ is the column of N observations for the utilization or the variability, $\mathbf{b}$ is the column of parameters, and $\boldsymbol{\epsilon}$ the errors. A row $\mathbf{h}(\mathbf{x})$ in design matrix $\mathbf{X}$ can for both utilization and variability be represented by:

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} x_1 - x_{0,1} & x_2 - x_{0,2} & \cdots & x_n - x_{0,n} & \frac{1}{x_1} - \frac{1}{x_{0,1}} & \cdots & \frac{1}{x_n} - \frac{1}{x_{0,n}} \end{bmatrix} \tag{5.20}$$

For the N observations in vector $\mathbf{y}$ holds that each element y_z, $z = 1, \ldots, N$, equals for the utilization

$$y_z = u_z(\mathbf{x}_z) - \overline{u}(\mathbf{x}_0, M)$$

and for the variability

$$y_z = c_z^2(\mathbf{x}_z) - \overline{c}^2(\mathbf{x}_0, M)$$

The unknown parameters can be estimated using least squares:

$$\hat{\mathbf{b}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{5.21}$$

Second step (b) is to build the station flow time approximation model. We start from the model given in (5.11). The regression term

$$\frac{\tilde{c}^2(\mathbf{x})}{1-\tilde{u}(\mathbf{x})} \tag{5.22}$$

is composed of the models $\tilde{u}(\mathbf{x})$ and $\tilde{c}^2(\mathbf{x})$ obtained in previous step (a). To estimate parameters $\boldsymbol{\beta}$ of flow time approximation (5.11), the same procedure is followed. Fixing the constant term such that $\tilde{\varphi}(\mathbf{x}_0)$ equals the mean observed flow time $\overline{\varphi}(\mathbf{x}_0)$ at the center point gives:

$$\beta_0 = \overline{\varphi}(\mathbf{x}_0, M) - \sum_{i=1}^{n} \beta_i x_{0,i} - \beta_{n+1} \frac{\tilde{c}^2(\mathbf{x}_0)}{1-\tilde{u}(\mathbf{x}_0)} \tag{5.23}$$

and for the flow time regression model:

$$\tilde{\varphi}(\mathbf{x}) = \overline{\varphi}(\mathbf{x}_0, M) + \sum_{i=1}^{n} \beta_i (x_i - x_{0,i}) + \beta_{n+1} \left(\frac{\tilde{c}^2(\mathbf{x})}{1-\tilde{u}(\mathbf{x})} - \frac{\tilde{c}^2(\mathbf{x}_0)}{1-\tilde{u}(\mathbf{x}_0)} \right) \tag{5.24}$$

which is again of the form of (5.19). Given N flow time observations in the design of experiments parameters $\boldsymbol{\beta}$ can be estimated using (5.21), with the elements of vector $\mathbf{y}$ equal to $y_z = \varphi(\mathbf{x}_z) - \overline{y}(\mathbf{x}_0, M)$, $z = 1, \ldots, N$. Design matrix $\mathbf{X}$ consists of N rows and each row $\mathbf{h}(\mathbf{x})$ is of the following form:

$$\mathbf{h}(\mathbf{x}) = \left[x_1 - x_{0,1} \quad x_2 - x_{0,2} \quad \cdots \quad x_n - x_{0,n} \quad \frac{c^2(\mathbf{x})}{1-u(\mathbf{x})} - \frac{c^2(\mathbf{x}_0)}{1-u(\mathbf{x}_0)} \right] \tag{5.25}$$

Approximations functions (5.17), (5.18), and (5.24) are regression models for each station in design variables $\mathbf{x}$. In principle, each design variable x_i, $i = 1, \ldots, n$, may have influence on the utilization, variability, and flow time response of each individual station. But in most cases, only a subset of all design variables will have an effect on the response of a particular station. The utilization, variability, and flow time responses of a station will be mainly related to the design variables of the station itself. Design variables of other stations will have a minor effect on the responses of the considered station. For instance, the number of machines of the last station in a flow line will have no influence on the response of the first station in the flow line. On the other hand, the response of a particular station may be influenced by design variables of preceding stations. For instance, the variability of a station depends on the departure pattern of the preceding station. In order to minimize the computational effort needed for the design of experiments, it is assumed in the remainder of this paper that the regression models of each station rely on the design variables related to that particular station.

5.6.2 Product flow time constraint approximation

To approximate the total product flow time $\tilde{\Phi}_k(\mathbf{x})$ of product flow k, all the individual station flow time approximations $\tilde{\varphi}_j(\mathbf{x})$ for each station j have to be combined.

Each station is denoted by $j = 1, \ldots, p$ with p the total number of stations. For each station j, the approximated flow time $\tilde{\varphi}_j(\mathbf{x})$ can be determined separately following Section 5.6.1. This results in parameters $\boldsymbol{\beta}_j$ for the flow time approximation in station j. Each flow time approximation is based on the utilization approximate function $\tilde{u}_j(\mathbf{x})$ and variability approximate function $\tilde{c}_j^2(\mathbf{x})$ with parameters $\boldsymbol{\alpha}_j$ and $\boldsymbol{\gamma}_j$. Using routing matrix R_{kj} of (5.3) and the approximation relation for each station j of (5.11), the approximation of the total flow time of product flow k now becomes:

$$\tilde{\Phi}_{\mathrm{nc},k}(\mathbf{x}) = \sum_{j=1}^{p} R_{kj} \cdot \tilde{\varphi}_j(\mathbf{x}) \tag{5.26}$$

The subscript "nc" (no correction) is added here, since this approximation does not correct for stochastic responses. The constraint correction is described below.

Since the responses are based on stochastic simulations, for each approximate optimization problem, the product flow time constraint bounds have to be tightened due to model based errors and because only a limited number of simulation replications is used to estimate the mean flow time responses. It is assumed that the corrected constraint $\tilde{\Phi}_k(\mathbf{x})$ in the approximate optimization problem of (5.5) is based on the same rationale as used in (5.6). The sample variance of the flow time of product flow k in iterate $\mathbf{x}_0$, $S_k^2(\mathbf{x}_0, M)$, is determined at the start of each cycle based on M replications. Determining the sample variance of the flow time in each of the N points in the experimental design is computationally too expensive. Nevertheless, from queueing theory it is known that the variance of the flow time in a queueing network depends on the mean flow time, i.e. a system that is simulated at a high utilization level, results in a high flow time, but also in a high flow time variance. Similarly, low utilization (low mean flow time) gives rise to low flow time variance.

In the approximate optimization problem we assume that the variance in a design point is proportional with the mean flow time. The following constraint correction is proposed which is an approximation of the upper bound of the $100(1-2\alpha)\%$ confidence interval on the mean response in any particular point $\mathbf{x}$:

$$\tilde{\Phi}_k(\mathbf{x}) = \left[\tilde{\Phi}_{\mathrm{nc},k}(\mathbf{x}) + t_{\alpha,M-1} \sqrt{\frac{S_k^2(\mathbf{x}_0, M)}{M}} \cdot \frac{\tilde{\Phi}_{\mathrm{nc},k}(\mathbf{x})}{\overline{\Phi}_k(\mathbf{x}_0, M)} \right] \cdot w(\mathbf{x}) \tag{5.27}$$

This approximate constraint function $\tilde{\Phi}_k(\mathbf{x})$ is now used in the approximate optimization problem of (5.5). Besides the correction for stochastic responses, an extra correction $w(\mathbf{x})$ is used to account for errors in the fitted linear regression models. This correction is denoted with $w(\mathbf{x})$ and adds more correction for points away from $\mathbf{x}_0$ and less correction for points close to $\mathbf{x}_0$. In $\mathbf{x}_0$ itself, $w(\mathbf{x})$ equals 1.0, which corresponds with no extra correction. The correction $w(\mathbf{x})$ is defined as:

$$w(\mathbf{x}) = 1 + \xi \frac{1}{n} \sum_{i=1}^{n} (x_i - x_{0,i})^2 \tag{5.28}$$

with ξ the user-defined parameter that can be used to add more or less correction for design points far away from $\mathbf{x}_0$. Basically, the correction is proportional with the

squared distance to $\mathbf{x}_0$. This correction function requires that each design variable is appropriately scaled having the same order of magnitude.

5.7 Sequential approximate optimization sequence

The SAO approach is summarized by the sequence presented in Figure 5.4. The approach follows a more or less standard SAO sequence (see Chapter 1). We have extended the sequence such that it can handle non-analyzable or non-valid simulations. A simulation is non-analyzable whenever the pre-defined throughput of a station is above the station capacity, i.e. this station has a utilization above 1.0. Discrete-event simulation of such a non-analyzable design point results in a simulation run in which no stable mean response output is obtained and the average work-in-progress keeps on growing. For the SAO sequence this implies that we need one extra step that checks whether or not all simulations were analyzable in the design of experiments (step 5). If this check fails, the design of experiments has to be revised excluding the non-valid design points and including some new design points. If a design turns out to be non-analyzable in step 8, the evaluated design cannot be accepted as the next optimal design. The sequence now becomes as follows:

0. Define the optimization problem including the design variables $\mathbf{x}$, objective function f, constraint functions $\mathbf{g}$, and routing matrix R. Also define which design variables will be included in the regression models for each station.

1. Set $r := 0$. Evaluate initial design $\mathbf{x}_0$ using M replications of the simulation model. If all simulations are valid, initial design $\mathbf{x}_0$ becomes the initial iterate of the first cycle $\mathbf{x}_0^{(0)}$. It can happen that one, more, or all simulation replications of the initial design seem to be non-analyzable. If this is the case, the sequence stops and the user has to provide a better starting point.

2. Define a rectangular search subregion around iterate $\mathbf{x}_0^{(r)}$ such that $\mathbf{x}_0^{(r)}$ is in the center of the search subregion. If this is not possible, because one of the search subregion bounds violates the corresponding design space bounds, then reposition the search subregion such that the associated boundaries of the search subregion and the design space coincide, while maintaining the size of the search subregion.

3. Determine the Design of Experiments (DoE) in the search subregion. The DoE starts from the M simulation replications at iterate $\mathbf{x}_0^{(r)}$ and adds N additional plan points. The DoE may in principle be of any type as long as $N \geq 2n + 1$ and each included design variable is evaluated at least at three levels (including $\mathbf{x}_0^{(r)}$). The DoE may not include design points that (i) already have been simulated in this cycle and proved to be non-valid in step 5, and (ii) have an approximate utilization above 1.0 for one or more stations based on the utilization approximations $\tilde{u}_j^{(r-1)}(\mathbf{x})$ of the previous cycle (not possible in the first cycle).

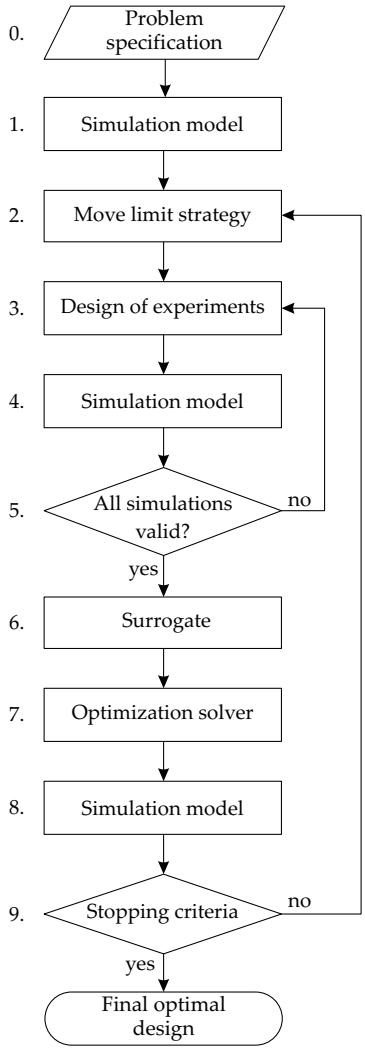

Figure 5.4: Sequence of the SAO approach.

4. Simulate the N additional plan points of the DoE.

5. Check if all simulations of the plan points are valid. If some plan points turned out to be non-analyzable, return to step 3 which defines a new DoE excluding these non-analyzable plan points.

6. Generate p station flow time approximations and combine them into q different product flow time approximations.

7. Solve the approximate optimization problem including the product flow time constraints $\tilde{\Phi}_k(\mathbf{x})$, $k = 1, \ldots, q$, and the explicit objective function.

8. Simulate the approximate optimal design M times. If one or more simulations seem to be non-analyzable, this design will always be rejected in the next step.

9. Check acceptance and stopping criteria. If not converged, update $\mathbf{x}_0^{(r+1)}$, set $r := r + 1$, and return to step 2.

Step 3, 4, and 5 are repeated sequentially, until all plan point simulations in the DoE are valid.

5.8 Implementation

The optimization approach has been implemented in the sequential approximate optimization framework described in Chapter 2. The design of experiments, the solution of the approximate optimization problem, move limit strategy, acceptance and stopping criteria are explained in further detail below.

5.8.1 Design of experiments

As mentioned before, a station flow time approximation may be based on all design variables or on a smaller subset. The latter is advantageous to reduce the required number of simulation evaluations in the experimental design. By taking only the station design variables into account, the design of experiments can be applied for each station separately.

For the station approximation building, we propose a two-level full-factorial design of experiments. Generally, iterate $\mathbf{x}_0$ is the center point. If iterate $\mathbf{x}_0$ is not the center point, an additional center point is added to the experimental design. By these means, each design is evaluated at least at three levels in each design variable direction. This enables to estimate parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ of the utilization and variability regression model, respectively. In the design of experiments, iterate $\mathbf{x}_0$ is replicated M times. The two-level full-factorial design includes 2^n designs points in the experimental design. Here, n equals the number of design variables associated to the particular station. Each of these points is replicated v times. Whenever iterate $\mathbf{x}_0$ is not the center point, this center point is added to the experimental design and also replicated v times. Examples of the experimental design are shown in Figures 5.5(a)

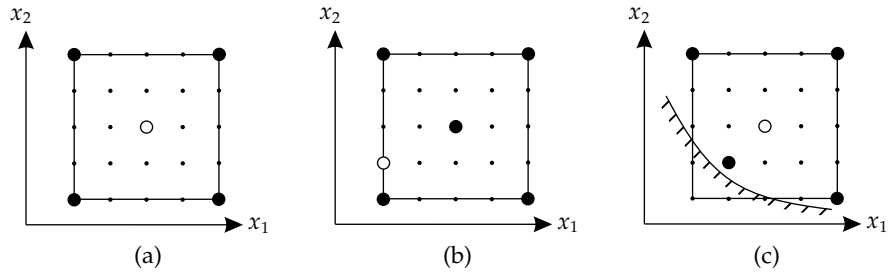

Figure 5.5: Examples of the two-level full-factorial design of experiments: (a) the basic design, (b) design with iterate $\mathbf{x}_0$ on one of the design variable bounds and including one additional design point in the center, and (c) design near the approximate utilization constraint. The small dots represent the candidate points. The filled dots represent the actual experimental design that have been selected. The open dot corresponds with iterate $\mathbf{x}_0$. Criterion (5.29) is used to add the non-corner point in example (c).

and 5.5(b). In the examples presented in Section 5.9 we will only consider the design variables associated to the station itself. But any other design variable may be included as well.

Whenever a design in the experimental design is simulated, the simulation may be non-analyzable and a steady-state solution is not obtained. If this is the case, the non-analyzable design point is removed and a new design point is added to the experimental design, while all analyzable design points remain. The new design point is placed such that the distance between this design point and all design points currently in the experimental design is maximized. This procedure may be repeated in case of multiple non-analyzable design points. Furthermore, to avoid simulation of non-analyzable design points, the utilization approximations $\tilde{u}_j^{(r-1)}$ $(j = 1, \ldots, p)$ of the previous cycle are used to exclude design points on beforehand. A design point may only be added to the experimental design, when the utilization value, determined by the utilization approximation of the previous cycle, in this particular design point is below 1.0.

Whenever a design point has to be repositioned, the following procedure is used. For each design variable direction we define a number of (integer) candidate levels equidistant in the search subregion as shown in Figure 5.5(c) (in the figure five candidate levels are used; in the examples of Section 5.9 we used seven candidate levels). It may be the case that, for integer design variables, the number of integer candidate levels are not possible (due to a smaller search subregion). Then, the method will select all integer candidate levels available in the search subregion (with a minimum of three levels; smaller search subregions are not possible). To maximize the distance between the new candidate point $\mathbf{x}_\mathrm{a}$ and the current design points $\mathbf{x}_{\mathrm{b},z}$ with z the running number of all current design points. To maximize the overall distance between all design points, for each candidate design point the following criterion d

is calculated:

$$d = \sum_{z} \sum_{i=1}^{n} \frac{1}{(x_{\mathrm{a},i} - x_{\mathrm{b},zi})^2} \tag{5.29}$$

The candidate point with the lowest value for criterion d is added to the experimental design. In Figure 5.5(c) such an experimental design is shown. This experimental design is in the neighborhood of an approximate utilization constraint that excludes some points from the basic experimental design.

5.8.2 Solving the approximate optimization problem

In each cycle of the SAO sequence, the approximate optimization problem is defined according to (5.5). The objective and constraint functions in this approximate optimization problem are known analytically and can be solved using a mathematical optimization solver. Since design variables may have integer restrictions, a mixed-integer optimization solver is needed. We used a branch-and-bound algorithm that iteratively calls the SQP algorithm of Lawrence et al. (1997). The branch-and-bound is based on depth-first and creates two branches for each relaxed optimization problem.

The starting point of the optimization solver in each search subregion is set to iterate $\mathbf{x}_0$. The approximation function value for the utilization in this starting point is always smaller than 1.0 and the flow time responses can be predicted. However, it might happen that in some parts of the search subregion, the utilization approximation is equal to or above 1.0. If this is the case, the flow time response cannot be predicted because the denominator of (5.22) is equal to or smaller than 0.0. To avoid the solver getting stuck in this part of the design space, a penalty factor is used. This penalty factor is linear with the utilization constraint violation. The penalty is added to all flow time evaluations with corresponding utilization close to or larger than 1.0, i.e. larger than 1.0-ϵ (we used $\epsilon = 0.05$ in the examples of Section 5.9).

5.8.3 Move limit strategy, acceptance, and stopping criteria

At the end of each cycle, the SAO strategy has a new cycle optimal design $\mathbf{x}_*^{(r)}$ available. A filter method is used to determine if this new cycle optimal design is accepted to be the iterate of the next cycle $\mathbf{x}_0^{(r+1)}$. The filter method was introduced by Fletcher and Leyffer (1998) and was suggested for use in SAO by Brekelmans et al. (2004). Gijsbers (2003) and Vijfvinkel (2004) considered SAO in the context of simulation-optimization, and adapted the filter to account for stochasticity in the simulation runs. The filter idea is that the filter consists of a list of accepted designs. A new cycle optimal design may only be added to this filter if it is not dominated by any other cycle optimal design in the filter. A design is dominated if there exists another design in the filter with both a lower objective value and a lower constraint violation (we consider the maximum constraint violation here). If the new design is accepted and included in the filter, any design in the filter that is dominated by the newly accepted design is removed from the filter. Since in our type of problems

the cycle time constraints are stochastic responses, the maximum constraint violation is determined on the basis of the upper bounds of the $100(1-2\alpha)\%$ confidence interval of the constraints of (5.6).

The stopping criteria are quite straightforward. The SAO sequence is stopped if the search subregion cannot be further reduced when the minimum search subregion size has been reached and the new cycle optimal design has been previously found or has not been accepted. The optimal design now becomes the design in the filter which has the lowest objective value and for which the upper bound of the $100(1-2\alpha)\%$ confidence interval for each constraint is smaller than the corresponding constraint bound.

The move limit strategy defines the position and size of the search subregion at the start of each new cycle. The search subregion is placed such that the new cycle optimal design $x^{(r+1)}$ becomes the center of the new search subregion. If this is not possible, because one of the search subregion bounds violates the design space bounds, the search subregion is repositioned such that the associated boundaries of the search subregion and the design space coincide, while maintaining the size of the search subregion. The search subregion is reduced whenever a design is not accepted in the filter. The reduction factor for each design variable direction is two. If for an integer design variable the width of the search subregion is an odd or non-integer number, the is increased to the nearest even (integer) value. If a design is accepted, the search subregion width is maintained. An exception are those design variables that do not change much, i.e. change less than 25% of the search subregion width. For these design variables the width of the search subregion is reduced by a factor two.

5.9 Test examples

This section presents two examples of optimization problems with a discrete-event simulation model in the loop. The χ discrete-event specification language (Rooda and Vervoort, 2003) is used for the modeling and simulation of the manufacturing systems. Determination of $u(\mathbf{x})$ and $c^2(\mathbf{x})$ is based on the multi-machine EPT algorithm presented in Chapter 3. The second example contains batching machines. For these stations, the batching EPT algorithm of Chapter 4 is used. In the examples the significance level is set at $\alpha = 0.05$.

5.9.1 Four-station flow line

Consider the four-station manufacturing system of Figure 5.6 (Hopp and Spearman, 2001). This flow line produces a single product type with mean product flow time Φ_1. The mean station flow times are denoted with $\varphi_1, \ldots, \varphi_4$. Each station j consists of m_j identical machines. The corresponding data for each station is given in Table 5.1. This table presents the fixed cost FC_j, unit cost UC_j, mean effective process time $t_{e,j}$, and squared CV of effective process time $c^2_{e,j}$ for each station j. Products arrive following a Poisson process (arrival coefficient of variation $c^2_A = 1.0$) at an arrival rate of $r_A = 2.5$ products/hour.

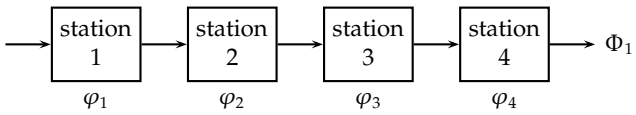

Figure 5.6: Four station flow line with one product flow.

Table 5.1: Data for four-station flow line.

station	fixed cost [k$]	unit cost [k$]	t_e [hrs]	c_e^2 [-]
1	225	100	1.50	1.00
2	150	155	0.78	1.00
3	200	90	1.10	3.14
4	250	130	1.60	0.10

The optimal design problem is defined following (5.1):

$$\begin{array}{ll} \text{Minimize} & f(\mathbf{x}) = \sum_{j=1}^{4} FC_j + \sum_{j=1}^{4} UC_j x_j \\ \text{subject to:} & E[\Phi_1(\mathbf{x}, \omega)] \leq 6.0, \\ & x_1 \geq 4, x_2 \geq 2, x_3 \geq 3, \text{ and } x_4 \geq 5 \end{array} \tag{5.30}$$

The objective is to find the minimum cost solution depending on fixed and unit cost such that the mean product flow time does not exceed 6.0 hours. The design variables relate to the number of machines in each station, vector $\mathbf{x} = [m_1, m_2, m_3, m_4]^T$. The cost $f(\mathbf{x})$ is defined as a function of fixed cost FC_j and unit cost UC_j for each station j.

In order to obtain an analyzable simulation model, the utilization of each station may not exceed one. The utilization of each station can be calculated using (5.8). The Minimum-Cost-Capacity-Feasible solution (MCCF) that satisfies the utilization restriction equals vector $\mathbf{x}_{\text{MCCF}} = [4, 2, 3, 5]^T$ resulting in the following utilization levels for each station: $\mathbf{u}_{\text{MCCF}} = [0.94, 0.98, 0.92, 0.80]^T$. This minimum-cost-capacity-feasible solution is taken as lower bounds on the design variables of the optimization problem.

The number of simulation replications of the center point in each search subregion, i.e. the starting point and each cycle approximate optimum, is evaluated $M = 15$ times. The design of experiments corresponds with a star-design of experiments: each station j has one associated design variable m_j, and each variable is evaluated at a high and a low level. This gives eight points in the experimental design in addition to the center point $\mathbf{x}_0$. The eight points are evaluated twice. The simulation run length of each simulation evaluation equals the production of 65,000 products. The flow time estimation is based on the last 50,000 products which discards the first 15,000 products to overcome the transient initial phase of each simulation.

In the previous work of Abspoel et al. (2001) and Gijsbers (2002) this optimal design problem has also been considered. They used a similar SAO approach, but with

different approximation models. Abspoel et al. (2001) used a pure linear approximation given by:

$$\tilde{\Phi}(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \beta_i x_i$$

They did not manage to find the optimum in 50 independent starts of the optimization approach (starting point was equal to the MCCF solution), but they found 9 different solutions of the optimal solution (most of them neighbor points). Gijsbers (2002) used

$$\tilde{\Phi}(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \beta_i \frac{1}{x_i(x_i - t_{\mathrm{e},i}/t_{\mathrm{a},i})}$$

to approximate the total product flow time and found the true optimum in 90% of the cases of the individual optimization runs. Again, the other reported solutions are neighbor points. Using our newly developed method, we found the true optimal design $\mathbf{x}_* = [6, 3, 5, 6]^{\mathrm{T}}$ for all 50 optimization runs using starting point $\mathbf{x}_0 = [4, 2, 3, 5]^{\mathrm{T}}$ as well as for starting point $\mathbf{x}_0 = [6, 6, 6, 6]^{\mathrm{T}}$. The cost of this design equals 3,120 k$. The initial width of the search subregion was $\Delta_i = 8$ in all design variable directions $i = 1, \ldots, n$. The correction parameter was set at $\xi = 0.0$ in this example. The optimal design was found within 3 to 7 cycles.

5.9.2 Re-entrant flow line

This example is developed based on the twelve-station example as presented by Hopp et al. (2002, Section 4.2). This example considers a re-entrant flow line of twelve stations. Each station consists of a queue and a number of identical parallel machines. Table 5.2 shows the parameters of the machines for each station. The following symbols are used: bs is the batch size, t_{s} is the mean setup time, t_{f} is the mean time between failures, and t_{r} is the mean time to repair. Six stations consist of single-lot machines ($bs = 1$) and six stations consist of batch machines ($bs > 1$). In this example a single-lot machines are referred to as batch machines with a batch size of one. It is assumed that the batch machines may only process full batches with equal recipes for all the products in the batch.

This example considers two product flows: product A and product B. Table 5.3 shows the process recipes for each product type. Flow A has 20 steps and flow B has 24 steps. Flow A is released to the line in batches of two lots with mean time between batch arrivals $t_{\mathrm{A}} = 60$ minutes and flow B is released in batches of five-lots with mean time between batch arrivals $t_{\mathrm{A}} = 120$ minutes. Both arrival streams of batches are Poisson, and thus $c_{\mathrm{A}}^2 = 1.0$.

Only full batches are allowed for processing. Each machine in the station processes batches of the same batch size. A batch may only be accumulated if enough lots are queued from the same type (A or B) and are in the same process step (same recipe). It is not allowed to mix different sort of recipes in a process batch. The process times are deterministic, i.e. the SCV of the natural process time $c_0^2 = 0.0$. If a machine is ready processing a batch, all the products of this batch are immediately transfered to the next queue in the flow and the machine is ready to process the next

Table 5.2: Machine parameters.

station	process	yield	*bs*	t_s [min]	t_f [hrs]	t_r [hrs]	fixed cost [k$]	unit cost [k$]
1	preclean	1.00	1	15	200	2.0	100	130
2	laser	1.00	1	30	200	1.5	100	220
3	alignment	0.99	1	10	50	5.0	100	70
4	clean	1.00	5	0	200	5.0	100	110
5	photo	1.00	1	25	200	8.0	100	350
6	etch	0.98	5	5	200	8.0	100	30
7	strip	1.00	5	15	150	2.0	100	80
8	oxide	1.00	8	10	100	2.0	100	70
9	mask	1.00	5	45	250	2.0	100	65
10	nitride	0.97	8	0	200	2.0	100	55
11	poly	1.00	1	0	100	4.0	100	45
12	probe	0.94	1	0	200	1.0	100	20

batch. Possibly, a product may not be processed correctly. The yield of each process is also shown in Table 5.2. In the simulation model, when a batch finishes processing, for each product in the batch it is determined, using a Bernoulli chance distribution, whether it may proceed in the flow (and the product is immediately transported to the next queue), or has to leave the flow immediately.

A setup is required only if a machine starts processing a batch with a different recipe as the previous batch. If two batches with equal recipes are processed on one machine, a setup is not needed at the start of the processing of the second batch. This is referred to as 'recipe-dedication'. If a machine is not dedicated for the next recipe to be processed, the setup may start whenever all the products of the next batch are waiting in the queue. If multiple batches are waiting in the queue, the next available machine that is ready to process a batch, selects a batch for which it has already been dedicated. If multiple batches are waiting in the queue, and multiple machines are available to start processing, each machine selects a batch for which it already has been dedicated. If still batches remain in the queue and a machine is available, but the machine is dedicated for none of the batches in the queue, this machine performs a setup and is dedicated to the first available batch in the queue (FCFS). After the setup, the batch is processed by the machine. Each setup has a mean setup time t_s and is exponentially distributed, i.e. SCV of setup times $c_s^2 = 1.0$.

The failure of machines is modeled as follows. A machine can only fail during processing of a batch. The mean time to failure is denoted with t_f. The SCV of the time to failure equals $c_f^2 = 1.0$, i.e. time to failure is exponentially distributed. Since the machine only fails during processing, the time to failure is actually a *processing* time to failure. The simulation model accumulates all the time that the machine was in process. If this total processing time equals the time to failure, the machine fails during the processing of a batch. The machine is repaired with a mean repair time of t_r and corresponding $c_r^2 = 1.0$ (exponentially again). After the repair time, the remaining processing time is needed to finish the batch that is still in process.

Table 5.3: Process recipes.

	Product A		Product B	
step	station	t_0 [min]	station	t_0 [min]
1	1	10.0	1	12.0
2	2	20.0	2	22.0
3	3	25.0	3	30.0
4	4	20.0	4	20.0
5	5	15.0	5	12.0
6	6	10.0	6	8.0
7	7	5.0	7	5.0
8	8	25.0	8	20.0
9	9	15.0	9	20.0
10	5	25.0	5	20.0
11	6	15.0	6	10.0
12	7	10.0	7	15.0
13	4	25.0	4	20.0
14	10	20.0	10	15.0
15	9	15.0	9	15.0
16	5	15.0	5	15.0
17	6	20.0	6	25.0
18	7	5.0	6	5.0
19	11	20.0	4	25.0
20	12	45.0	9	10.0
21			5	20.0
22			6	15.0
23			7	5.0
24			12	40.0

The utilization of a station is calculated using:

$$u = \frac{t_\mathrm{e}}{t_\mathrm{a} \cdot m \cdot bs}$$

with t_e the mean effective process time of *batches* which refers to the mean time per batch that the machines were busy with: processing, setups, and repairing.

Two optimization problems are considered. The first optimization problem treats the number of machines of all 12 stations as design variables. The second optimization problem focuses on three stations and optimizes regarding the number of machines, mean time to repair, and mean setup time.

The first optimization problem is defined as follows:

$$\begin{aligned} \text{Minimize} \quad & f(\mathbf{x}) = \sum_{j=1}^{12} FC_j + \sum_{j=1}^{12} UC_j x_j \\ \text{subject to:} \quad & E[\Phi_1(\mathbf{x}, \omega)] \leq 24.0, \\ & E[\Phi_2(\mathbf{x}, \omega)] \leq 24.0, \text{ and} \\ & x_i \geq 1, \quad i = 1, \ldots, 12 \end{aligned} \tag{5.31}$$

Table 5.4: Final optimal designs for three optimization runs with different ξ parameter values.

ξ	x												f [k$]	max ($\mathbf{g}$) [hrs]
0.0000	2	3	5	2	10	4	4	2	10	1	1	9	7,700	23.968
0.0025	2	3	4	2	10	4	5	2	9	1	1	8	7,625	23.954
0.0100	2	3	4	2	10	5	5	1	9	1	1	9	7,605	23.981

with x_i the design variable equal to the number of machines in station i, i.e. $\mathbf{x} = [m_1, \ldots, m_{12}]^{\mathrm{T}}$. The initial design point equals $x_i = 10$ for all stations and the width of the initial search subregion equals $\Delta_i = 8$ in each design variable direction. Each simulation replication equals the production of 150,000 products. The first 15,000 products are discarded to overcome the transient initial phase of each replication.

The optimization history of three different optimization runs are shown in Figure 5.7, 5.8, and 5.9. The figures show the objective value f and the maximum upper bound of the $100(1-\alpha)\%$ confidence interval of the constraints, denoted by $\max(\mathbf{g})$, of each cycle optimal design. Each optimization run had a different value for correction parameter $\xi = 0.0$, 0.0025, and 0.01, respectively, see (5.28). For a correction parameter $\xi = 0.0$ and 0.0025, the optimization yields statistically infeasible solutions during the optimization process. For $\xi = 0.01$, the optimization process shows fast convergence and intermediate feasible solutions. In each figure the minimum p-value is also given, which is a measure for statistical feasibility of a design. Vector $\mathbf{p}$ represents for each constraint the chance that this constraint is satisfied. The value for $\min(\mathbf{p})$ represents the chance that the most dominating constraint is satisfied. Since we use significance level $\alpha = 0.95$, a design with $\min(\mathbf{p}) > 0.95$ corresponds with a feasible design and $\min(\mathbf{p}) < 0.95$ corresponds with an infeasible design. In Figure 5.10 the design variables values and move limit values for each cycle are shown for the optimization run with $\xi = 0.01$. In Table 5.4 the optimal solution $\mathbf{x}_*$ obtained by each of the three optimization runs is summarized. The optimal design $\mathbf{x}_*$ obtained in the optimization run with $\xi = 0.01$ has the lowest objective value. This design results in utilization levels $\mathbf{u} = [0.56, 0.74, 0.60, 0.41, 0.84, 0.17, 0.12, 0.25, 0.07, 0.16, 0.63, 0.32]^{\mathrm{T}}$.

Each cycle required about 24 simulation runs for the design of experiments and 15 simulation replications to determine the feasibility of the cycle optimal design. One simulation run takes about 30 minutes. Simulations were carried out in parallel on a cluster of Linux computers using the Python Batchlib software developed by Hofkamp (2004).

It is difficult to compare our obtained optimal solution with the results obtained by Hopp et al. (2002) since they used an analytical queueing network approximation for their optimization. Our simulation model gives a somewhat different representation. The optimal solution reported by Hopp et al. (2002) appeared to be infeasible when run with our simulation model (mean flow time is about 40 hours and 44 hours for product flow A and B, respectively).

The second optimization problem considers design variables at three stations: 5, 9, and 12. These stations had the largest number of machines in the optimal solution of the first optimization problem. Three types of design variables are used: (i) the

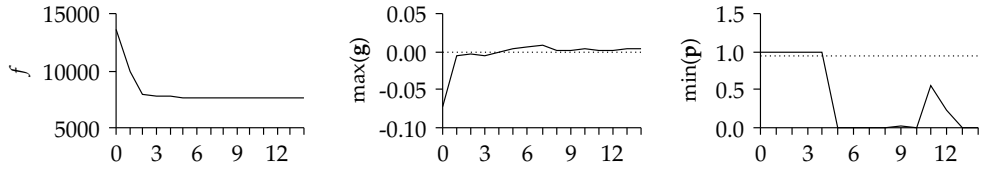

Figure 5.7: Optimization history of objective and constraint violation with $\xi = 0.0$.

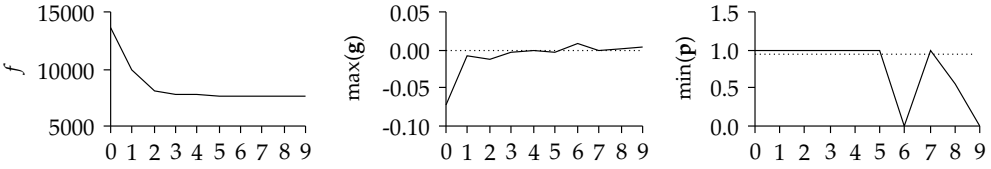

Figure 5.8: Optimization history of objective and constraint violation with $\xi = 0.0025$.

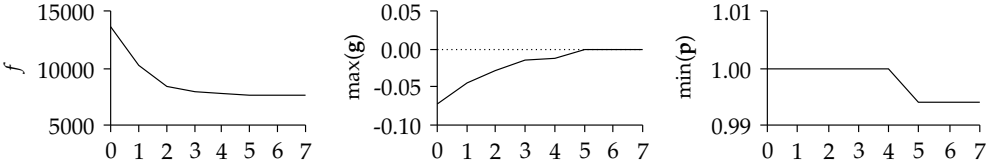

Figure 5.9: Optimization history of objective and constraint violation with $\xi = 0.01$.

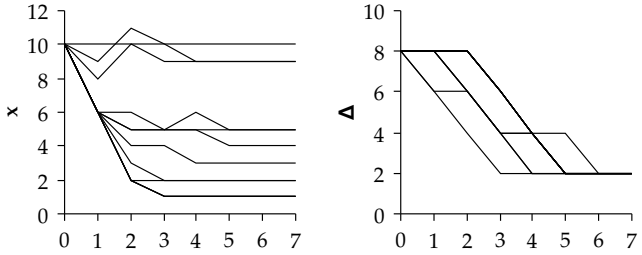

Figure 5.10: Optimization history of design variables and move limit width with $\xi = 0.01$.

number of machines of stations 5, 9, and 12, (ii) the mean setup time of stations 5 and 9 (station 12, probe, does not require setup), and (iii) the mean time to repair of stations 5, 9, and 12. This gives a total of eight design variables: three integer and five continuous. For all other stations, the number of machines was equal to the optimal design reported in Table 5.4, with $\xi = 0.01$. The setup time and time to repair were unchanged for these stations. The total vector of design variables now becomes:

$$\mathbf{x} = [m_5, m_9, m_{12}, t_{\mathrm{s},5}, t_{\mathrm{s},9}, t_{\mathrm{s},5}, t_{\mathrm{s},9}, t_{\mathrm{s},12}]^{\mathrm{T}}$$

The objective function is defined according to (5.31), but with different unit cost for the machines in stations $j = 5, 9$, and 12:

$$UC_j = \left[0.7 + 0.1\frac{t^*_{\mathrm{s},j}}{t_{\mathrm{s},j}} + 0.2\frac{t^*_{\mathrm{r},j}}{t_{\mathrm{r},j}}\right] \cdot UC^*_j \tag{5.32}$$

with UC^*_j, $t^*_{\mathrm{r},j}$, and $t^*_{\mathrm{s},j}$ the original cost, the original mean repair time, and original setup time of the machines, respectively. Using (5.32), the mean repair time is accounted for 20% in the cost of each machine and the setup time is accounted for 10% in the cost of the machine. Reducing the mean repair time and reducing the setup time results in an increase of the cost of the machines. Since station 12 (probe) does not require any setup, this is also omitted from the objective function.

The optimization history of an optimization run of the second optimization problem is shown in Figure 5.11 and Figure 5.12. The initial design equals $\mathbf{x}_0^{(0)} = [10, 10, 10, 25.0, 45.0, 300.0, 300.0, 60.0]^{\mathrm{T}}$ and the initial move limit width equals $\boldsymbol{\Delta}^{(0)} = [6, 6, 6, 40.0, 40.0, 400.0, 400.0, 40.0]^{\mathrm{T}}$. The correction parameter equals $\xi = 0.03$. In the correction function $w(\mathbf{x})$ each design variable was scaled according to the initial move limit width. Again, the simulation run length was equal to 150,000 products. The optimization run stopped at cycle 20, which was the given maximum number of cycles. The optimal solution was obtained in cycle 18 and is equal to $\mathbf{x}_* = [8, 2, 8, 12.47, 7.09, 1015.6, 677.8, 84.6]^{\mathrm{T}}$. The objective function $f = 6,362$ k\$ which is significantly lower than the solution found in the first optimization problem. The utilization of stations 5, 9, and 12 is equal to 0.82, 0.37, and 0.35, respectively. For station 9, the utilization increased from 0.07 to 0.37. This is caused by a decrease of setup time $t_{s,9}$ which allowed a decrease of the number of machines m_9.

5.10 Conclusion

The proposed linear regression models result in good quality approximations for the flow time in manufacturing systems. These approximations can be used in a Sequential Approximate Optimization approach (SAO) to solve simulation based optimization problems of manufacturing systems. The focus was on optimization problems with an explicit and deterministic objective function, subject to stochastic flow time constraints. The flow time constraints related to the various product types are determined via simulation. The method can be used for a black-box discrete-event simulation model of a manufacturing system. An open queueing network

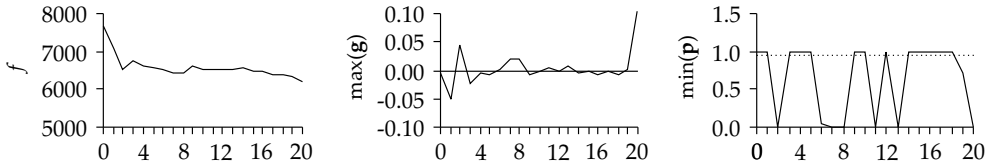

Figure 5.11: Optimization history of objective and constraint violation with $\xi = 0.01$.

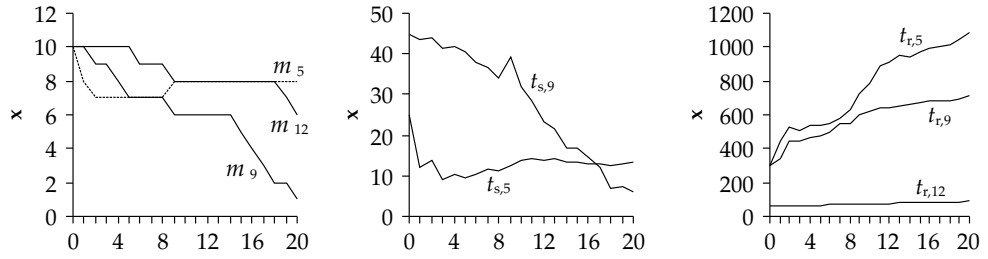

Figure 5.12: Optimization history of design variables with $\xi = 0.03$.

with a fixed routing for each product type is assumed. The design variables may be a mix of continuous and integer parameters. The approximation method needs three simulation responses for each station in the simulation model: mean flow time, utilization, and variability. These responses are used to build approximation models for each individual station.

The approximation model of the mean flow time relies on queueing physics of the individual stations. The main properties that exist in almost all types of stations are related to utilization and variability. The mean flow time of a station depends on the utilization in a non-linear fashion with an asymptote at $u = 1.0$. For a utilization near 1.0, the flow time increases to infinity. The mean flow time depends on variability in a linear fashion. The proposed linear regression model consists of a pure model in the design variables and one additional regression term that accounts for the queueing physics: $c^2(\mathbf{x})/(1 - u(\mathbf{x}))$. For the approximate functions $u(\mathbf{x})$ and $c^2(\mathbf{x})$, linear regression models are suggested. Each station may rely on a separate set of design variables, i.e. not all design variables influence the flow time of a station. For this reason, each station is subject to a separate design of experiments.

Key contribution of the proposed approximation model is that Effective Process Time (EPT) is used to describe the utilization and variability approximation models. EPT is a means to quantify effective capacity and variability of stations. The use of queueing physics with accompanying EPT parameters in the approximations increases the accuracy of the approximations. Even more important is that the EPT-based approach allows to include several types of design variables. All design variables that affect the effective process time at the stations can be included. These types of variables have to be related to capacity or operational time measures and may be integer or continuous.

The flow time approximation models have been implemented in an existing SAO

framework (see Chapter 2). For this purpose we developed a new SAO sequence. The approximate optimization problems are corrected for the stochastic simulation flow time responses. In each new cycle, the variance of the flow time response in the iterate (center point) is known based on M replications. We assumed that the variance of the flow time response is proportional with the mean flow time response within each search subregion. Furthermore, an extra correction is applied to account for errors in the fitted linear regression models. This correction has one extra user parameter to tighten or loosen the constraint function. A tight constraint (high correction) may show slow convergence, but most of the approximate optimal designs are feasible solutions. A loose constraint (low correction) may show faster convergence, but the chance that approximate optimal designs are infeasible, including the final cycle optimal design, is much higher.

The design of experiments has been extended such that our method can deal with non-analyzable regions in the design space. A simulation experiment is called non-analyzable if a steady-state solution is not reached because the required throughput exceeds the capacity of (at least) one of the stations. The plan points in the design of experiments are placed such that the chance that a design appears to be non-analyzable is as small as possible. For this purpose the utilization approximations for each station are used that were built in the previous cycle. The utilization approximations give an estimate whether or not intended plan points in the new design of experiments will be non-analyzable. This is highly advantageous, especially regarding design variables for which the contribution to the utilization is not known beforehand.

The method has been tested on two simulation based optimization applications: (i) a four-station flow line example, and (ii) a twelve-station re-entrant flow line example. For the four-station flow line example, the proposed method was able to find the optimal solution in 100% of all individual optimization runs, which improved the results presented in previous work of Abspoel et al. (2001) and Gijsbers (2002). For the twelve-station flow line, two optimization problems have been applied. The first optimization problem treated the number of machines in each station as a design variable (all integer design variables). The second optimization problem considered the number of machines, mean time to repair, and setup time of three selected stations (mixed integer-continuous design variables). In the first optimization problem, different settings for the constraint correction parameter ξ were used. By increasing this parameter the convergence rate was still sufficient and all cycle optimal designs were feasible. In the second optimization problem not all cycle optimal designs were feasible, but showed that decreasing mean setup times may lead to a more optimal design with lower cost that still satisfies the flow time constraints.

Chapter 6

Conclusions and Recommendations

This thesis focuses on three objectives as mentioned in the introduction. In the next three sections for each objective the main conclusions are drawn and recommendations for further research are given.

Summarizing, this thesis contributes by proposing:

- a sequential approximate optimization framework for engineering optimization problems with a simulation model in the loop,
- definitions for effective process time to quantify variability, and
- a linear regression function that can be used to approximate flow time performance in manufacturing systems.

These three research objectives followed from the two main research questions posed in the introduction. The answer to the first question is yes: approximate optimization concepts that have proved to be successful in the structural optimization field can indeed be employed for simulation optimization of discrete-event manufacturing systems regarding flow time performance. The second question can only be partially answered at this stage: variability can indeed be quantified in a single performance measure on the basis of shop floor data that relates to factory physics without identifying the individual contributing disturbances. Building simulation meta models of complete networks using the developed EPT approach is recommended for further research.

6.1 Framework for sequential approximate optimization

A framework has been developed to provide an open and flexible environment for the development and implementation of Sequential Approximate Optimization (SAO) strategies. The framework consists of three basic layers: (i) the optimization

problem layer, (ii) the SAO sequence layer, and (iii) the numerical routines layer. The framework starts from a collection of optimization sequences and a collection of numerical routines. A typical SAO sequence consists of a number of steps that are carried out iteratively. Each step is represented by one numerical routine. The numerical routines are available from the numerical routines layer and are grouped into a number of modules. Each module consists of routines that perform the same computational task in an SAO sequence. Modules are available regarding, e.g. design of experiments, approximation models, and move limit strategies. The module-based structure of the framework facilitates easy exchange of numerical routines in an SAO sequence and furthermore allows one to (re-) design the SAO strategy for the application at hand.

The framework has been implemented in the object-oriented programming language Python. The optimization problem is specified in the problem layer using Python constructs and pre-defined objects available in the framework. Additionally, Python code representing the SAO sequence has to be provided. One may use 'built-in' sequences or build a new one using predefined classes from the framework library. For both the problem layer and the sequence layer Python is used as the specification language, and one may take advantage of all the functionality of this language. The numerical routines can be provided in Python or in any other (compiled) code, such as Matlab or C++. The interfaces to the various computational languages can be created easily within Python.

The framework has been successfully used so far in various application domains. In particular, optimization problems from the ADOPT-project in the field of structural optimization, optimization of dynamic mechanical systems, and discrete-event simulation-based optimization have been considered. One optimization example is the design optimization of a ten-bar truss structure under deterministic loading conditions and under bounded-but-unknown loading conditions. The framework has been applied likewise to two-bar and seventy-two-bar truss examples. Furthermore, the framework has been applied on optimization problems regarding micro-electromechanical systems (Gurav et al., 2004) and design for robustness and reliability (Van Rooij, 2004).

The use of the framework in other application domains is promising. The basic optimization data objects have been implemented, such as variables, values, designs, and points. These data objects were sufficient to describe the optimization problems and optimization strategies of the above mentioned application domains. For other application domains it may be necessary to extend the framework with new objects. Since the framework is object-oriented, these new objects may (re-) use existing functionality of existing objects. In reliability-based optimization, for instance, specific distributions may be needed to properly describe the reliability measures. These new distribution functions can be included by defining new classes that are based on existing classes.

The framework has been specifically designed for implementing SAO strategies. An SAO strategy can be referred to as a single-point approximation method or a multi-point approximation method. For single-point approximation methods function evaluation and gradient information is used to build approximations in each

search subregion. Multi-point approximation methods are often based on function evaluations alone, but gradient information may be included as well. The evaluation of a design point requires evaluation of a computationally expensive simulation model. We have had good experience with adding new SAO sequences to the framework. In Chapter 2 a more or less standard SAO sequence has been presented. This sequence has been extended towards anti-optimization schemes that require nested optimizations for each design point evaluation. In Chapter 5, the standard SAO sequence is extended to deal with non-analyzable design points.

The ADOPT framework assumes that there is one simulation model involved in the optimization problem. An interesting future extension of the framework would be the possibility to develop SAO strategies for multiple coupled simulation models in the context of decomposition-based Multidisciplinary Optimization (MDO).

6.2 Variability measures based on effective process time

The concept of Effective Process Time (EPT) is a means to represent process time and all process disturbances by one distribution with mean t_e and coefficient of variation c_e. Since EPT relates to basic factory physics, parameters t_e and c_e can be used as a fundamental performance measure. In this thesis new algorithms have been developed to compute EPT parameters from operational factory data.

Effective process time algorithms have been proposed in Chapters 3 and 4. These algorithms can be used to measure effective process time realizations in a real-world factory based on arrival-event and departure-event data. In Chapter 3, EPT algorithms are proposed for workstations consisting of single-lot machines, i.e. machines that process one product at a time. The required data for these EPT algorithms is a list of arrival and departure events containing the lot identification number and the number of the machine the lot will be processed on (in case of arrival) or was processed on (in case of departure). The idea is that this data is used to compute the time that each lot claims capacity of a machine. A lot claims capacity of a machine whenever it is actually processed on the machine or whenever it is waiting for processing while the machine is not processing another lot.

For batching machines, algorithms are introduced that transform arrival events and departure events of individual lots into arrival events and departure events of *batches*. On the basis of these arrival and departure events of batches, the single-lot machine EPT algorithm can be used to compute the EPT parameter of processed batches. The transformation algorithms can account for the recipes of the individual lots. That is, the EPT calculation accounts for lots with different recipes that may not be processed together in one batch. Besides measuring the arrival and departure events of individual lots, these batch transformation algorithms also require the identification of the recipe for each lot.

Chapter 3 distinguishes two cases: EPT non-idling and EPT general. In the EPT non-idling case a lot that starts a capacity claim will in the future also be processed on that particular machine. In the EPT general case, this may be violated, e.g. when a lot is not dispatched on an available idle machine, but will be processed in the future on a machine that is currently busy. In the latter example capacity is lost due

to keeping a machine (effectively) idle for some reason. The EPT algorithm proposed in Chapter 3 accounts for this loss of capacity.

Chapter 4 shows that when the EPT non-idling assumption is obeyed, the EPT algorithm can be simplified and that EPT realizations can be computed for each machine separately. This observation allowed the extension of the algorithm towards batching equipment and recipe-dependent processing. Any violation of the EPT non-idling assumption is viewed to be part of the workstation dispatching. That is, it is due to the dispatching policy that a machine is kept idle from an EPT point of view. This is similar to the case that a recipe-based batch forming rule cannot generate a new batch for the idle batch machine since not sufficient lots of the same recipe are available.

The EPT algorithm for single-lot machines (Chapter 3) was validated using test examples of discrete-event simulation models of workstations. The test examples considered workstations with disturbances caused by failure of equipment and disturbances caused by unequal process times among the machines. The EPT algorithm was used to obtain the t_e and c_e parameters. The obtained EPT parameters were validated using an analytical queueing equation as well as simulation to confirm that t_e and c_e are correct representations of the effective capacity and effective variability, and to check whether t_e and c_e can be used to accurately predict flow time of the original simulation model. For the test examples that included unreliable machines (equipment downs), parameters t_e and c_e were able to accurately predict flow time. The computed t_e and c_e values are independent of the utilization. For a special case of equipment failures (equipment independent failures), we observed that EPT parameters t_e and c_e depend on utilization. This caused a small deviation in predicted flow times by the queueing equation with respect to the original simulation model. The EPT algorithm of Chapter 3 was, however, not able to correctly predict flow time in case of a large deviation in capacity among machines in one workstation.

The EPT algorithms for batch machines were validated using test examples that included unreliable machines and multiple recipes with different processing times. In these examples, the EPT non-idling assumption was obeyed and each machine in the workstation had equal capacity. The EPT-based simulation models were able to correctly predict the flow time of the original simulation model. Therefore, EPT parameters t_e and c_e correctly quantified effective capacity and effective variability in these examples.

Two case studies were performed based on data obtained from the Philips Semiconductor wafer fab MOS4YOU. The first case study considered all single-lot machine workstations and the second case study considered all batch machine workstations. In both case studies the t_e and c_e values were computed for each workstation based on data obtained from the Manufacturing Execution System (MES) which included arrival-event and departure-event data of lots. The calculated t_e and c_e for the single-lot machines were used in the $G/G/m$ queueing equation to estimate the flow time of the lots. The estimated flow time values were close to the real flow times obtained from the MES. A similar case study was carried out for batching equipment. Instead of using a queueing equation, here we used a simulation model that included the EPT parameters and the batch dispatching policy. Again, this simulation model

was able to generate similar results as obtained from the MES.

EPT can also be used in simulation models to predict flow time of lots. In the examples and the case study such simulation models of workstations were introduced. The individual machines in these workstation models are subject to process times that are distributed regarding the computed EPT parameters. The individual process times of the lots were distributed following a Gamma distribution with mean t_e and coefficient of variation c_e. Such a simulation model in which capacity and variability is based on EPT distributions alone is called a meta model in this thesis. An EPT-based meta model is a simulation model in which disturbances are not modeled separately, but EPT distributions are used instead. This idea of EPT-based simulation meta modeling gives good prospects to create simulation meta models of the complete manufacturing system, i.e. by linking all the meta models of the individual workstations. This meta model would be a valuable tool to predict flow time performance of a complete (re-entrant) flow line. The meta model could provide an accurate simulation model that can be used in the optimization. One may optimize using the simulation meta model for structural changes related to, e.g., batch sizes, buffer sizes, and number of machines in a workstation.

A further question for research is: what should be included in EPT? The initial concept of EPT started with the idea that processing and all disturbances should be included in EPT. From Chapters 3 and 4 can be concluded that this might not always be the best approach. The EPT algorithm presented in Chapter 3 is based on the underlying $G/G/m$ queueing physics. If a workstation operates more or less like such a $G/G/m$ queueing system, the proposed EPT algorithm can be used satisfactory. In Chapter 4 the $G/G^k/1$ queueing equation is used to understand the physics of a workstation with batch machines that obeys the non-idling assumption. For cases that strongly violate the queueing assumptions, the underlying physics have to be described with an appropriate queueing equation or simulation model. This might be the case for specific shop floor realities, such as dispatching policies and batching rules. These shop floor realities cannot be accounted for in EPT realizations and should be modeled explicitly in the (simulation) meta model. For types of equipment other than single-lot and batching equipment the EPT algorithms and meta models may be adapted to account for specific process capabilities. This also holds for specific process disturbances that violate the $G/G/m$ queueing assumptions, such as large capacity differences among machines in one workstation, utilization dependent EPT realizations, or non-idling due to infrequent but long machine interruptions (see for a detailed example, e.g., Wullems, 2003).

6.3 Approximation method for flow time performance

Chapter 5 proposes a new approximation method to describe product flow time of an open queueing network with a fixed routing. Semiconductor wafer fabrication is an example of such a queueing network. Starting point is that the proposed approximations will be used in a Sequential Approximate Optimization (SAO) approach and are able to provide good quality approximations of flow time performance in the search subregion of each SAO cycle.

The idea of the newly developed approximation method is the incorporation of queueing physics of each individual workstation in the approximation model of the total product flow time. The approximation method builds flow time approximations in each cycle for each workstation separately. Since the routing of each product flow is fixed, the product flow time is the sum of flow time accumulated at each (re-) visited workstation. Two basic queueing physics principles are incorporated for each workstation: (i) for an utilization close to 1.0, the workstation flow time increases non-linearly to infinity, and (ii) the variability has a proportional effect on the workstation flow time. As a consequence, the flow time approximation of each workstation should depend on: the design variables itself, the utilization response, and the variability response, i.e. $\varphi = \varphi(\mathbf{x}, u(\mathbf{x}), c^2(\mathbf{x}))$.

For the workstation flow time approximation, Chapter 5 proposed a new linear regression model which consists of a pure linear model in the design variables and one additional linear regression term that accounts for the queueing physics. The additional linear regression term includes the two above-mentioned queueing physics principles by means of the following regression term: $c^2(\mathbf{x})/(1-u(\mathbf{x}))$. For the approximate functions $u(\mathbf{x})$ and $c^2(\mathbf{x})$ we suggest linear regression models including linear and reciprocal terms that match well with typical design variables for manufacturing systems. The linear regression parameters in each of the three approximation models are estimated using simulation responses of utilization, variability, and flow time. From a simulation run the individual workstation flow time can be obtained directly. However, the workstation utilization and variability responses cannot be determined directly. The idea is to determine utilization and variability from simulation events on the basis of EPT.

Key contribution of the proposed approximation model is that EPT is used to describe the utilization and variability approximation models. Based on the $G/G/m$ queueing equation we suggest to describe utilization and variability by $u(\mathbf{x}) = t_e/(t_a \cdot m)$ and $c^2(\mathbf{x}) = (c_a^2 + c_e^2)/2$, respectively. The utilization and variability responses are determined on the basis of the developed EPT algorithms. These EPT algorithms determine the t_e and c_e^2 values for each workstation based on arrival and departure events of the lots.

The newly developed flow time approximation model has been incorporated in a multi-point sequential approximate optimization approach. The presented sequential approximate optimization approach is used for the optimization of flow time performance in discrete-event manufacturing systems. In particular, the presented optimization problem aims to minimize a deterministic cost function with respect to a number of stochastic flow time constraints. The flow time constraints are related to the total flow time of each product type, where each product type has a fixed routing. The proposed flow time approximations are used to approximate the constraints in each iteration of the SAO sequence.

An SAO sequence for simulation-based optimization has been developed for solving the above-mentioned optimization problem. The SAO sequence builds flow time approximations in each iteration. The sequence can handle non-analyzable simulations. A simulation is non-analyzable, if the utilization of one of the workstations is equal to or above 100%. This occurs whenever the throughput of the product flows

exceeds the capacity of the workstation. For such a system, a steady-state flow time response is never obtained. Whether a design point is analyzable or not, cannot be determined on beforehand. It might happen that a plan point in the design of experiments step of the SAO sequence is non-analyzable. After simulating all plan points, the non-analyzable design point is repositioned in the design of experiments and simulated again. The proposed SAO sequence is able to perform this task.

The newly developed SAO sequence that included the new approximation function has been tested on two simulation-based optimization problems. The optimization problems considered a four-station flow line and a twelve-station re-entrant flow line, respectively. The optimization problems are defined to minimize cost subject to flow time constraints. For the four-station flow line, all machines in the discrete-event simulation model are single-lot machines with distributed processing times and no process disturbances. The number of machines in each workstation are treated as integer design variables. The dispatching policy was first-come-first-served for all workstations. These workstations match very well with the $G/G/m$ queueing behavior. The SAO approach was able to find the correct optimum in 100% of the optimization runs. The twelve-station flow line consists of single-lot machine workstations and batch machine workstations. The machines are subject to process times, machine failure, and recipe-dependent setup times. Furthermore, the dispatching policies are also based on the recipes of the individual lots. For this reason, the workstation behavior may deviate from the $G/G/m$ queueing equation. However, the SAO approach was still able to generate good quality approximations. Two optimization problems were considered. The first optimization problem considered the number of machines in each of the twelve workstations as design variable. The cost of the optimal solution was equal to 7,605 k$. The second optimization problem considered three integer design variables: the number of machines at three stations, and five continuous design variables: the mean setup time of two stations and the mean repair time of three stations. Since an increase of mean setup time was allowed, less machines were needed and the cost of the optimal solution was equal to 6,362 k$.

Approximation models based on factory physics result in good quality flow time approximations. The approximations presented here are based on $G/G/m$ flow time performance. For workstations that behave similarly to a $G/G/m$ system, the presented method proved to work well. If other types of manufacturing systems are considered, it might be beneficial to develop case-specific approximation models. These case-specific approximation models should be able to accurately approximate the flow time performance in relation to the existing shop floor realities. Specific shop floor realities are, for example, batching equipment, dispatch rules, control strategies, and blocking.

Bibliography

Abspoel, S. J., Etman, L. F. P., Vervoort, J., van Rooij, R. A., Schoofs, A. J. G., and Rooda, J. E. (2001). Simulation based optimization of stochastic systems with integer design variables by sequential multipoint linear approximation. *Structural and Multidisciplinary Optimization*, 22(2):125–138.

Adan, I. J. B. F. and Resing, J. A. C. (2001). Queueing theory. Lecture notes, Eindhoven University of Technology, `ftp://ftp.win.tue.nl/pub/stoch-or/queueing.pdf`.

Alexandrov, N. M., Dennis, J. E., Lewis, R. M., and Torczon, V. (1998). A trust region framework for managing use of approximation models in optimization. *Structural Optimization*, 15(1):16–23.

Ames, V. A., Gililland, J., Konopka, A., and Barber, H. (1995). Semiconductor manufacturing productivity; overall equipment effectiveness (OEE) guidelines. Technology transfer #950327443 A-GEN, Revision 1.0, Sematech, `http://www.sematech.org/`.

Angün, E., Gürkan, G., Den Hertog, D., and Kleijnen, J. (2003). Response surface methodology with stochastic constraints for expensive simulation. submitted.

Azadivar, F. (1999). Simulation optimization methodologies. In Farrington, P. A., Nemhard, H. B., Evans, G. W., and Sturrock, D., editors, *Proceedings of the 1999 Winter Simulation Conference*, pages 93–100, Piscataway, NJ.

Barthelemy, J. F. M. and Haftka, R. T. (1993). Approximation concepts for optimum structural design: a review. *Structural Optimization*, 5(3):129–144.

Bisschop, J. and Roelofs, M. (2002). *AIMMS – The User's Guide*. Paragon Decision Technology, Haarlem, The Netherlands, `http://www.aimms.com/`.

Brekelmans, R., Driessen, L., Hamers, H., and Den Hertog, D. (2004). Constrained optimization involving expensive function evaluations: a sequential approach. *European Journal of Operational Research*, to be published.

Brooke, A., Kendrick, D., Meeraus, A., and Raman, R. (1998). *GAMS – A User's Guide*. Washington, `http://www.gams.com/`.

Bruyneel, M., Duysinx, P., and Fleury, C. (2002). A family of MMA approximations for structural optimization. *Structural and Multidisciplinary Optimization*, 24(4):263–276.

Buzacott, J. A. and Shanthikumar, J. G. (1993). *Stochastic Models of Manufacturing Systems*. Prentice Hall, Englewood Cliffs.

Carson, Y. and Maria, A. (1997). Simulation optimization: methods and applications. In Andradottir, S., Healy, K. J., Withers, D. H., and Nelson, B. L., editors, *Proceedings of the 1997 Winter Simulation Conference*, pages 118–126, Piscataway, NJ.

Chang, P. L., Huang, M. G., and Chou, Y. C. (1998). Evaluating system performances for semiconductor fabrication using open general queueing networks. *International Journal of Operations & Quantitative Management*, 4(3):327–342.

Craig, K. J. and Stander, N. (2003). An improved version of DYNAMIC-Q for simulation-based optimization using response surface gradients and an adaptive trust region. *Communications in Numerical Methods in Engineering*, 19(11):887–896.

Craig, K. J., Stander, N., and Balasubramanyam, S. (2003). Worst-case design in head impact crashworthiness optimization. *International Journal for Numerical Methods in Engineering*, 57(6):795–817.

Eldred, M. S., Giunta, A. A., van Bloemen Waanders, B. G., S. F. Wojtkiewicz, J., Hart, W. E., and Alleva, M. P. (2002). *DAKOTA, a Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis*. Sandia, Albuquerque, NM, version 3.0, `http://endo.sandia.gov/DAKOTA/software.html`.

Elishakoff, I., Haftka, R. T., and Fang, J. (1994). Structural design under bounded uncertainty – optimization with anti-optimization. *Computers and Structures*, 53(6):1401–1405.

Etman, L. F. P., Adriaens, J. M. T. A., van Slagmaat, M. T. P., and Schoofs, A. J. G. (1996). Crash worthiness design optimization using multipoint sequential linear programming. *Structural Optimization*, 12:222–228.

Fadel, G. M., Riley, M. F., and Barthelemy, J. F. M. (1990). Two point exponential approximation method for structural optimization. *Structural Optimization*, 2:117–129.

Fadel, G. M. and Cimtalay, S. (1993). Automatic evaluation of move-limits in structural optimization. *Structural Optimization*, 6:233–237.

Fletcher, R. and Leyffer, S. (1998). Nonlinear programming without a penalty function. Numerical analysis report NA/171, University of Dundee, Dundee, Scotland.

Fleury, C. and Braibant, V. (1986). Structural optimization: a new dual method using mixed variables. *International Journal for Numerical Methods in Engineering*, 23(3):409–428.

Fourer, R., Gay, D. M., and Kernighan, B. W. (1993). *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press.

Fu, M. C. (1994). Optimization via simulation: A review. *Annals of the Operations Research*, 53:199–247.

Gijsbers, J. A. A. (2002). A sequence of linear regression approximations for simulation-based optimization of manufacturing systems. Master's thesis, Eindhoven University of Technology, Systems engineering report SE-420314.

Gijsbers, J. A. A. (2003). Integer simulation-based optimization: a new multipoint approximation approach with probabilistic filter acceptance and intermediate design variables. Systems engineering report SE-420398, Eindhoven University of Technology.

Giunta, A. A. and Eldred, M. S. (2000). Implementation of a trust region model management strategy in the DAKOTA optimization toolkit. In *Proceedings of the AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Long Beach, CA.

Glasserman, P. (1991). *Gradient Estimation via Perturbation Analysis*. Kluwer Academic, Boston.

Gosavi, A. (2003). *Simulation-based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Kluwer Academic, Boston.

Gurav, S. P., Langelaar, M., Goosen, J. F. L., and van Keulen, F. (2003). Different approaches to deal with bounded-but-unknown uncertainty-based design optimization: Application to mems. In *Proceedings of The Fifth World Conference for Structural and Multidisciplinary Optimization*, Venice, Italy.

Gurav, S. P., Kasyap, A., Sheplak, M., Cattafesta, L., Haftka, R. T., Goosen, J. F. L., and Van Keulen, F. (2004). Uncertainty-based design optimization of a micro piezoelectric composite energy reclamation device. In *Proceedings of 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Washington, DC.

Haftka, R. T., Nachlas, J. A., Watson, L. A., Rizzo, T., and Desai, R. (1987). Two point constraint approximation in structural optimization. *Computer Methods in Applied Mechanics and Engineering*, 60(3):289–301.

Haftka, R. T. and Gürdal, Z. (1991). *Elements of Structural Optimization*. Kluwer Academic, Dordrecht, The Netherlands, 3rd edition.

HKS (2002). *Abaqus Version 6.3 Documentation*. Hibbitt, Karlsson & Sorensen, Inc., Pawtucket, RI, `http://www.abaqus.com/`.

Hofkamp, A. T. (2004). Batchlib – a parallel computation backend in Python. Systems engineering report, Eindhoven University of Technology, in preparation.

Hopp, W. J. and Spearman, M. L. (2001). *Factory Physics: Foundations of Manufacturing Management*. Irwin McGraw-Hill, London, 2nd edition.

Hopp, W. J., Spearman, M. L., Chayet, S., Donohue, K. L., and Gel, E. S. (2002). Using an optimized queueing network model to support wafer fab design. *IIE Transactions*, 34(2):119–130.

Kleinrock, L. (1975). *Queueing systems*. Wiley-Interscience, London.

Koch, P. N., Evans, J. P., and Powell, D. (2002). Interdigitation for effective design space exploration using iSIGHT. *Structural and Multidisciplinary Optimization*, 23(2):111–126.

Kock, A. A. A., Wullems, F. J. J., Etman, L. F. P., Adan, I. J. B. F., and Rooda, J. E. (2004). Performance evaluation and simulation meta-modelling of single server flow lines subject to blocking: an effective process time approach. *IIE Transactions*, submitted.

Law, A. M. and Kelton, W. D. (2000). *Simulation Modeling and Analysis*. McGraw-Hill, Boston, 3rd edition.

Lawrence, C. T., Zhou, J. L., and Tits, A. (1997). User's guide for CFSQP version 2.5: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. Technical report TR-94-16r1, Institute for Systems Research, University of Maryland.

Little, J. D. C. (1961). A proof for the queueing formula $L = \lambda W$. *Operations Research*, 9(3):383–387.

Lu, S. C. H., Ramaswamy, D., and Kumar, P. R. (1994). Efficient scheduling policies to reduce mean and variance of cycle-time in semiconductor plants. *IEEE Transactions on Semiconductor Manufacturing*, 7(3):374–388.

Mathworks (2002). *Using Matlab*. The Mathworks, Inc., Natick, MA, version 6.5, `http://www.mathworks.com/`.

Montgomery, D. C. and Runger, G. C. (2003). *Applied Statistics and Probability for Engineers*. Wiley, London, 3rd edition.

Myers, R. H. and Montgomery, D. C. (2002). *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley-Interscience, New York, 2nd edition.

Nakajima, S. (1988). *Introduction to TPM: Total Productive Maintenance*. Productivity Press, Cambridge, MA.

OTC (2003). *NEOS guide*. Optimization Technology Center, `http://www.ece.northwestern.edu/OTC/`.

Papalambros, P. Y. and Wilde, D. J. (2000). *Principles of Optimal Design: Modeling and Computation*. Cambridge University Press, New York.

Park, S., Mackulak, G. T., and Fowler, J. W. (2001). An overall framework for generating simulation-based cycle time-throughput curves. In Peters, B. A., Smith, J. S., Medeiros, D. J., and Rohrer, M. W., editors, *Proceedings of the 2001 Winter Simulation Conference*, pages 1178–1187, Washington, DC.

Pedersen, P. (1981). *Optimization of Distributed Parameter Structures*, chapter The Integrated Approach of FEM-SLP for Solving Problems of Optimal Design. Sijthoff and Noordhoff, Alphen aan de Rijn.

Pérez, V. M., Renaud, J. E., and Watson, L. (2002). Interior point sequential approximate optimizaion methodology. In *Proceedings of the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA.

Phoenix (2004). ModelCenter overview. Technical report, Phoenix Integration, Blacksburg, VA, `http://www.phoenix-int.com/`.

Rooda, J. E. and Vervoort, J. (2003). Analysis of manufacturing systems. Lecture notes, Eindhoven University of Technology, `http://se.wtb.tue.nl/documentation`.

Rubinstein, R. Y. (1986). *Monte-Carlo Optimization: Simulation and Sensitivity of Queueing Networks*. Wiley, New York.

Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language Reference Guide*. Addison Wesley, Reading.

Sattler, L. (1996). Using queueing curve approximations in a fab to determine productivity improvements. In *Proceedings of the 1996 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pages 140–145, Cambridge, MA.

Schmit, L. A. and Farshi, B. (1974). Some approximation concepts for structural synthesis. *AIAA Journal*, 12(5):692–699.

Schömig, A. K. (1999). On the corrupting influence of variability in semiconductor manufacturing. In Sturrock, D., Fishwick, P. A., Farrington, P. A., and Nembhard, H. B., editors, *Proceedings of the 1999 Winter Simulation Conference*, pages 837–842, Pascataway, NJ.

SEMI (2000). Standard for definition and measurement of equipment productivity. Technical report SEMI E79-0200, Semiconductor Equipment and Materials International, `http://www.semi.org/`.

Snyman, J. A. and Hay, A. M. (2002). The Dynamic-Q optimization method: an alternative to SQP? *Computers and Mathematics with Applications*, 44(12):1589–1598.

Stander, N., Eggleston, T., Craig, K. J., and Roux, W. (2003). *LS-OPT User's Manual*. Livermore Software Technology Corporation, Livermore, CA, `http://www.lstc.com/`.

Sterian, A. (1999). *PyMat – An interface between Python and Matlab*. version 1.02, `http://claymore.engineer.gvsu.edu/~steriana/Python/pymat.html`.

Sturm, R., Silvi, T., Frauenhoffer, F., and Treiber, T. (1999). A simulation model for advanced release and order planning. *Future Fab International*, 6:71–74.

Svanberg, K. (1987). The method of moving asymptotes – a new method for structural optimization. *International Journal for Numerical Methods in Engineering*, 24:359–373.

Svanberg, K. (1995). A globally convergent version of MMA without linesearch. In Rozvany, G. I. N. and Olhoff, N., editors, *Proceedings of the First World Congress of Structural and Multidisciplinary Optimization*, pages 9–16, Goslar, Germany.

Svanberg, K. (1999). The MMA for modeling and solving optimization problems. In *Proceedings of the Third World Congress of Structural and Multidisciplinary Optimization*, Amhert, NY.

Thomas, H. L., Vanderplaats, G. N., and Shyy, Y. K. (1992). A study of move limit adjustment strategies in the approximation concepts approach to structural synthesis. In *Proceedings of the 4th AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization*, pages 507–512, Cleveland, OH.

Toropov, V. V., Filatov, A. A., and Polynkin, A. A. (1993). Multiparameter structural optimization using FEM and multipoint explicit approximations. *Structural Optimization*, 6:7–14.

Toropov, V. V., Van Keulen, F., Markine, V. L., and De Boer, H. (1996). Refinements in the multi-point approximation method to reduce the effects of noisy responses. In *Proceedings of the 6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pages 941–951, Bellevue, WA.

Trolltech (2001). *Programming with QT*. Trolltech AS, Oslo, Norway, version 3.0, `http://www.trolltech.com/`.

Uzsoy, R., Lee, C. Y., and Martin-Vega, L. A. (1992). A review of production planning and scheduling models in the semiconductor industry, part I. *IIE Transactions*, 24(4):47–61.

Van Campen, E. J. J. (2001). *Design of a multi-process multi-product wafer fab*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

Van Rooij, R. A. (2004). *Sequential Approximate Optimization for Design Robustness and Reliability*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, in preparation.

Van Rossum, G. and Drake, F. L. (2003a). *Python Reference Manual*. `http://www.python.org/doc/`.

Van Rossum, G. and Drake, F. L. (2003b). *Extending and Embedding the Python Interpreter*. `http://www.python.org/doc/`.

Vanderplaats, G. N. (1976). CONMIN – a FORTRAN program for constrained function minimization: User's manual. Technical report TM X-62, 282, NASA.

Vanderplaats, G. N. (1979). Approximation concepts for numerical airfoil optimization. Technical report 1370, NASA.

Vijfvinkel, M. (2004). Filter methods for accepting solutions of approximate optimization subproblems in an integer simulation-based optimization algorithm. Systems engineering report SE-420395, Eindhoven University of Technology.

Wang, L. P. and Grandhi, R. V. (1995). Improved two-point function approximation for design optimization. *AIAA Journal*, 33(9):1720–1727.

Wujek, B. A. and Renaud, J. E. (1998). New adaptive move-limit management strategy for approximate optimization, part 1. *AIAA Journal*, 36(10):1911–1921.

Wullems, F. J. J. (2003). Data collection for simulation flow lines with blocking; the role of machine failure in throughput loss. Master's thesis, Eindhoven University of Technology, Systems engineering report SE-420316.

Samenvatting

Dit promotieonderzoek maakt deel uit van het ADOPT-project, dat gefinancierd wordt door de Technologiestichting STW. ADOPT is een samenwerkingsverband tussen de technische universiteiten van Eindhoven en Delft. Doel van het ADOPT-project is om gezamenlijk te komen tot de ontwikkeling van een optimaliseringsgereedschap waarin optimaliseringsproblemen op basis van rekenintensieve modellen kunnen worden beschreven en opgelost. De herhaald benaderende optimaliseringsstrategie zal gebruikt worden om een directe koppeling tussen het rekenintensieve model en een wiskundig optimaliseringsalgoritme overbodig te maken. Dit wordt bereikt door het opeenvolgend genereren en oplossen van benaderende optimaliseringsproblemen. In deze benaderingen kunnen probleemspecifieke kenmerken worden vastgelegd. Kenmerken die binnen het ADOPT-project worden beschouwd zijn: stochastische en ruisgevoelige responsies, discontinuïteiten in responsies en integer ontwerpvariabelen.

De bijdrage van dit proefschrift is drieledig. Allereerst is er voor het ADOPT-project een framewerk ontwikkeld waarin herhaald benaderende optimaliseringsstrategieën kunnen worden geïmplementeerd. Dit framewerk levert een open structuur waarin de optimaliseringsstrategie en de benaderingen kunnen worden aangepast naar de kenmerken van het beschouwde optimaliseringsprobleem. Op de tweede plaats is er specifiek voor de optimalisatie van productielijnen een nieuw benaderingsconcept ontwikkeld. Deze productielijnen worden verondersteld gemodelleerd te zijn met behulp van discrete-event simulatie. Kenmerkend voor deze simulatie zijn de stochastische responsies en integer ontwerp variabelen. Typische integer ontwerpvariabelen zijn het aantal gebruikte machines en de grootte van de geproduceerde batches. Tot slot wordt in dit proefschrift de zogenaamde effectieve-procestijd-methode (EPT) beschreven. Deze methode is gebaseerd op wachtrijtheorie en levert EPT als nieuwe prestatie-indicator voor de doorlooptijd. Verder kan het EPT concept als basis dienen voor eenvoudige maar realistische simulatiemodellen, omdat details op de werkvloer aangaande productieonderbrekingen niet afzonderlijk worden meegenomen.

Het framewerk voor herhaald benaderend optimaliseren is gebaseerd op een objectgeoriënteerde structuur en bestaat uit methoden, (externe) numerieke routines en interfaces naar andere softwarepakketten. In het framewerk is het mogelijk om (i) het optimaliseringsprobleem te specificeren dat o.a. het simulatie model bevat, (ii) de optimaliseringssequentie te specificeren die de volgorde van numerieke stappen

bepaalt en (iii) de numerieke routines te specificeren die gebruikt worden in iedere numeriek stap. Een typische optimaliseringssequentie bestaat uit een aantal stappen die bedoeld zijn voor o.a. het uitvoeren van een proefopzet, het opbouwen van de benaderingen en het oplossen van de benaderende optimalisatieproblemen. Iedere afzonderlijke numerieke stap kan beschouwd worden als een 'black-box' functie zoals bijvoorbeeld verkregen via een externe software bibliotheek. De gebruiker van het framewerk kan de optimalisatiesequentie en de gebruikte numerieke routines aanpassen. In beschikbare herhaald benaderende optimalisatie-implementaties is dit over het algemeen niet mogelijk. Het framewerk is toegepast op het ontwerpprobleem van een vakwerk met tien staven waarbij de belasting zowel bekend alsook onzeker is. Het framewerk is daarnaast gebruikt voor ontwerpproblemen binnen het ADOPT-project waaronder de optimalisatie van Microelectromechnical Systems (MEMS) en productielijnen, en de optimalisatie voor robuustheid en betrouwbaarheid.

De effectieve-procestijd-methode (EPT) bepaalt de capaciteit en variabiliteit van machines en werkstations van productielijnen. Het concept van EPT is bekend vanuit de literatuur, maar er is nog geen methode beschikbaar om daadwerkelijk de EPT te meten in operationele fabrieken of simulatiemodellen. EPT voegt productietijden en productieverstoringen samen tot één enkele prestatie-indicator. Typische voorbeelden van productieverstoringen zijn het faalgedrag van machines, insteltijden, en operator-beschikbaarheid. Het doel is om te komen tot algoritmes die EPT parameters kunnen bepalen op basis van gegevens uit de productielijn. Deze parameters worden in combinatie met overeenstemmende wachtrijfysica worden gebruikt als prestatie-indicator voor de doorlooptijd. Algoritmes zijn ontwikkeld voor de bepaling van EPT voor machines die gebruikt worden in de semiconductorindustrie. De algoritmes zijn getest op simulatievoorbeelden en op operationele data uit een fabriek van Philips Semiconductors. Dit resulteerde in simulatie-metamodellen van de afzonderlijke werkstations, die vervolgens gecombineerd kunnen worden tot een metamodel van de complete fabriek.

De optimalisatie van productielijnen vereist benaderingsmodellen met een hoge nauwkeurigheid. In dit proefschrift wordt de prestatie van doorlooptijden in productielijnen geoptimaliseerd. Vanuit de wachtrijtheorie is bekend dat de doorlooptijd zich voor bepaalde ontwerpvariabelen sterk niet-lineair gedraagt. De ontwikkelde benaderingsmodellen houden rekening met dit fysisch niet-lineaire gedrag. In voorgaand onderzoek is het gebruik van lineaire regressie modellen voorgesteld. Er wordt hierop voortgebouwd en een nieuwe meer generieke lineaire regressiebenadering voorgesteld uitgaande van het concept van de EPT. De gedachte is om EPT te gebruiken in de benaderingsmodellen voor de doorlooptijd. Deze benaderingsmodellen hebben dan een directe relatie met wachtrijtheorie. De te schatten parameters in de benaderingsmodellen worden bepaald met simulatieresponsies van een discrete-event model. Dit nieuwe type benaderingsmodel is toegevoegd aan het ontwikkelde framewerk voor herhaald benaderend optimaliseren. Dit heeft geleidt tot een nieuwe methode voor optimalisatie van productielijnen. Deze methode is succesvol getest op ontwerpproblemen voor productielijnen met vier werkstations en twaalf werkstations.

Curriculum Vitae

1975	Geboren te Harderwijk
1988-1994	Atheneum aan het Kruisheren Kollege te Uden
1994-1999	Werktuigbouwkunde aan de Technische Universiteit Eindhoven, afstudeerrichting Systems Engineering
1999-2004	Assistent in Opleiding bij de groep Systems Engineering van de Technische Universiteit Eindhoven
2001	Winnaar "DuPont Photomasks Best Paper Award" op de Advanced Semiconductor Manufacturing Conference voor de congresbijdrage *Quantifying operational time variability: The missing parameter for cycle time reduction* (met L.F.P. Etman, E.J.J. van Campen en J.E. Rooda)

Dankwoord

Graag wil ik de volgende mensen bedanken die een bijdrage hebben geleverd aan dit proefschrift:

- Pascal Etman en Koos Rooda voor de bezielende begeleiding en discussies. Met name zij hebben mij gevormd tot de wetenschappelijk onderzoeker die ik geworden ben.
- De andere leden van het ADOPT-team: René van Rooij, Koen Vervenne, Sham Gurav, Bert Schoofs, Koo Rijpkema, Henk Nijmeijer en Fred van Keulen. Het werken in een dergelijk multidisciplinair team heeft mij kennis laten maken met andere onderzoeksgebieden.
- De studenten die een bijdrage hebben geleverd aan het onderzoek: Paul van Bakel, Maarten Rooney, Joost Gijsbers, en Ad Kock.
- De medewerkers van de groep Systems Engineering, met name Albert Hofkamp voor zijn adviezen ten aanzien van het object-georiënteerde ontwerp van het optimaliseringsframewerk en om dit onderzoek te gebruiken voor het testen van zijn ontwikkelde Batchlib software. Deze software heeft het mogelijk gemaakt om meerdere simulaties tegelijkertijd op een parallel Linux-rekencluster uit te voeren. Mieke Lousberg bedank ik voor het luisteren naar de vele verhalen uit mijn dagelijkse leven.
- Medewerkers van Philips Semiconductors, met name Aubin Wilkens en Edgar van Campen voor de inspirerende discussies en Frans Brouwers voor het leveren van de benodigde operationele gegevens.
- Margriet Jansz van de Technologiestichting STW voor haar begeleiding en de gebruikerscommissie voor hun interesse in de resultaten van het onderzoek.
- Ivo Adan en Onno Boxma van de groep Stochastische Besliskunde voor hun wiskundige ondersteuning en suggesties.
- Mijn ouders, familie en vrienden voor hun geduld, steun en vertrouwen.
- Susan, omdat jij begrijpt wie ik ben, wat ik voel en wat ik wil bereiken.

Johan Jacobs
Geldrop, 13 september 2004

Stellingen

behorende bij het proefschrift

Performance Quantification and Simulation Optimization of Manufacturing Flow Lines

1. Het opnemen van zo veel mogelijk detail in een simulatiemodel van een fabriek leidt niet tot nauwkeurigere voorspellingen.
   Dit proefschrift, hoofdstuk 1

2. Een correct algoritme voor de berekening van effectieve procestijden maakt slechts gebruikt van aankomst- en vertrekgebeurtenissen van producten.
   Dit proefschrift, hoofdstuk 3

3. Het toevoegen van fysische kennis aan benaderingsmodellen is voor een succesvolle optimalisering noodzakelijk.
   Dit proefschrift, hoofdstuk 5

4. Hoewel een machine niet meer dan 100% beladen kan zijn, kan een utilisatie van meer dan 100% wel een betekenis hebben.
   Dit proefschrift, hoofdstuk 5

5. Het effect van variabiliteit in fabricagesystemen wordt ten onrechte niet meegenomen in operationele beslissingen.

6. Een analyse model kan alleen als simulatiemodel worden betiteld als er stochastische gebeurtenissen aan ten grondslag liggen.

7. MATLAB is geen object-georiënteerde programmeertaal.

8. Hoewel in het basisonderwijs veel didactische methodes bestaan om zwakkere leerlingen te helpen, is het minstens zo belangrijk om betere leerlingen uitdagingen te bieden.
   Centrum voor ErvaringsGericht Onderwijs, `http://www.cego.be/`

9. De gelijkzwevende stemming klinkt zo gek nog niet.
   Simon Stevin, "Vande Spiegheling der Singconst", manuscript van rond 1600.

10. Kindertheater is geen kinderspel.
    Arie Spinazie, `http://www.ariespinazie.nl/`

11. Een manager van een fastfood-restaurant wordt verplicht om verse hamburgers weg te gooien, maar wordt verleid om oude hamburgers te verkopen.

12. Door het gebruik van route-navigatiesystemen, zoals GPS, is men vaker de weg kwijt.