Control of flexible manufacturing systems
with non-zero setup times

H.P.J. Felten
s0474102

SE 420 423

Master's Thesis

Supervisor: prof. dr. ir. J.E. Rooda
Coaches: dr. A.Y. Pogromsky
dr. ir. A.A.J. Lefeber

Technische Universiteit Eindhoven
Department of Mechanical Engineering
Systems Engineering Groep

Eindhoven, April 2005

FINAL ASSIGNMENT

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mechanical Engineering
Systems Engineering Group

February 2004

Student	H.P.J. Felten
Supervisor	Prof.dr.ir. J.E. Rooda
Advisors	Dr.ir. A.A.J. Lefeber Dr. A. Pogromsky
Start	February 2004
Finish	February 2005
Title	Verification of continuous-time models for control of manufacturing lines

Subject

The modern flexible manufacturing lines consist of a number of machines, buffers and transportation lines. Some of the machines can perform multiple operations, so one of the control problems for such manufacturing lines is to determine a switching policy for the machines to guarantee a certain objective, for example to minimize the time required to achieve the desired production level. The problem of control of manufacturing lines is extremely difficult in its full generality. However, recently, some of the control problems were solved under assumptions that the manufacturing line can be approximately modelled as a system of differential logical equations. The goal of the project is to investigate to what extend the results derived for the simplified models can be utilized to control the real manufacturing lines.

Assignment

First, for given manufacturing systems its approximate model should be found. This model should belong to a certain class of differential logical equations. For the simplified model, using the analytical methods, a switching control policy should be determined. This switching policy should minimize the time required to achieve the desired production levels. Next step of the project is to investigate effect of the discretization, batching and stochasticity on the performance of the controlled manufacturing line.

Prof.dr.ir. J.E. Rooda Dr.ir. A.A.J. Lefeber Dr. A. Pogromsky

Preface

This Master's thesis is the final assignment of a five year curriculum for becoming Master of Science in Mechanical Engineering, with a specialization in Systems Engineering. The Systems Engineering Group, at the Technische Universiteit Eindhoven, aims to develop methods, techniques and tools for the design of advanced industrial systems. It focuses on production systems comprising multiple communicating subsystems that work in parallel. An example of such a system is a production system which processes multiple part types.

My final assignment was performed within the Hybrid Manufacturing Systems research theme, which is one of the research themes of the System Engineering Group. The research of my Master's thesis focussed on controlling production systems which process multiple part types. Within the Hybrid Manufacturing Systems research theme this was the first step into the research of controlling and optimizing such systems.

First of all, I would like to expres my highest appreciation to my coaches Sasha Pogromski and Erjen Lefeber for there guidance and support during my final assignment. I am also very grateful to professor Rooda and the other employees of the Systems Engineering groep for providing the interesting environment in which I could study. Furthermore, I would like to thank my parents who were always there to support me and who provided me the means to study. Also, my thanks goes out to my little sister Hanneke for the many interesting discussions during diner on the must irrelevant subjects. I would also like to thank my girlfriend Cynthia for her support, patience and help during my study. Finally, I would like to thank everybody who supported me or provided me with a great time during my study, especially, my good friends Harld and Bas, my study colleagues Marco, Frans, Rudolf, Casper and all the people from Panache. I would like to end with a quote from a famous engineer.

"I cannae change the laws of physics"
Montgomery "Scotty" Scott, chief engineer U.S.S. Enterprise

Henk-Pieter Felten
Eindhoven, April 2005

Summary

Modern flexible manufacturing systems consist of a number of machines, buffers and transportation lines. Some of the machines are capable of processing multiple types of parts or performing multiple operations on the same part. If the next part type or operation is different from the previous part type or operation a setup is often required for these machines. However, production capacity is lost every time a setup is performed. Therefore, the policy that decides when to perform a setup has a large influence on the performance of the manufacturing system.

Flexible manufacturing systems are distinguished in two different classes with regard to the production paths of the parts. A flexible manufacturing system is called *acyclic* if the machines can be ordered so that parts can only move from one machine to a machine higher in the ordering. The system is called *non-acyclic* if such an ordering is not possible. For a production system that only consists of one machine, a single machine system, also a distinction is made on basis of the part type properties. A single machine system is said to be *symmetric* if all part types have identical arrival rate, setup time and process rate distributions, otherwise the system is said to be *asymmetric*.

Various control policies for flexible manufacturing systems exist. From these policies, three promising policies have been chosen, namely the Savkin policy (SAV) [Sav03], the Clear the Largest Works policy (CLW) [Per89] and the Clear the Largest Scaled Age policy (CLSA) [Ols99]. For CLW it has been proven in [Liu92] that it is optimal with respect to minimizing the amount of (unfinished) work present in a symmetric stochastic system with discrete parts. For CLSA it has been shown by simulation in [Ols99] that this policy performs well compared to other policies and that it functions in a stochastic system with discrete parts. A disadvantage of both policies is that they are not guaranteed stable for non-acyclic systems. To overcome this problem [Hum94] introduced a technique called *regulators* which stabilize these policies. SAV does not need these regulators, because it is already stable by itself for non-acyclic systems. A drawback of SAV is that it is designed for a system without variability or for systems with a special limited variability. Furthermore, the work in a system can only be continuous. Therefore, it has been studied whether SAV can function in a stochastic system with discrete parts. It turned out that SAV can not function in such an environment. To partially overcome this drawback SAV has been adapted such that it functions in a discrete system with a triangular distribution for the arrival and process rates. The adapted Savkin policy is denoted with SAV^a.

The three policies have been compared with each other for a simple single machine system with acyclic production paths and for a more complex manufacturing system with non-acyclic production paths. The comparison was made by simulating a discrete system with a triangular

distribution for the arrival and process rate. As a performance measure the average flow time of all the parts has been used. The results show that CLSA has the best performance compared with the other policies for single machine systems with acyclic production paths. SAV^a has the best performance for non-acyclic systems which have parameters such that the system is guaranteed stable if it is controlled by CLSA or CLW. Introducing regulators in the systems only worsens the performance for CLSA and CLW. However, regulators are necessary for non-acyclic systems which have parameters such that the system is unstable if it is controlled by CLSA or CLW. The results show that for these systems with regulators CLSA and CLW perform better than SAV^a. Therefore, it is not possible to point out one policy from SAV^a, CLSA, CLW, CLSA with regulators or CLW with regulators which has the best overall performance.

The Savkin policy is the only control policy which is guaranteed stable by itself for a non-acyclic system. However, SAV^a does not always have a good performance. Therefore, it has been explored if the Savkin policy could be improved in order to increase its performance. First, the performance of the original non-adapted Savkin policy was analyzed. It has been proven in this thesis for a manufacturing system with a continuous flow of work and no variability that the Savkin policy is not optimal with respect to minimizing the average flow time. Furthermore, it has been shown that the average flow time of the system is not independent of the initial buffer levels. Then, it has been tried to improve the Savkin policy such that for any initial buffer level the average flow time converges to the lowest value possible for that system. For a simple two buffer machine system with a continuous flow of work and no variability this succeeded. Finally, it has been proven for the two buffer single machine system controlled by the improved Savkin policy that for certain initial buffer levels the average flow time converges to its minimum in the smallest amount of time possible.

Samenvatting (in Dutch)

Moderne flexibel productiesystemen bestaan uit machines, buffers en transportlijnen. Sommige van de machines kunnen meerdere producttypes bewerken of verschillende bewerkingen uitvoeren op hetzelfde product. Voor deze machines is omstellen vaak noodzakelijk wanneer het volgende type product of bewerking verschilt van de vorige. Echter, elke keer dat er een omstelling plaats vindt gaat er productiecapaciteit verloren. Om die reden heeft de besturingsstrategie die bepaalt wanneer er omgesteld wordt een grote invloed op de prestatie van het productiesysteem.

Aan de hand van de productiepaden van de producten kan een flexibel productiesysteem worden onderverdeeld in twee groepen. Een flexibel productiesysteem wordt *acyclisch* genoemd wanneer de machines in het systeem geördend kunnen worden, zodat de aanwezige producten altijd van een machine lager in de ordening naar een machine hoger in de ordening stromen. Als deze ordering niet mogelijk is wordt het system *niet-acyclisch* genoemd. Een systeem dat slechts bestaat uit 1 machine, een enkel machine systeem, kan ook nog onderverdeeld worden aan de hand van de eigenschappen van de producten. Een enkel machine systeem wordt *symmetrisch* genoemd wanneer alle producten de zelfde aankomst-, bewerkings- en omsteltijd verdelingen hebben, zo niet dat wordt het systeem *asymmetrisch* genoemd.

Voor flexibel productiesystemen bestaan verscheidene besturingsstrategieën. Uit de verschillende besturingsstrategieën zijn drie veel belovende strategieën gekozen, namelijk de Savkin strategie (SAV) [Sav03], de Clear the Largest Work strategie (CLW) [Per89] en de Clear the Largest Scaled Age policy (CLSA) [Ols99]. Voor CLW is bewezen in [Liu92] dat deze strategie optimaal is voor het minimaliseren van de hoeveelheid (onvoltooide) producten in een symmetrisch stochastisch systeem met discrete producten. In [Ols99] is voor CLSA aangetoond door middel van simulaties dat deze strategie goed presteert ten opzichte van andere besturingsstrategieën en dat deze strategie functioneert in een stochastisch systeem met discrete producten. Een nadeel van beide strategieën is dat ze niet gegarandeerd stabiel zijn voor niet-acyclische systemen. Door gebruik te maken van een techniek genaamd regulatoren [Hum94] kan een instabiel niet-acyclisch systeem gestabiliseerd worden. SAV heeft deze regulatoren niet nodig, omdat deze strategie uit zichzelf al stabiel is voor niet-acyclische systemen. Een schaduwzijde van SAV is dat deze strategie alleen ontworpen is voor een systeem zonder variabiliteit of voor een systeem met een speciale beperkte variabiliteit. Bovendien, is SAV alleen ontworpen voor systemen met een continue stroom werk. Daarom, is er onderzocht of SAV kan functioneren in een stochastisch systeem met discrete producten. Het is gebleken dat in een dergelijk systeem dat SAV niet goed functioneert. Dit nadeel is gedeeltelijk weggewerkt door SAV aan te passen, zodat de strategie functioneert in een discreet systeem met een driehoeksverdeling voor de aankomst- en bewerkingstijden. De aangepaste Savkin strategie

wordt aangeduid met SAV^a.

De drie strategieën zijn met elkaar vergelijken voor een acyclisch enkel machine systeem en voor een complexer productie systeem met niet-acyclische productiepaden. De vergelijking is uitgevoerd door het simuleren van een discreet productiesysteem met een driehoeksverdeling voor de aankomst- en bewerkingstijden. Voor de prestatiewaardering is gebruik gemaakt van de gemiddelde doorlooptijden van alle producten in het systeem. Uit de resultaten blijkt dat CLSA beter presteert dan de andere strategieën voor acyclische enkel machine systemen. SAV^a presteert beter dan de andere strategieën voor niet-acyclische systemen met zulke parameters dat het systeem stabiel is als het bestuurt worden door CLW of CLSA. Het introduceren van regulatoren verslechtert alleen maar de prestaties van CLW en CLSA. Echter, regulatoren zijn wel noodzakelijk voor niet-acyclische systemen met parameters zodat het systeem instabiel is als het bestuurd wordt door CLW of CLSA. Het blijkt dat de prestaties van CLW en CLSA met regulatoren voor deze systemen beter zijn dan die van SAV^a. Uit SAV^a, CLW, CLSA, CLW met regulatoren, CLSA met regulatoren kan dus geen strategie worden aangewezen die altijd de beste prestatie levert.

De Savkin strategie is de enige besturingsstrategie welke gegarandeerd stabiel is van zichzelf voor een niet-acyclisch systeem. Echter, SAV^a presteert niet altijd even goed in vergelijking met de andere besturingsstrategieën. Daarom is er onderzocht of het mogelijk is de prestatie van de strategie te vergroten door de Savkin strategie te verbeteren. Daarvoor werd eerst de prestatie van de originele niet aangepaste Savkin strategie bestudeerd. Er is bewezen in dit verslag voor een productiesysteem met een continue stroom werk en geen variabiliteit dat de Savkin strategie niet optimaal presteert met betrekking tot het minimaliseren van de gemiddelde doorlooptijd. Verder is er aangetoond dat de gemiddelde doorlooptijd van het systeem niet onafhankelijk is van de initiële buffer niveaus. Daarna, is er getracht de Savkin strategie te verbeteren zodat onafhankelijk van de initiële buffer niveaus de gemiddelde doorlooptijd altijd convergeert naar de laagste waarde mogelijk voor dat systeem. Voor een acyclisch enkel machine systeem met 2 buffers en en continue stroom van werk en geen variabiliteit is dit gelukt. Tenslotte, is er bewezen voor het 2 buffers enkel machine system bestuurd door de verbeterde Savkin strategie dat voor bepaalde initiële buffer niveaus de gemiddelde doorlooptijd in de kortst mogelijk tijd naar de laagst mogelijk waarde convergeert.

List of Acronyms and Symbols

Acronyms

CLSA	Clear the Largest Scaled Age control policy as introduced in [Ols99]
CLW	Clear the Largest Work control policy as introduced in [Per89]
CLW$^{\mathrm{a}}$	Slightly modified version of CLW as introduced in this report, when there is a tie between buffers, it is broken by choosing the buffer with the largest scaled age
SAV	Control policy as introduced by Savkin in [Sav03]
SAV$^{\mathrm{a}}$	Modified version of SAV as introduced in this report, this version is stable for flexible manufacturing systems with a triangular distribution for the process and arrival rate and with discrete parts.

Symbols

$A_{p,i}(t)$	Total age of all the work present in buffer $b_{p,i}$ at time t
$\hat{A}_{p,i}(t)$	Total expected age after a setup of all the work present in buffer $b_{p,i}$
$A^a_{p,i}(s,t)$	Cumulative input of buffer $b_{p,i}$ in the period $[s,t]$
$A^r_{p,i}(s,t)$	Cumulative output from buffer $b_{p,i}$ in the period $[s,t]$
$b_{p,i}$	Buffer in front of machine $\alpha_{p,i}$ containing work of type p
B_m	Set containing all buffers at machine m
d_p	Desired production level of part type p
D	Vector containing the lengths of all production runs from all buffers $b_{p,1}$
D_p^k	Length of a production run from buffer $b_{p,1}$ during cycle k of the improved Savkin policy for a two buffer single machine system
$f_{p,i}^j$	Time at which work of type p starts arriving in buffer $b_{p,i}$ during a scheduling period j
$F_{p,i}^j$	Time at which the input of work of type p at buffer $b_{p,i}$ ends after it started arriving at $f_{p,i}^j$
i	i^{th} step of a production path of a part type
$\mathcal{I}_p$	Set containing all the production step numbers of a part type p
j	j^{th} scheduling period T

$J_{p,i}$	Number of scheduling periods T necessary to guarantee that the cumulative input of buffer $b_{p,i}$ equals the desired production level of part type p
k_m	Number of buffers at machine m
$l_{p,i}$	Average transportation delay when a part of type p moves from machine $\alpha_{p,i}$ to machine $\alpha_{p,i+1}$
$l^c_{p,i}$	Constant transportation delay when a part of type p moves from machine $\alpha_{p,i}$ to machine $\alpha_{p,i+1}$
m	Machine number
M	Number of machines in a flexible manufacturing system
$\mathcal{M}$	Set containing all machines
n_p	Last step of a production path of a part of type p
p	Part type
P	Number of part types in a flexible manufacturing system
$\mathcal{P}$	Set containing all part types
$q(t)$	Vector containing the machine states of all the machines present in a flexible manufacturing system at time t
$q_m(t)$	State of machine m at time t
$\mathfrak{R}$	Area in which all positions lay that have a distance ϵ or less from $x^{\mathrm{id}}(t^{\mathrm{id}}_{1,1})$
$t^j_{p,i}$	Time at which machine $\alpha_{p,i}$ starts a production run from buffer $b_{p,i}$ during scheduling period j
$t^{\mathrm{id}}_{p,i}$	Time at which machine $\alpha_{p,i}$ under control of a Savkin policy, which follows the ideal Savkin trajectory, starts a production run from buffer $b_{p,i}$
T	The length of a scheduling period for the Savkin policy
T_0	Minimal scheduling period as computed by the Savkin control policy for a flexible manufacturing system with the production levels $d_1, d_2, \ldots, d_P$
$T^{\mathrm{id}}_{p,i}$	Time at which machine $\alpha_{p,i}$ under control of a Savkin policy, which follows the ideal Savkin trajectory, ends a production run from buffer $b_{p,i}$
$T^j_{p,i}$	Time at which a production run by machine $\alpha_{p,i}$ from buffer $b_{p,i}$ during scheduling period j ends
T^k	Length of one cycle of the improved Savkin policy for a two buffer single machine system
$\hat{T}^k$	Time after k cycles of the improved Savkin policy for a two buffer single machine system
$u_{p,i}(t)$	Regulator speed
$\hat{u}^c_{p,i}(t)$	Regulator speed constant
u_m	Average utilization of machine m
u^s_m	Utilization of machine m under control of the Savkin policy
$w_{p,i}$	Scaling factor for the Clear Largest Scaled Age policy

$x(t)$	Vector containing the buffer levels of all the buffers present in a flexible manufacturing system at time t
x_0	Vector containing the buffer levels of all the buffers present in the flexible manufacturing system at time 0
$x^{\mathrm{id}}(t)$	Vector containing the buffer levels at time t of a two buffer single machine system under control of a Savkin policy, which follows the ideal Savkin trajectory
$x_{p,i}(t)$	Level of buffer $b_{p,i}$ at time t
$x^{\mathrm{id}}_{p,i}(t)$	Level of buffer $b_{p,i}$ at time t of a two buffer single machine system under control of a Savkin policy, which follows the ideal Savkin trajectory
$y(t)$	Vector containing the cumulative outputs from buffer b_{p,n_p} in the period $[0,t]$ of all the part types present in a flexible manufacturing system.
$y_p(t)$	Cumulative output of part type p from buffer b_{p,n_p} in the period $[0,t]$
$y_{p,i}(t)$	Cumulative output of part type p from buffer $b_{p,i}$ in the period $[0,t]$
$\alpha_{p,i}$	Machine at which the i^{th} step of a production path of part type p takes place
θ_m	Time required to perform a setup at machine m if all setups require the same amount of time
θ^s_m	Time required to perform a setup at machine m as computed by the Savkin policy
$\theta_{1\mapsto 2}$	Time required to perform a setup when the machine of a two buffer single machine system switches from buffer $b_{1,1}$ to buffer $b_{2,1}$
$\theta_{2\mapsto 1}$	Time required to perform a setup when the machine of a two buffer single machine system switches from buffer $b_{2,1}$ to buffer $b_{1,1}$
$\theta^{\mathrm{e}}_{p,1}(k)$	Amount of time that the setup for buffer $b_{p,1}$ is extended in cycle k for a two buffer single machine system
$\theta^{\mathrm{t}}_{p,1}(k)$	Total length of a setup with extension for buffer $b_{p,1}$ in cycle k for a two buffer single machine system
Θ^s_m	Minimal time required to perform a setup at machine m as computed by the Savkin policy
λ_p	Average arrival rate of parts of type p at machine $\alpha_{p,1}$
$\lambda_p(t)$	Arrival rate of parts of type p at machine $\alpha_{p,1}$ at time t
λ^c_p	Constant arrival rate of a parts of type p at at machine $\alpha_{p,1}$
$\lambda_{p,i}(t)$	Arrival rate of parts of type p at buffer $b_{p,i}$ at time t
$\lambda'_{p,i}$	Burst arrival rate of a parts of type p at machine $\alpha_{p,i}$
$\lambda''_{p,i}$	Conditional burst arrival rate arrival rate of a parts of type p at machine $\alpha_{p,i}$
$\mu_{p,i}$	Average rate at which parts of type p are processed by machine $\alpha_{p,i}$
$\mu^c_{p,i}$	Constant rate at which parts of type p are processed by machine $\alpha_{p,i}$
ρ_m	Average total load factor of machine m
$\rho_{p,i}$	Average load factor of a part type p on machine $\alpha_{p,i}$

ρ''_m	Conditional burst congestion level of machine m
$\tau^l_m(t)$	Earliest time instance starting from time t at which at least one of the buffers of machine m is not empty
$\tau_m[q_m(\cdot)\|^t_0](t)$	Time at which machine m started to work from the buffer from which machine m is currently processing work at time t

Contents

Chapter 1

Introduction

1.1 Scope of the assignment

The modern flexible manufacturing system consist of a number of machines, buffers and transportation lines. Some of the machines are capable of processing multiple types of parts or performing multiple operations on the same part. If the next part type or operation is different from the previous part type or operation a setup is often required for these machines. For example, car manufacturers run stamping plants where the setup between part types involves changing dies. The policy used for switching from processing a part type to a different part type has a large influence on the performance of the manufacturing system. For example, consider a machine which has to process two different parts and thus has two buffers in front of the machine. Switching to the other buffer every time a part has been processed results in loss of production time. However, switching to a different buffer only after a long period processing the same part results in large buffer levels and longer waiting times of the parts in the buffers. Therefore, it is important to determine a switching policy for the flexibele manufacturing systems that guarantees a certain objective. For example, to minimize the time required to achieve a desired production level or minimize the average flow time.

A lot of tools exist that try to find an optimal schedule for processing jobs that are all present at initial time. However, in a modern production environment jobs arrive continuously over time. This continuously arrival of jobs is highly relevant in deciding which part to process next. The focus of this report is on the control of these modern production environment. For these systems various control policies exist. Some of these control policies have been developed by approximating manufacturing systems with differential logical equations. The goal of this project is to investigate how and to what extend these approximation models and their control policies can be utilized to control real manufacturing systems.

Most of the various control policies that exist have been developed and proven to be stable for an acyclic manufacturing system with no variability present. However, a manufacturing system without variability is not realistic. Therefore, the influence of variability needs to be investigated. In this report the influence of variability is investigated with the use of discrete event simulation, because analytical results are not yet available. It can be shown that most of the policies developed are unstable for certain conditions when applied to a non-acyclic system. A number of techniques to stabilize these systems have been proposed

in literature. Just recently a switching policy was introduced which is stable without such techniques [Sav03], but this policy is only suitable for deterministic systems. The questions that this reports tries to answer can be summarized as follows:

- Which control policies are available for controlling flexible manufacturing systems?
- Are these policies suitable for stochastic environments and, if not, can they be adapted?
- For acyclic systems, how do these policies perform and what is the influence of certain characteristics of the system on the performance?
- For non-acyclic systems, which techniques are available for stabilizing unstable control policies and how do they perform compared to the just recently developed stable switching policy [Sav03]?
- When the previous questions are answered, is it possible to point out one policy with the best performance and if not, is it possible to improve the most promising policy in order to increase its performance?

In this report the average flow time of all parts is used as a performance measure for a control policy.

1.2 Outline

The outline of this report is as follows. The next chapter introduces the notation used to describe a flexible manufacturing system in this report. Furthermore, the results of a literature survey on the subject of controlling such lines are given. In Chapter 3, three control policies, chosen from the literature survey, are explained further. Then, it is investigated if and how the policies need to be adapted to function in a stochastic discrete environment. In Chapter 4 simulation experiments are setup and performed in order to compare the three policies with each other for a simple single machine system with acyclic production paths. After that the policies are compared for a more complicated manufacturing system containing non-acyclic production paths. Furthermore, instability problems that may arise are studied and it is explored what the effect is of equipping (if necessary) the policies with an existing stabilizing tool. Finally, it is investigated in Chapter 6 if it is possible to improve the most promising policy.

Chapter 2

Flexible manufacturing systems

2.1 Introduction

This chapter starts with an introduction on the different characteristics of a flexible manufacturing system. Furthermore, this chapter introduces a continuous approximation of a flexible manufacturing system and various definitions are presented. A flexible manufacturing system is controlled by a control policy. Therefore, the general properties of a control policy are introduced. Finally, the results of a literature survey on the subject of control policies are discussed.

2.1.1 Flexible manufacturing system characteristics

The flexible manufacturing systems described in this report have the following features:

1. There are P part-types which belong to the set $\mathcal{P} = \{1, 2, \ldots, P\}$ and there is a set $\mathcal{M} = \{1, 2, \ldots, M\}$ of machines.

2. Parts of type p require processing at the machines $\alpha_{p,1}, \alpha_{p,2}, \ldots, \alpha_{p,n_p}$ in that order, where $\alpha_{p,i} \in \mathcal{M}$ and i belongs to the set $\mathcal{I}_p = 1, 2, \ldots, n_p$.

3. Raw parts of type p arrive to the system at machine $\alpha_{p,1}$ with an average rate of $\lambda_p > 0$.

4. At the i^{th} machine they visit, parts of type p enter the buffer labelled $b_{p,i}$, from which they are eventually processed by this machine at an average rate of $\mu_{p,i} > 0$.

5. Parts of type p incur an average transportation delay $l_{p,i} \geq 0$ when moving from machine $\alpha_{p,i}$ to machine $\alpha_{p,i+1}$.

6. The machine m serves the buffers $B_m := \{b_{p,i} : \alpha_{p,i} = m\}$. A minimal setup time $\theta_m \geq 0$ is required when machine m switches from processing parts of type p present in buffer $b \in B_m$ to processing parts of another type p' present in buffer $b' \in B_m$. The machine does not work during such a setup.

Until stated otherwise, it is assumed that the transportation delay, arrival and process rate are stochastic variables with an average of respectively $l_{p,i}, \lambda_p$ and $\mu_{p,i}$. It will be explicitly stated when these variables are assumed to have constant values. A constant transportation delay, arrival or process rate is denoted with $l^c_{p,i}, \lambda^c_p$ and $\mu^c_{p,i}$ respectively. The minimal setup time θ_m is assumed constant throughout this report. Therefore, the '^{c}' is left out for simplicity.

An example of a flexible manufacturing system is given in Example 2.1 to illustrate the above description.

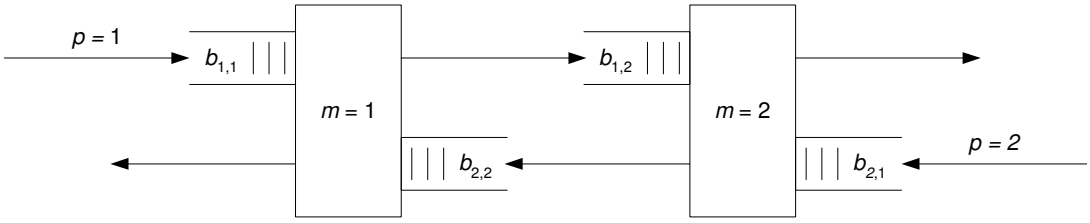

Figure 2.1: The flexible manufacturing system of Example 2.1.

Example 2.1. *Consider the flexible manufacturing system depicted in Figure 2.1 and described as:*

$$\begin{aligned} \mathcal{P} &= \{1,2\} & \mathcal{M} &= \{1,2\} \\ \alpha_{1,1} &= 1 & \alpha_{1,2} &= 2 \\ \alpha_{2,1} &= 2 & \alpha_{2,2} &= 1 \\ \lambda^c_1 &= 1 & \lambda^c_2 &= 2 \\ \mu^c_{1,1} &= 2 & \mu^c_{1,2} &= 3 \\ \mu^c_{2,1} &= 2 & \mu^c_{2,2} &= 4 \\ l^c_{1,1} &= 0.1 & l^c_{2,1} &= 0.1 \\ \theta_1 &= \frac{2}{3} & \theta_2 &= 1 \end{aligned}$$

The manufacturing system consists of two parts $\mathcal{P} = \{1,2\}$ and two machines $\mathcal{M} = \{1,2\}$. The arrival rates of the parts are respectively $\lambda^c_1 = 1$ and $\lambda^c_2 = 2$. The first step for processing part 1 is on machine one ($\alpha_{1,1} = 1$), with a process rate of 2 ($\mu^c_{1,1} = 2$). The second step is on machine two ($\alpha_{1,2} = 2$) with a rate 3 ($\mu^c_{1,2} = 3$). Part 2 is first processed on machine two ($\alpha_{2,1} = 2$) with a rate of 2 ($\mu^c_{2,1} = 2$) and after that on machine one ($\alpha_{2,2} = 1$) with a rate of 4 ($\mu^c_{2,2} = 4$). The time required to transport part 1 from machine one to two and part 2 from machine two to one equals 0.1 ($l^c_{i,1} = 0.1$). The time required to perform a setup on machine one equals $\frac{2}{3}$ ($\theta_1 = \frac{2}{3}$) and for machine two equals 1 ($\theta_2 = 1$).

In addition to the various system parameters provided above the following parameters are used to describe a flexible manufacturing system. Let $x_{p,i}(t)$ denote the level of buffer $b_{p,i}$ at time t and for any $1 \leq m \leq M$ and let $q_m(t)$ denote the state of machine m. Furthermore, let $B_m := \{b_{p,i} : \alpha_{p,i} = m\}$ be the set of the buffers served by machine m, and introduce the set of symbols $\hat{B}_m := B_m \cup \{0\}$. The function $q_m(t) \in \hat{B}_m$ is defined for all times t as follows:

$$q_m(t) := \begin{cases} 0, & \text{if machine } m \text{ does not work at time } t; \\ b_{p,i}, & \text{if machine } m \text{ works with buffer } b_{p,i} \text{ at time } t. \end{cases}$$

Moreover, let $y_p(t)$ denote the cumulative output of part-type p from buffer b_{p,n_p} over $[0,t]$.

For a flexible manufacturing system the following physically natural initial conditions are considered:

$$x(t) = x_0 \qquad y(t) = 0 \qquad q(t) = 0 \qquad \forall t < 0, \tag{2.1}$$

where x_0 is a given vector with non-negative components. Furthermore, the control policy in use should always make sure that the following condition holds:

$$x_{p,i}(t) \geq 0 \qquad \forall t \geq 0,\ \forall i \in \mathcal{I},\ \forall p \in \mathcal{P}, \tag{2.2}$$

since a buffer level physically cannot become negative.

The state of buffer $b_{p,1}$ in a system without variability can now be described by the following logical-differential equations [Sav98]:

$$\begin{aligned}
&\textbf{if } q_{\alpha_{p,1}}(t) = b_{p,1} \\
&\qquad \textbf{then } \dot{x}_{p,1}(t) = \lambda_p^c - \mu_{p,1}^c, \\
&\textbf{if } q_{\alpha_{p,1}}(t) \neq b_{p,1} \\
&\qquad \textbf{then } \dot{x}_{p,1}(t) = \lambda_p^c.
\end{aligned} \tag{2.3}$$

These equations are a continuous approximation of the discrete flow of parts. The contents of the buffers can been seen as work. It is convenient to see the work as a fluid and the buffers as tanks.

Furthermore, for $2 \leq i \leq n_p$ and $1 \leq p \leq P$ the level of a buffer can be described with:

$$\begin{aligned}
&\textbf{if } \left(q_{\alpha_{p,i}}(t) = b_{p,i} \textbf{ and } q_{\alpha_{p,i-1}}(t - l_{p,i}^c) = b_{p,i-1}\right) \\
&\qquad \textbf{then } \dot{x}_{p,i}(t) = \mu_{p,i-1}^c - \mu_{p,i}^c, \\
&\textbf{if } \left(q_{\alpha_{p,i}}(t) = b_{p,i} \textbf{ and } q_{\alpha_{p,i-1}}(t - l_{p,i}^c) \neq b_{p,i-1}\right) \\
&\qquad \textbf{then } \dot{x}_{p,i}(t) = -\mu_{p,i}^c, \\
&\textbf{if } \left(q_{\alpha_{p,i}}(t) \neq b_{p,i} \textbf{ and } q_{\alpha_{p,i-1}}(t - l_{p,i}^c) = b_{p,i-1}\right) \\
&\qquad \textbf{then } \dot{x}_{p,i}(t) = \mu_{p,i-1}^c, \\
&\textbf{if } \left(q_{\alpha_{p,i}}(t) \neq b_{p,i} \textbf{ and } q_{\alpha_{p,i-1}}(t - l_{p,i}^c) \neq b_{p,i-1}\right) \\
&\qquad \textbf{then } \dot{x}_{p,i}(t) = 0.
\end{aligned} \tag{2.4}$$

If $y_{p,i}(t)$ denotes the cumulative output of part-type p from buffer $b_{p,i}$ for $1 \leq i \leq n_p$ and $1 \leq p \leq P$ over the time interval $[0,t]$, then $y_{p,i}(t)$ is described by the initial condition $y_{p,i}(0) = 0$ and:

$$\begin{aligned}
&\textbf{if } q_{\alpha_{p,i}}(t) = b_{p,i} \\
&\qquad \textbf{then } \dot{y}_{p,i}(t) = \mu_{p,i}^c, \\
&\textbf{if } q_{\alpha_{p,i}}(t) \neq b_{p,i} \\
&\qquad \textbf{then } \dot{y}_{p,i}(t) = 0.
\end{aligned} \tag{2.5}$$

The above mentioned continuous approximation of a flexible manufacturing system belongs to the class of hybrid dynamical systems. A system is said to be *hybrid dynamical* [Sav98] if it combines continuous and discrete behavior and involves, thereby, both continuous and discrete state variables. For example, in the above approximation the buffer levels $x_{p,i}(t)$ are continuous and the machine states $q_m(t)$ are discrete.

For the earlier described flexible manufacturing system the following definitions are presented.

Definition 2.1. [Hop96] The *utilization* of a machine is the fraction of time it is not idle for lack of parts. This includes the fraction of time the machine is working on parts or has parts waiting and is unable to work on them due to machine failure, setup or other detractors. The utilization is computed as:

$$\text{utilization} = \frac{\text{arrival rate}}{\text{effective processing rate}}, \tag{2.6}$$

where the effective processing rate is defined as the maximum average rate at which the workstation can process parts, considering the effects of failures, setups and all other detractors.

In the manufacturing system described above the only detractor present is setup and possible variability. However, the effect of the setup time on the maximum average rate at which the workstation can process parts is difficult to determine. Thus, making it hard to compute the utilization. Therefore, the following definition is introduced.

Definition 2.2. The *average total load factor* ρ_m of a machine m ($m \in \mathcal{M}$) is the average fraction of time that machine m is processing parts and is defined as:

$$\rho_m = \sum_{b_{p,i} \in B_m} \frac{\lambda_p}{\mu_{p,i}}. \tag{2.7}$$

Note that the average load factor of a part type p ($p \in \mathcal{P}$) on machine $\alpha_{p,i}$ ($i \in \mathcal{I}_p$) equals:

$$\rho_{p,i} = \frac{\lambda_p}{\mu_{p,i}}. \tag{2.8}$$

In other words, the average total load factor is the necessary fraction of time that machine m must work on average to ensure stability. Therefore, for a manufacturing system with a non-zero minimal setup time θ_m a necessary *capacity condition* for stability of any control policy is:

$$\rho_m < 1 \qquad \forall m \in \mathcal{M}. \tag{2.9}$$

Note that for the average utilization u_m and the average total load factor ρ_m the following is always true:

$$\rho_m \leq u_m \qquad \forall m \in \mathcal{M}. \tag{2.10}$$

For the above described system the average total load factor equals the average utilization if the minimal setup time θ_m equals zero.

For a system with no variability and with a setup time of zero the capacity condition (2.9) can be *relaxed* to:

$$\rho_m = u_m \leq 1 \qquad \forall m \in \mathcal{M}. \tag{2.11}$$

However, in a stochastic environment the utilization has to be smaller than 1, since a utilization of 1 results in an unstable system. The average total load factor equals the utilization if the minimal setup time θ_m equals zero. Therefore, for a system with variability capacity condition (2.9) should be satisfied.

Flexible manufacturing systems are distinguished into two different classes with regard to the production paths of the parts.

Definition 2.3. [Per89] A flexible manufacturing system is called *acyclic* if the machines can be ordered so that parts can only move from one machine to a machine higher in the ordering. The system is called *non-acyclic* if such an ordering is not possible.

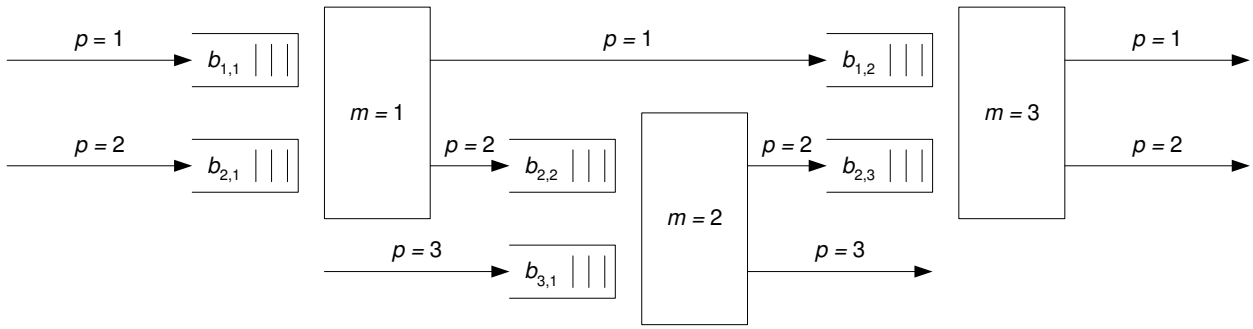

Figure 2.2: An acyclic manufacturing system.

An example of an acyclic manufacturing system is depicted in Figure 2.2. The figure shows a flexible manufacturing system consisting of 3 machines and 3 different part types. Part type one is first processed on machine one and then on machine three. The production path of part type two is: machine one, machine two and finally machine three. Parts of type 3 need only to be processed on machine two. All the parts flow from a machine lower in the ordering to a machine higher in the ordering. The ordering of the machines is $1 \prec 2 \prec 3$. A system where parts visit a machine more than once is called *re-entrant*. An example of a re-entrant system is shown in Figure 2.3. The figure shows a flexible manufacturing system consisting of two machines and only one part type. The production path of this part type is: machine one, machine two, machine two and finally machine one again. Thus, parts visit machine one and machine two more than once. The system of Figure 2.3 is also non-acyclic, since no ordering exists between the machines. Note that a non-acyclic system differs from a re-entrant system. A system which is not re-entrant can still be non-acyclic. An example of a non-acyclic system which has no re-entrant production paths is given in Example 2.1. A part visits each machine only once, thus the system is not re-entrant. However, the system is non-acyclic, since the machines cannot be ordered.

For a production system that only consists of one machine, a single machine system, a distinction is also made on basis of the part type properties.

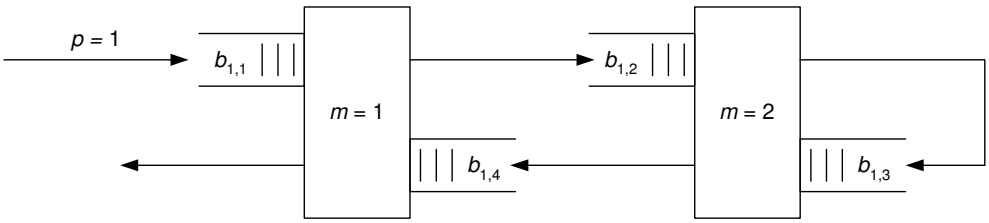

Figure 2.3: A non-acyclic manufacturing system.

Definition 2.4. [Ols99] A single machine system is said to be *symmetric* if all part types have identical arrival rate, setup time and process rate distributions, otherwise the system is said to be *asymmetric*.

In order to study deterministic multi machine systems it is useful to consider the following class of inputs.

Definition 2.5. [Per94] Inputs belong to the class of *bounded burstiness* inputs if there exists a constant γ_p such that:

$$\int_s^t \lambda_p(t) \cdot dt \leq \lambda_p^c \cdot (t-s) + \gamma_p \quad \forall t \geq s \geq 0. \tag{2.12}$$

In which $\int_s^t \lambda_p(t) \cdot dt$ denotes the cumulative input of parts of type p ($p \in \mathcal{P}$) into the system.

The class of bounded burstiness inputs is one of the few classes for which bounds on system parameters, such as buffer levels, can be determined.

2.1.2 Control policy characteristics

The characteristics of a flexible manufacturing system have now been described. A control policy is required, which controls such a manufacturing system. The general properties of a control policy are introduced in this section.

Definition 2.6. For a flexible manufacturing system a control policy is said to be *stable* if for every initial condition, there exists a finite constant $L_{b,i}$, such that for every (p, i):

$$x_{p,i}(t) \leq L_{p,i} \leq \infty \qquad \forall t \geq 0,\ \forall p \in \mathcal{P},\ \forall i \in \mathcal{I}_p. \tag{2.13}$$

Thus, a control policy is said to be stable if every buffer in the flexible manufacturing system is bounded. Note that a control policy can only be stable provided that the capacity condition (2.9) holds for the flexible manufacturing system.

Definition 2.7. [Per94] A control policy for a single machine system is said to be *stable in isolation (SI)*, if it stabilizes the single machine under bounded burstiness inputs.

Definition 2.8. [Per89] A control policy is said to be *distributed* if the policy makes decisions, regarding scheduling of parts to be worked on at machine m, by only observing the buffer levels at machine m.

Definition 2.9. [Per94] A distributed control policy for a multi machine system is said to be *distributed stable in isolation (DSI)* if every machine implements a stable in isolation control policy.

The control policies can be divided in three different service disciplines [Tak86].

1. *Exhaustive service*: the machine continues to process from a buffer until the buffer is emptied. Parts arriving at the buffer currently being emptied are also processed.
2. *Gated service*: the machine processes only those parts which are waiting in the buffer at the moment the machine started the setup for that buffer.
3. *Limited service*: a buffer is served until either:
   (a) The buffer is emptied.
   (b) A specified number of parts are processed.

Furthermore, a control policy can have the following properties [Ols99].

- *Queue greedy*: the machine never idles at a buffer while there are parts in the buffer.
- *System greedy*: the machine never idles while there are parts in one of its buffers.
- *Patient*: if there are no more parts in the buffer then the machine idles at the last buffer it processed from.
- *FIFO service*: The parts in a buffer are processed in order in which they arrived (i.e. First In First Out).

The in this report needed characteristics of a flexible manufacturing system and of a control policy, have now been described. In the next section it is investigated by means of a literature survey, which control policies already exist.

2.2 Literature survey

A control policy is required in order to control the flexible manufacturing system that has been introduced in the previous section. In this section the results of a literature survey on control policies are discussed. In the survey it is assumed, until stated otherwise, that the time required to perform a setup is non-zero. Obviously, the necessary capacity condition (2.9) should be satisfied.

In literature a lot of information can be found that deals with *static* models. These static models try to solve the problem of scheduling a fixed number of parts or jobs, with known processing requirements, on a given set of machines, in order to minimize or maximize certain performance measures like the average flow time. However, the solution procedure grows increasingly more complex with an increasing number of parts or scheduling horizon. Also in a modern production environment jobs arrive continuously over time and thus the number of parts are not known at initial time. Therefore a more *dynamical* approached is preferred.

Unfortunately, the greater part of literature deals with static scheduling and not with dynamic scheduling. Most literature on dynamic scheduling follows the approach as set by [Ger86] and [Tak86]. The former states that a setup should not be performed too often because of the resulting reduction of capacity and a setup should not be performed infrequently, because of the resulting increase in flow time and buffer levels. The latter introduces among other things the three service disciplines, exhaustive, gated and limited, as introduced in the previous section.

In [Per89] different distributed control policies are introduced, such as *Clear the Largest Buffer (CLB)* and *Clear the Largest Work (CLW)*. These policies are all exhaustive, patient, queue and system greedy and have a FIFO service discipline. The policies mentioned in [Per89] all belong to a more general class of policies, namely the *Clear a Fraction (CAF)* control policies. The control policies are proven stable for a single machine in isolation in a deterministic environment. In addition, for an acyclic system it is proven that a CAF policy will stabilize a multi machine system (in a deterministic environment) for all initial conditions. In [Per89] it is still an open question if a CAF policy will stabilize a non-acyclic system. However, a modified CAF *backoff* policy is provided, which will stabilize a non-acyclic system for all initial conditions. This modified policy processes a part type p when it would have processed the same part type if the machine had been in isolation. In other words, the policy uses part type ordered time slices, with a length determined by the working of each machine as though it is in isolation.

In [Kum90] it is shown that a CAF policy does not stabilize a non-acyclic manufacturing system. It turns out that bounded levels of the buffers is not a property independent of the initial conditions. Even a setup time equal to zero can result in instability. Furthermore, [Kum90] introduces an *Universally Stabilizing Supervisory Mechanism (USSM)* that can be used by a supervisor to stabilize any control policy. The USSM

- truncates all long enough production runs, and
- maintains a FIFO (First In First Out) list of all buffers with a large number of parts.

Though this mechanism is called supervisory it can be implemented in a distributed way. Note the difference between the two stabilizing policies introduced by [Per89] and [Kum90]. The policy of [Per89] (backoff) is used to implement a CAF policy at each machine as though it were operating in an isolated mode, but the policy of [Kum90] (USSM) can be used as a "safety net" for *any* policy. The stability of the CAF policy with backoff is achieved at the price of possible enforced idleness and thus possible unnecessarily large buffer levels. The USSM ensures stability without forced idleness, though some time is lost due to repeated switchings [Per94].

In [Per94] deterministic manufacturing environments with bounded burstiness inputs have been analyzed. It is shown that distributed stable in isolation (DSI) policies are stable for acyclic systems with bounded burstiness inputs. It is also shown that the CAF policies belong to the class of stable in isolation (SI) policies and to the class of distributed stable in isolation (DSI) policies. Furthermore, a third approach is provided to stabilize a non-acyclic system, which implements system elements called *regulators*. The idea of using regulators is due to [Hum94]. However, [Per94] introduces an improved version. It is proven that if the regulators satisfy certain conditions, then the system will be stable for all DSI policies with bounded

burstiness inputs. In [Hum94] it is also shown that CAF policies with backoff and USSM can both be treated as particular cases of totally regulated systems. Note, that all provided proofs of stability are only valid for deterministic systems or systems with inputs containing bounded burstiness.

In [Liu92] it is shown for a symmetric single machine system with variability and discrete parts that a *Stochastically Largest Queue (SLQ)* policy stochastically minimizes the (unfinished) work in the system. If all buffer levels at the machine are known, a SLQ policy switches to a queue known to have the largest buffer level. Note that a Clear the Largest Buffer policy (CLB) belongs to the class of SLQ policies and in a symmetric system so does Clear the Largest Work (CLW). Furthermore, [Liu92] shows that queue greedy, exhaustive, FIFO service policies stochastically minimize the (unfinished) work in a single machine system. Hence, for symmetric systems, queue greedy exhaustive policies also minimize the expected waiting time in a buffer [Ols99]. Finally [Liu92] shows for symmetric systems that patient policies minimize the expected waiting time.

In [Coo98] en [Coo99] it is discussed that a non idling policy does not have to perform better than an idling policy. In [Ols99] a new heuristic policy is introduced, which in this report is called *Clear the Largest Scaled Age (CLSA)*, and it uses the scaled age of the parts in each queue, as well as queue statistics, to decide which queue to service next. Like the CAF policies this policy is exhaustive, patient, queue and system greedy and has a FIFO service discipline. This policy is compared with other (heuristic) policies, by simulating the policies for a single machine in isolation in a stochastic environment. It turns out that the proposed policy has a good performance with respect to the average waiting time in the buffers, the waiting time variance and the confidence interval. However, a drawback is that this policy only works good with significant large setup times (it does not work with a setup equal to zero).

In [Lu91] several control policies are analyzed for re-entrant manufacturing systems with a setup time θ_m equal to zero. It is shown for a single part manufacturing system, that the *First Buffer First Serve (FBFS)*, *Last Buffer First Serve (LBFS)*, *Earliest Due Date (EDD)* and *Least Slack (LS)* policies are stable even with inputs containing bounded burstiness. Simulation tests are performed and it seems that LBFS may wel be the best policy for minimizing the flow time, while LS may be the best policy for minimizing the variance of the delay. For a multiple part situation the *uniform* LBFS and *Interleave* FBFS policies are stable. Open questions in [Lu91] are the stability of the *First Comes First Serve (FCFS/FIFO)* policy for single and multiple part systems and the stability of the LS and EDD policy in a multiple part system and the stability of the LS and EDD policy in a multiple part system. In [Sei94] it is shown that FIFO/FCFS can be unstable. In [Ban97] it is demonstrated by *simulation* again with a setup time θ_m equal to zero that the FIFO/FCFS, *Shortest Mean Processing Time First* and *Shortest Remaining Processing Time First* policies can be unstable. Also applying LBFS at station 1 and FBFS at station 2 can result in instability. Furthermore, simulation is used to calculate feasible regions of stability.

In [Sav98] the concept of regularizability for a complex switched server queueing network with non zero setup times is introduced. A switched server queueing network, with bounded time-varying arrival rates, is called regularizable if there exist a control policy such that the following two conditions hold:

1. All the trajectories of the closed-loop system are bounded

2. In case of constant average arrival rates, the relevant dynamics of the closed-loop system exhibits a regular eventually periodic behavior.

For regularizable networks the paper introduces a control policy which guarantees boundedness and a regular predictable behavior of all the trajectories (acyclic and non-acyclic) of the network. Note, that there is a large similarity between inputs containing bounded burstiness and these bounded time-varying arrival rates. This policy is non-greedy and impatient. Furthermore, the policy is a variation on the limited service discipline. A modified version of the control policy is introduced in [Sav03].

The difference between the policies is the focus of [Sav03] on a manufacturing environment and in [Sav98] on a general complex switched server queueing network.

2.3 Resumé

The CAF policies are stable for acyclic systems or for non-acyclic systems with certain stabilizing techniques, all for systems in deterministic environment with inputs containing bounded burstiness. The CLSA heuristic policy has been shown to be stable with the use of simulation for a single machine in a stochastic environment. In [Liu92] it is shown that SLQ policies minimize the unfinished work and the number of parts in a stochastic symmetric single machine system. The Savkin policy is stable for all deterministic systems and for systems with bounded variability in the arrival rates. The policies as discussed in [Lu91] are stable for deterministic non-acyclic systems without setup.

All proven stability is based on hybrid dynamical systems. However, in a realistic manufacturing environment, variability, non-zero setup times, non continuous flow of parts, asymmetric queues and more than one machine are often present, thus none of the policies is proven or shown to work in such environment. In the following chapters it is investigated what the influences are of stochasticity, discretization, asymmetry or multiple machines on the performances of the different policies. The following policies are compared with each other.

- Savkin policy (SAV) [Sav98], [Sav03], because this policy is already proven to be stable for non-acyclic systems.
- Clear the Largest Work policy (CLW) [Per89], which is a special case of the CAF policy, because this policy is optimal for the amount of parts and unfinished work in a symmetric system.
- Clear the Largest Scaled Age (CLSA) [Ols99], because it has been shown that this policy performs well compared to other policies.

The policies as discussed in [Lu91] are not used for comparison, because these policies are not proven stable for systems with a setup time θ_m not equal to zero.

Chapter 3

Control policies

3.1 Introduction

The previous chapter introduced a flexible manufacturing system, which can produce different types of parts on the same machine. A certain amount of time is required in order to change the production from one part type to a different part type. Furthermore, general characteristics of a control policy, which can control such a manufacturing system, have been introduced. Finally, the results of a literature survey on different control policies have been given and from that survey three policies were chosen for comparison. In this chapter these three policies, which are the Savkin policy (SAV), the Clear the Largest Work (CLW) policy and the Clear the Largest Scaled Age policy (CLSA), are are explained in more detail. SAV and CLW are only designed for systems without variability or for systems with a special limited variability. Furthermore, the work in the systems can only be continuous. However, in a realistic manufacturing system variability is present and the work consists of discrete parts. Therefore, it is investigated in this chapter if the policies need to be adapted in order to function in a more realistic system with variability and discrete parts.

3.2 Control policies

3.2.1 Control policy of Savkin

The control policy of Savkin (SAV), as has been introduced in [Sav98], is the only control policy found to be proven stable for both acyclic and non-acyclic systems. In [Sav03] a modified version of the policy is introduced, which focusses on a manufacturing environment. The policy of [Sav03] has been developed for a deterministic hybrid dynamical manufacturing system, such as the system described in Section 2.1. Thus, the policy is only defined for constant values of the transportation delay, arrival and process rate.

The feedback policy that [Sav03] proposes has the following general form. When processing work from a buffer b and the buffer becomes empty or when the machine has been processing work from that buffer for a certain amount of time then perform a setup to a different buffer b'. The length of the setup to b' depends on the time that the machine has been processing parts from buffer b. This can be formalized in the following general form: design sets $\mathcal{T}_m(b \mapsto b')$

and functions $F_m : \left([q_m(\cdot), x_b(\cdot)]_0^t\right) \mapsto [\theta_m, \infty)$,where $1 \leq m \leq M,\ b \in B_m,\ b' \in B_m$. These sets and functions define a feedback policy of the form:

$$\textbf{if } \left(q_m(t) = b \quad \textbf{and} \quad [q_m(\cdot), x_b(\cdot)]_0^t \in \mathcal{T}_m(b \mapsto b')\right) \textbf{ then}$$

$$\begin{aligned} &\theta_m^s(t) := F_m\left([q_m(\cdot), x_b(\cdot)]_0^t\right) \\ &q_m(\hat{t}) := 0 \quad \forall \hat{t} \in (t, t + \theta_m^s(t)] \\ &q_m(t + \theta_m^s(t) + 0) := b'. \end{aligned} \tag{3.1}$$

Notice that a feedback policy of the form (3.1) is distributed and that the setup time $\theta_m^s(t)$ is the control variable. However the condition:

$$\theta_m^s(t) \geq \theta_m, \tag{3.2}$$

should be satisfied.

In [Sav03] the following constants $d_1 > 0, d_2 > 0, \ldots, d_p > 0$ are introduced, which are desired production levels and $T > 0$, which is the length of a given scheduling period. The control objective of [Sav03] is then formalized in the following definitions and problem statement.

Definition 3.1. [Sav03] The closed-loop system (2.3), (2.4), (2.5) and (3.1) is said to be *regular* with the production levels $d_1, d_2, \ldots, d_P$ and the scheduling period T if it is stable and the following condition holds. For any solution $[q(t), x(t)]$ to the system with initial conditions (2.1), the output $y(\cdot)$ satisfies:

$$\lim_{j \to \infty} (y_{p,i}((j+1)T) - y_{p,i}(jT)) = d_p \quad \forall p \in \mathcal{P},\ \forall i \in \mathcal{I}_p. \tag{3.3}$$

Definition 3.2. [Sav03] Assume that $d_1 > 0, d_2 > 0, \ldots, d_P > 0$ are given. The minimal time T_0 for which there exist constant arrival rates $\lambda_1^c > 0, \lambda_2^c > 0, \ldots, \lambda_P^c > 0$ and a feedback policy such that the closed-loop system is regular with the production levels $d_1, d_2, \ldots, d_P$ and the scheduling period T_0, is called the *minimal scheduling period* of the system with the production levels $d_1, d_2, \ldots, d_P$.

In other words, a flexible manufacturing system with constant arrival rates is called regular if the two following conditions hold:

1. All the trajectories of the closed-loop system are bounded, which means that the system can operate with finite buffer capacity.
2. The production of the part-types $1, 2, \ldots, P$ over time intervals $[jT, (j+1)T]$ converges to the given desired production levels $d_1, d_2, \ldots, d_P$ as j tends to infinity.

The possible minimal value of such time T is called the minimal scheduling period T_0. The Savkin policy is designed to solve the following problem.

Problem statement: For the following manufacturing system defined by its production paths $\alpha_{p,1}, \alpha_{p,2}, \ldots, \alpha_{p,n_p}$ of the part types $\mathcal{P} = \{1, 2, \ldots, P\}$, constant machines rates $\mu_{p,i}^c$, constant transportation delays $l_{p,i}^c$ and minimal setup times θ_m. Find the minimal scheduling period T_0 of the system with given desired constant production levels $d_1, d_2, \ldots, d_P$. For any

$T \geq T_0$, propose constants arrival rates $\lambda_1^c > 0, \lambda_2^c > 0, \ldots, \lambda_P^c > 0$ and a feedback policy such that the closed-loop system is regular with the production levels $d_1, d_2, \ldots, d_P$ and the scheduling period T.

Note that the above problem statement is a slightly modified version of the following optimization problem: find the minimal time T and a control policy such that production of part types over the time interval $[0, T]$ equals to given desired levels. No constructive implementable real time solutions are known for this problem. However, the modified problem has a constructive real time solution.

The main result

The minimal scheduling period T_0 can now be calculated with [Sav03]:

$$T_0 := \max_{m=1,\ldots,M} \left[k_m \theta_m + \sum_{b_{p,i} \in B_m} \frac{d_p}{\mu_{p,i}^c} \right], \tag{3.4}$$

in which k_m represents the number of buffers in B_m. Furthermore, the following constants are introduced for $T \geq T_0$ [Sav03]:

$$\Theta_m^s := \frac{T - \sum\limits_{b_{p,i} \in B_m} \frac{d_p}{\mu_{p,i}^c}}{k_m}. \tag{3.5}$$

The feedback policy that is proposed in [Sav03] reads as follows. Form the following cyclic sequence of buffers of machine m:

$$b_1 \mapsto b_2 \mapsto \ldots \mapsto b_{k_m} \mapsto b_1. \tag{3.6}$$

Where $b_1, b_2, \ldots, b_{k_m}$ denote the k_m buffers of machine m in an arbitrary order. Let $b \in B_M$, then $\mathbf{next_{SAV}}[b]$ is the buffer from B_m that is next to b in the cycle sequence (3.6). Let $[x(t), q(t)]$ be a trajectory of the system and introduce the function:

$$\tau_m[q_m(\cdot)|_0^t](t) := \inf \{ t_0 \leq t |\ q_m(s) = q_m(t) \qquad \forall s \in (t_0, t] \}. \tag{3.7}$$

Which is the time when the machine m started to work from the current buffer. The feedback policy then becomes:

$$\textbf{if } \left(q_m(t) = b_{p,i} \quad \wedge \quad \left(x_{p,i}(t) = 0 \quad \vee \quad t - \tau_m[q_m(\cdot)|_0^t] = \tfrac{d_p}{\mu_{p,i}^c} \right) \right) \textbf{ then}$$

$$\begin{aligned} &\theta_m^s(t) := \Theta_m^s + \tfrac{d_p}{\mu_{p,i}^c} - t + \tau_m[q_m(\cdot)|_0^t] \\ &q_m(\hat{t}) := 0 \quad \forall \hat{t} \in (t, t + \theta_m^s(t)] \\ &q_m(t + \theta_m^s(t) + 0) := \mathbf{next_{SAV}}[b]. \end{aligned} \tag{3.8}$$

Since $T \geq T_0$ it is implied by (3.4) and (3.5) that $\Theta_m^s \geq \theta_m$, thus requirement (3.2) is always satisfied. The arrival rates can then be calculated with:

$$\lambda_p^c := \frac{d_p}{T} \qquad \forall p. \tag{3.9}$$

In words, the controller limits that the time the machine is allowed to process from a buffer. The machine idles when the time limit has not yet been reached and the buffer is empty. The setup for a different buffer is started when the time limit expires. However, the setup time can be larger than the time required to perform the setup in order to regulate the total system. Note that the policy does not depend on the transportation delay $l_{p,i}^c$.

In [Sav03] no proof is provided for the main results as stated above. In [Sav98] it is shown that a switched server queueing network is regular with the policy described above. However the network as described in [Sav98] differs slightly from the manufacturing system described above. Therefore, it shown in Appendix A that the proof provided in [Sav98] also holds for the manufacturing system in [Sav03].

3.2.2 Clear the Largest Work Policy

The Clear the Largest Work policy (CLW) belongs to the class of Clear A Fraction (CAF) policies. CAF policies are all exhaustive, patient, queue and system greedy and have a FIFO service discipline. Note that the policy has been developed for a deterministic hybrid dynamical manufacturing system with bounded burstiness inputs.

Definition 3.3. [Kum90] An exhaustive policy is said to be a *Clear A Fraction (CAF)* policy, if for each machine m there exists a constant $\epsilon_m > 0$, and a constant K_m such that if machine m commences a setup for buffer $b_{p,i} \in B_m$ at time t, then:

$$x_{p,i}(t) \geq \epsilon_m \sum_{b_{q,j} \in B_m} x_{q,j}(t) - K_m. \tag{3.10}$$

The Clear the Largest Work (CLW) Policy belongs to the class of CAF policies.

Definition 3.4. [Per89] An exhaustive policy is said to be a *Clear the Largest Work (CLW)* policy if machine m commences a setup for buffer $b_{p,i} \in B_m$ at time t such that:

$$\frac{x_{p,i}(t)}{\mu_{p,i}^c} \geq \frac{x_{q,j}(t)}{\mu_{q,j}^c} \qquad \forall b_{q,j} \in B_m. \tag{3.11}$$

In words, after emptying a buffer, the machines will serve the buffer that takes the most time to empty.

Let $b_{p,i} \in B_m$ and introduce the following function:

$$\tau_m^l(t) := \inf\{s \geq t |\ \exists x_{p,i}(s) \neq 0\}. \tag{3.12}$$

In other words, $\tau_m^l(t)$ is the time $s \geq t$ at which one or more of the buffers is or are not empty. Furthermore, assume that machine machine m has been processing work from $b_{p,i}$,

then the function $\text{next}_{\mathbf{LW}}[B_m](t)$ returns the buffer $b_{q,j} \neq b_{p,i},\ b_{q,j} \in B_m$ which contains the most work as defined by (3.11).

Let $b_{p,i} \in B_m$. The CLW feedback policy can now be defined:

$$\begin{aligned}
&\textbf{if } (q_m(t) = b_{p,i} \quad \wedge \quad x_{p,i}(t) = 0) \textbf{ then} \\
&\quad \textbf{if } (x_{q,j}(t) = 0 \quad \forall b_{q,j} \in B_m) \textbf{ then} \\
&\qquad q_m(\tilde{t}) := 0 \quad \forall \tilde{t} \in (t, \tau_m^l(t)] \\
&\qquad q_m(\tau_m^l(t) + 0) := b_{p,i} \\
&\quad \textbf{else} \\
&\qquad q_m(\hat{t}) := 0 \quad \forall \hat{t} \in (t, t + \theta_m] \\
&\qquad q_m(t + \theta_m + 0) := \text{next}_{\mathbf{LW}}[B_m](t).
\end{aligned} \tag{3.13}$$

In words, if at time t the buffer $b_{p,i}$ from which machine m is processing becomes empty. Then the policy checks if all buffers at machine m are empty. If all buffers are empty, the machine idles until one of the buffers becomes non-empty. The policy then assigns machine m to work again on buffer $b_{p,i}$ and starts all over. If not all buffers are empty, machine m commences a setup for the buffer containing the most work. Note that the policy does not depend on the transportation delay $l_{p,i}^c$.

3.2.3 Clear the Largest Scaled Age Policy

The Clear the Largest Scaled Age (CLSA) heuristic policy as introduced by [Ols99] uses part ages multiplied by a scale factor to determine which buffer should be served next. The two previous discussed control policies (SAV and CLW) were analytical proven to be stable for a deterministic system or for a system with a limited variability, both with only continuous work. CLSA has not been proven stable. However, it was shown with the use of simulation that CLSA is stable for a stochastic single machine system with a discrete flow of parts.

Definition 3.5. [Ols99] An exhaustive policy is said to be a *Clear the Largest Scaled Age (CLSA)* policy if machine m commences a setup for a non-empty buffer $b_{p,i} \in B_m$ at time t such that:

$$w_{p,i}\hat{A}_{p,i}(t) \geq w_{q,j}\hat{A}_{q,j}(t) \qquad \forall b_{q,j} \in B_m \text{ for which } x_{q,j} \neq 0 \text{ holds.} \tag{3.14}$$

In which $\hat{A}_{p,i}(t)$ is the *total expected age* after a setup for buffer $b_{p,i}$ and $w_{p,i}$ is a scaling factor:

$$\hat{A}_{p,i}(t) = \lambda_p \frac{{\theta_m}^2}{2} + \theta_m x_{p,i}(t) + A_{p,i}(t). \tag{3.15}$$

In which $A_{p,i}(t)$ denotes the *total age* currently present in buffer $b_{p,i}$. Note, that the buffer level $x_{p,i}(t)$ has a discrete value and indicates the number of parts in the buffer. The total age is the age of all the parts in the buffer add up together. In other words, if $tin_{p,i}^j$ is the

time at the jth part arrives in buffer $b_{p,i}$ after the buffer became empty then the total age is defined as:

$$A_{p,i}(t) = \sum_{j=1}^{x_{p,i}} (t - tin_{p,i}^{j}). \tag{3.16}$$

The scaling factor $w_{p,i}$ is defined as:

$$w_{p,i} = \frac{1}{\theta_m(1 - \frac{\lambda_{p,i}}{\mu_{p,i}})}. \tag{3.17}$$

Note that the scaling factor $w_{p,i}$ assigns a low priority to buffers with a long setup time or a low utilization.

Furthermore, assume that machine m has been processing work from $b_{p,i} \in B_m$, then the function, $\text{next}_{\mathbf{CLSA}}[B_m](t)$ returns the buffer $b_{q,j} \neq b_{p,i},\ b_{q,j} \in B_m$ which has the largest scaled age, as defined in (3.14). The CLSA feedback policy can now be defined:

$$\begin{aligned}
&\textbf{if } (q_m(t) = b_{p,i} \quad \wedge \quad x_{p,i}(t) = 0) \textbf{ then} \\
&\qquad \textbf{if } (x_{q,j}(t) = 0 \quad \forall b_{q,j} \in B_m) \textbf{ then} \\
&\qquad\qquad q_m(\tilde{t}) := 0 \quad \forall \tilde{t} \in (t, \tau_m^l(t)] \\
&\qquad\qquad q_m(\tau_m^s(t) + 0) := b_{p,i} \\
&\qquad \textbf{else} \\
&\qquad\qquad q_m(\hat{t}) := 0 \quad \forall \hat{t} \in (t, t + \theta_m] \\
&\qquad\qquad q_m(t + \theta_m + 0) := \text{next}_{\mathbf{CLSA}}[B_m](t).
\end{aligned} \tag{3.18}$$

In which $\tau_m^l(t)$ is defined as in (3.12). In words, if at time t buffer $b_{p,i}$ from which machine m is processing is empty. Then the policy checks if all buffers at machine m are empty. If all buffers are empty, the machine idles until one of the buffers becomes non-empty. The policy then assigns machine m to work again on buffer $b_{p,i}$ and starts all over. If not all buffers are empty, machine m commences a setup for the non-empty buffer having the largest scaled age.

In [Ols99] no (stochastic) transportation delay $l_{p,i}$ was included in the simulation used to show stability. As can been seen from (3.14) and (3.18), a transportation delay does not influence the policy directly. The transportation delay does influence the arrival rate. However, it only adds more variability to the arrival rate. CLSA was shown stable for a stochastic single machine system, thus adding a transportation delay does not result in instability for the single machine system.

3.3 Adapting the policies

In the previous section the control policies SAV, CLW and CLSA have been explained. SAV and CLW have been developed for a deterministic system or for a system with a limited variability, both with only continuous work. No variability and no discrete parts is not realistic for a manufacturing environment. Therefore, it is investigated in this section if the

policies can cope with stochastic systems with discrete parts instead of deterministic systems with a continuous flow of work. Furthermore, if a policy can not cope with a stochastic system with discrete parts, the policy is adapted so that is can work with such a system.

In the next chapter the three policies are compared for different system parameters such as arrival rates, number of buffers, etc. However, the policy of Savkin works with given desired production levels and not with given arrival rates. The policy of Savkin can be rewritten such that it works with given arrival rates instead of given desired production levels. This version was already provided in [Sav98]. In order for the Savkin policy to work with arrival rates equations (3.4), (3.5) and (3.8) are changed respectively into:

$$T_0 := \max_{m=1,\ldots,M} \left\lceil \frac{k_m \theta_m}{1 - \sum_{b_{p,i} \in B_m} \frac{\lambda_p^c}{\mu_{p,i}^c}} \right\rceil, \tag{3.19}$$

$$\Theta_m^s := \frac{T - \sum_{b_{p,i} \in B_m} \frac{\lambda_p^c T}{\mu_{p,i}^c}}{k_m}, \tag{3.20}$$

$$\textbf{if } \left(q_m(t) = b_{p,i} \quad \wedge \quad \left(x_{p,i}(t) = 0 \quad \vee \quad t - \tau_m[q_m(\cdot)|_0^t] = \frac{\lambda_p^c T}{\mu_{p,i}^c} \right) \right) \textbf{ then}$$

$$\begin{aligned} &\theta_m^s(t) := \Theta_m^s + \tfrac{\lambda_p^c T}{\mu_{p,i}^c} - t + \tau_m[q_m(\cdot)|_0^t] \\ &q_m(\hat{t}) := 0 \quad \forall \hat{t} \in (t, t + \theta_m^s(t)] \\ &q_m(t + \theta_m^s(t) + 0) := \mathbf{next_{SAV}}[b]. \end{aligned} \tag{3.21}$$

Note that the rewritten version cannot deal with total load factors equal to one and setup times equal to zero.

For CLW it can happen that there is a tie, at the moment that the policy needs to decide which buffer to process from. In other words, several buffers have the same largest workload. In order to make the comparison between the policies more fair, the following adaption is made to CLW. If there is a tie, it is broken by choosing the buffer with the largest scaled age. This slightly adapted CLW policy is denoted with CLW$^{\text{a}}$.

3.3.1 Introducing variability

The policy of Savkin and the Clear the Largest Work policy are developed for a deterministic system or for a system with a limited variability, both with only continuous work. In this section it is investigated if these two policies can cope with unlimited variability. Only variability in the process and arrival rate is studied. Until stated otherwise it is assumed that a transportation delay is not present. In Chapter 2 it was explained for a stochastic manufacturing system that the utilization u_m and the average total load factor ρ_m should be smaller than 1, in order to avoid instability. The utilization of a machine is the fraction of time it is not idle for the lack of parts. Therefore, the utilization of a machine with non-zero setup times and processing different parts, should include a factor for the time required to perform a setup. As stated in Chapter 2, this setup factor is difficult to determine. However,

this factor is easy to determine for a machine under control of SAV. The utilization is easy to compute, because of the regular behavior present. The setup fraction equals the fraction of a scheduling period T the machine is deliberate not processing parts. The time that machine m is not processing parts in one scheduling period T equals Θ_m^s, which can be computed with equation (3.5) or (3.20). Thus, the setup fraction equals:

$$\frac{k_m \Theta_m^s}{T}. \tag{3.22}$$

The utilization u_m^s for a machine under the SAV control policy then becomes:

$$u_m^s = \sum_{b_{p,i} \in B_m} \frac{\lambda_p^c}{\mu_{p,i}^c} + \frac{k_m \Theta_m^s}{T} \tag{3.23}$$

If one fills in Θ_m^M then

$$\left.\begin{array}{l} u_m^s = \sum\limits_{b_{p,i} \in B_m} \frac{\lambda_p^c}{\mu_{p,i}^c} + \frac{k_m \Theta_m^s}{T} \\ \Theta_m^s := \frac{T - \sum\limits_{b_{p,i} \in B_m} \frac{\lambda_p T}{\mu_{p,i}^c}}{k_m} \end{array}\right\} u_m^s = 1. \tag{3.24}$$

As can been seen from (3.24), the utilization for the Savkin policy equals 1. Thus, the Savkin policy is only stable for a deterministic system. SAV always controls the system such that for the given total load factor $\lambda_p^c/\mu_{p,i}^c$ the utilization equals one. For a stochastic system tuning the policy with the average total load factor $\lambda_p/\mu_{p,i}$ seems a logical choice. However, the real total load factor is stochastic and can be larger than the average total load factor. This results in an utilization higher than one, thus instability. Therefore, the Savkin policy is adapted, in order to introduce variability without resulting in an unstable system. The idea behind the adaptation is that the policy is tuned with the largest total load factor possible. The maximum arrival rate and the minimum process rate have to be known in order to compute the largest total load factor possible. To know these values a special distribution for the stochastic variables (process and arrival rate) is used. The form of the distribution is an isosceles triangle, such that the average of the distribution equals the position of the top of triangle. The triangular distribution is depicted in Figure 3.1. The advantage of a triangular distribution is that it has limited extreme values. A disadvantage is the limited variability. The average of the distribution is set equal to the average arrival rate or average process rate. The angular points of the triangle are chosen in such a way that the maximum value of the load factor of a part p on machine m equals $80/87 = 1.175$ times the average load factor of a part p on machine m. The left and right angular point lay respectively at $80/87$ and $94/87$ times the average. The different parameters are summarized in Table 3.1.

	Arrival rate	Process rate	Load factor of one part
average	λ_p	$\mu_{p,i}$	$\rho_{p,i}$
maximum	$\lambda_p^{\max} = \frac{94}{87}\lambda_p$	$\mu_{p,i}^{\max} = \frac{94}{87}\mu_{p,i}$	$\rho_{p,i}^{\max} = \frac{\lambda_p^{\max}}{\mu_{p,i}^{\min}} = \frac{94}{80} = 1.175$
minimum	$\lambda_p^{\min} = \frac{80}{87}\lambda_p$	$\mu_{p,i}^{\min} = \frac{80}{87}\mu_{p,i}$	$\rho_{p,i}^{\min} = \frac{\lambda_p^{\min}}{\mu_{p,i}^{\max}} = \frac{80}{94}$

Table 3.1: The parameters of the triangular distribution.

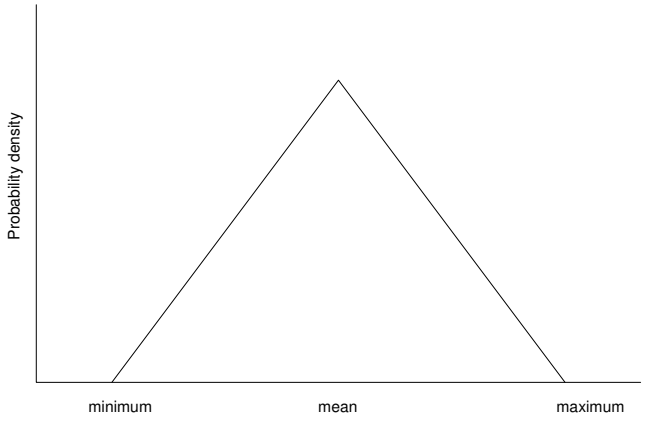

Figure 3.1: The triangular distribution.

The Savkin policy can now be tuned with the parameters which result in the highest total load factor. Thus, the maximum arrival rates and the minimum process rates are used to tune the Savkin policy. These settings result in a controller, which is guaranteed stable even for the highest total load factor possible. However, the average total load factor is lower, since the average arrival and process rate differ from the values used for tuning the controller. Therefore, the limited time allowed to process parts from a buffer is much larger than needed. This results in the machine to idle often, because the buffer is empty before the time limit has been reached. The machine under utilizes its capacity. This adapted version of SAV is denoted as SAV$^{\text{a}}$.

Calculating the utilization for a system under control of CLW$^{\text{a}}$ or CLSA is more difficult. However, variability does not cause problems for the two policies, because the policies do not, like SAV, control the machine in such a way that the utilization equals 1. The only restriction for introducing variability is that the capacity condition (2.9) is satisfied.

3.3.2 Introducing discretization

The SAV and CLW policy are designed for hybrid dynamical systems, thus for a continuous flow of work. However, in a realistic manufacturing environment this flow often consists of individual parts. In this section it is investigated if the two policies can cope with discrete

work. Consider a deterministic system in which the work present consists of discrete parts and where the machine has to finish processing a part before it starts with a different operation. If this system is under control of SAV and when using integer desired production levels d_p, the machine finishes a part exactly at the same time it should switch to a different buffer. However, if the systems is stochastic or when the desired production levels d_p are not integer values, the possibility exist that the machine is still processing a part at the moment it should switch to a different buffer. Thus, the Savkin policy cannot cope with a discrete flow of work. In order overcome this problem the already adapted SAV^a policy is adapted even further so that it finishes a part before it switches to a different buffer. As a result the computed setup time θ_m^s (3.8) can become smaller than the minimal required setup time θ_m. Therefore, if this occurs the computed setup time θ_m^s is set equal to the minimal required setup time θ_m in order to satisfy condition (3.2).

The CLW^a and CLSA control policies do not limit the time allowed to process a part type. The policies only control to which buffer a machine should switch. This is not affected by the by the introduction of discretization.

3.4 Resumé

In this chapter SAV, CLW and CLSA have been explained which can be used for controlling flexible manufacturing systems. SAV and CLW are designed only for a deterministic system or for a system with a limited variability, both with only continuous work. Therefore, it is has been investigated in this chapter if these policies can cope with stochastic systems with discrete parts. It turned out that CLW can, but SAV can not cope with such systems. To partially overcome this problem, SAV has been adapted such that it functions in a discrete system with a triangular distribution for the arrival and process rates. The adapted Savkin policy is denoted with SAV^a. Furthermore, CLW has been slightly adapted to make a comparison between the policies more fair. This slightly adapted CLW policy is denoted with CLW^a. The (adapted) policies are now suitable for a more realistic manufacturing environment. In the next chapter simulation studies are designed and performed for comparing the policies for a single machine system with parts which have an acyclic production path.

Chapter 4

Acyclic systems

In the previous chapter three control policies, SAV, CLW and CLSA, have been explained that are suitable for controlling a flexible manufacturing system as described in Chapter 2. If necessary, the policies have been adapted to be suitable for realistic flexible manufacturing systems. SAV and CLW have been adapted and the adapted versions are denoted with SAV^a and CLW^a. In this chapter a first comparison between the policies is made by simulating a single machine flexible manufacturing system. The comparison is made by simulating flexible manufacturing systems and study the performances of the policies. Furthermore, the influences of system parameters on the performance is investigated. Simulation experiments are designed for comparing the different policies for different situations. It is difficult to study and analyze the performances of the policies for a large complex system. For that reason, a simple single machine system with acyclic production paths is studied. However, first the performance of a control policy is defined.

4.1 Simulation experiments

In this section experiments are designed to compare the performances of CLSA, SAV^a and CLW^a and to study the influence of system parameters on the performances. The average flow time of all parts is chosen, as the performance measure for a policy. The lower the average flow time, the higher the performance of a control policy. In [Ols99] it has been shown, using simulation, that CLSA performs well compared to other policies. Therefore, the flow times of SAV^a and CLW^a are compared to the flow time of CLSA. The percentile difference between the flow time of CLSA and SAV^a or CLW^a is examined. If a is the flow time of CLSA and b is the flow time of SAV^a or CLW^a then the percentage difference is computed as $100(b-a)/a$. The results of the simulation experiments are presented and discussed in Section 4.2.

The system under consideration exists of only one workstation, which processes P part types. The parts visit the machine only once. In Figure 4.1 the system is depicted.

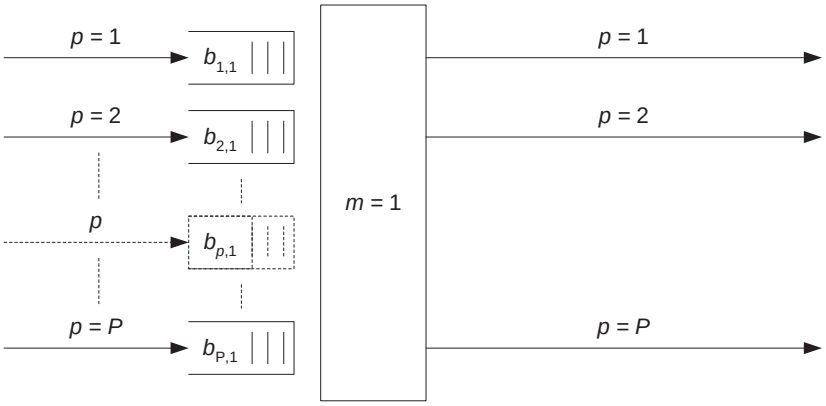

Figure 4.1: An acylic single machine system with P part types.

Variability is present in the arrival and process rates in the form of triangularly distributions. In Section 3.3.1 is was explained how and why a triangularly distribution is used. The average total load factor (2.7) has to be smaller than one, otherwise the system can not be stabilized.

A single machine system can be symmetric or asymmetric, both situations are investigated. A symmetric system is studied first.

4.1.1 A symmetric system

For a symmetric system, the following systems parameters are interesting to investigate:

- The number of buffers at a machine.
- The initial buffer levels.
- The average total load.

Experiment 1: the number of buffers at a machine

In this experiment it is investigated if and how the number of buffers that a machine has to serve, influence the performance of a control policy. For this experiment the average total load factor (2.7) is set equal to 0.8, the process rates equal to 1 for every part type and the setup time equal to 1. The arrival rate of a part depends on the number of buffers in the system in order to keep the load factor equal to 0.8. Since the number of buffers equals the number of part types, the arrival rate of a part type p becomes:

$$\lambda_p = \frac{\rho_m}{P}\mu_{p,1}. \tag{4.1}$$

The system can be summarized as:

$$\begin{aligned}
\mathcal{P} &= \{1, 2, \ldots, P\} \\
\mathcal{M} &= \{1\} \\
\rho_m &= 0.8 \\
\alpha_{p,1} &= 1 \\
\lambda_p &= \frac{\rho_m}{P}\mu_{p,1} \\
\mu_{p,1} &= 1 \\
\theta_1 &= 1.
\end{aligned}$$

The system is studied for $2, 3, 10, 20$ and 100 buffers at machine 1.

Experiment 2: the initial buffer levels

The goal of this experiment is to investigate if and how the initial buffer levels influence the performance of the policies for a stochastic system. The setup of the system is the same as for Experiment 1. The number of part types in the system is two. The system is studied for initials buffer levels of 0 and 100 parts for every buffer.

Experiment 3: the average total load

The same system as for Experiment 1 is used to investigate the influence of the total load factor on the performances of the policies. There are 10 part types in the system. The total load factor is varied by changing the arrival rates of the parts according to (4.1). The following total load factors are used: $0.8, 0.6$ and 0.4.

4.1.2 An asymmetric system

A system is called asymmetric if not all part types have the same arrival rate, setup time and process rate distributions. The SAV$^{\text{a}}$ policy can not cope with asymmetric setup times. All the setup times have to be identical. Therefore, only asymmetry in the arrival and process rate is studied. The system under consideration is the same as for the symmetric system, as depicted in Figure 4.1. The number of buffers is always even in order not to introduce skewness. The following parameters are investigated:

- The size of the asymmetry in the arrival rates.
- The size of the asymmetry in the process rates.

Experiment 4: the size of the asymmetry in the arrival rates

Asymmetry of the arrival rates is introduced in the following manner. As mentioned earlier the number of buffers at the machine is always even. The buffers are divided in two groups, each of the same size $P/2$. Thus, the first group exists of the buffers $b_{1,1}, b_{2,1}, \ldots, b_{\frac{P}{2},1}$ and

the second group of the buffers $b_{(\frac{P}{2}+1,1)}, b_{(\frac{P}{2}+2,1)}, \ldots, b_{P,1}$. Asymmetry in the arrival rates is introduced by assigning the first group a lower average arrival rate than the other group in such a way that:

$$\lambda_{(\frac{P}{2}+1)\ldots P} = \xi \cdot \lambda_{1\ldots\frac{P}{2}}, \tag{4.2}$$

in which P is the number of buffers served by the machine and ξ a factor for the asymmetry present. The total load factor then equals:

$$\rho_m = \sum_{p=1}^{\frac{P}{2}} \frac{\lambda_p}{\mu_{p,1}} + \sum_{p=(\frac{P}{2}+1)}^{P} \frac{\lambda_p}{\mu_{p,1}}. \tag{4.3}$$

Using (4.2), this can be rewritten:

$$\lambda_{1\ldots\frac{P}{2}} = \frac{\rho_m \mu_{p,1}}{(1+\xi)} \frac{2}{P}. \tag{4.4}$$

The parameters for this experiment are the same as for Experiment 1, thus the total load factor (2.7) is set equal to 0.8, the process rates equal to 1 for every part type and the setup time equal to 1. The arrival rates depend on the number of buffers in the system, as defined by (4.2) and (4.4). The system then becomes:

$$\begin{aligned}
\mathcal{P} &= \{1, 2, \ldots, P\} \\
\mathcal{M} &= \{1\} \\
\rho_m &= 0.8 \\
\alpha_{p,1} &= 1 \\
\mu_{p,1} &= 1 \\
\lambda_{1\ldots\frac{P}{2}} &= \frac{\rho_m \mu_{p,1}}{(1+\xi)} \frac{2}{P} \\
\lambda_{(\frac{P}{2}+1)\ldots P} &= \xi \cdot \lambda_{1\ldots\frac{P}{2}} \\
\theta_1 &= 1.
\end{aligned}$$

The system is studied for 10 buffers and for asymmetric factors of $0, 5, 10, 25, 50$ and 100.

Experiment 5: the size of the asymmetry in the process rates

In this experiment the influence of asymmetry in the process rates is investigated. Asymmetry in the process rates is introduced in the same manner as for the arrival rates. Equations (4.2) and (4.4) rewritten for the process rates become:

$$\mu_{(\frac{P}{2}+1)\ldots P,1} = \xi \cdot \mu_{1\ldots\frac{P}{2},1} \tag{4.5}$$

and

$$\mu_{1\ldots\frac{P}{2},1} = \frac{\lambda_p \left(1 + \frac{1}{\xi}\right)}{\rho_m} \frac{P}{2}. \tag{4.6}$$

The system setup is the same as for Experiment 4. The arrival rates equals 0.08 for every part type. The system is investigated for an asymmetric factor of 0 and 25 in the process rates.

4.2 Simulation results and discussion

In this section the results of the experiments as discussed in Section 4.1 are given. The results are obtained by simulation.

The simulations are performed using the formalism χ [Hof02], Python [Ros04] and Matlab [Mat02]. The different models can be found in Appendix D. Comparing the policies on the results of just one simulation can lead to wrong conclusions, because of the variability present in the simulation results. In order to overcome this problem, multiple simulations are performed, as described in Appendix C. The relative error, as defined in Appendix C, used for the simulations equals 5%.

The results of the experiments are depicted in tables. In the first column of such a table the values of the parameter that is studied are given. In the second to fourth column the average flow times of the system under control of the different policies are given. In the fifth column the percentile difference of the average flow time from SAV^a compared to CLSA is depicted. In the final column the percentile difference between CLW^a and CLSA is given. Note, that the percentile difference is computed as follows. If a is the flow time of CLSA and b is the flow time of SAV^a or CLW^a then the percentage difference is computed as $100(b-a)/a$. If the difference is:

- > 0, than the policy performs worse than CLSA.
- $= 0$, than the policy performs the same as CLSA.
- < 0, than the policy performs better than CLSA.

4.2.1 Symmetric system

Experiment 1: the number of buffers at a machine

In Table 4.1 the simulation results of Experiment 1 are depicted. From column two to four it is clear that the average flow time increases as the number of buffers increases. A possible explanation for this increase is that the time between two production runs from the same buffer increases with an increasing number of buffers, since more buffers need to be processed from. This results in longer waiting times of the parts in the buffers and thus in an increase of the average flow time.

In [Liu92] it has been proven that the CLW policy has the best performance for a symmetric system. The system of Experiment 1 is a symmetric system. Thus, CLW^a should have the best performance, since it only differs slightly from CLW. The table shows that CLSA has the same performance as CLW^a. A possible explanation for the good performance of CLSA is the following. The symmetry in the system causes the largest buffer to have, on average, the largest scaled age. Thus, CLSA behaves the same as CLW^a. The performance of SAV^a is

worse than that of CLSA and worsens with an increasing number of buffers. An explanation for this deterioration of performance is the underutilization of the capacity present. The average total load factor is lower than SAV^a is tuned for. This results in the machine to idle often.

Num. of Buffers	Flow time			Percentile difference to CLSA	
	SAV^a	CLW^a	CLSA	SAV^a	CLW^a
2	11.0	4.0	4.0	175%	0%
3	19.3	6.5	6.5	197%	0%
10	77.5	24.0	24.0	223%	0%
20	160.9	48.8	48.8	230%	0%
100	827.6	231.3	232.0	257%	0%

Table 4.1: The results of Experiment 1.

Experiment 2: the initial buffer levels

The results of Experiment 2 can be found in Table 4.2. It can be seen that the initial buffer levels do not influence the performance of the system under any of the policies. The initial buffer levels do not influence the performance of CLW and CLSA, because the initial buffer levels are emptied and "forgotten" on the long term. This also holds for SAV^a, because of the underutilization of the capacity present in the system. There is enough capacity left to process the initial buffer levels. As explained in Section 3.3.1, the underutilization is caused by the adaption made to SAV. Another reason for the lack of influence of the initial buffer levels is the variability present, resulting in the initial buffer levels to be "forgotten" on the long term.

Initial buffer level	Flow time			Percentile difference to CLSA	
	SAV^a	CLW^a	CLSA	SAV^a	CLW^a
0	11.0	4.0	4.0	175%	0%
100	11.0	4.0	4.0	175%	0%

Table 4.2: The results of Experiment 2.

Experiment 3: the average total load

The results of Experiment 3 are provided in Table 4.3. As can been seen from column two to four, the average flow time decreases with a decreasing average total load factor. For a non-switching machine the waiting time in the buffers is influenced by the utilization. The waiting time decreases with a decreasing utilization. The utilization consists among other things of the load factor. Therefore, the load factor has the same influence as the utilization on the waiting time in the buffers, also for machines in flexible manufacturing system. A decreasing average total load factor results in a decreasing waiting time of the parts in the

buffers. The flow time consists of the waiting time in the buffers and the process time of the part. Thus, the average flow time decreases with an decreasing average total load factor.

From the final column it can been seen that the load factor does not influence the performance of CLW^a compared to CLSA. As can be seen from the fifth column the performance of SAV^a compared to CLSA seems to increase with an decreasing load factor. An possible explanation for the increasing performance of SAV^a compared to CLSA is the following. A smaller load factor results in a decreasing scheduling period T, thus buffers are emptied more often for a shorter period and the idling periods are shorter. However, since there is enough capacity, the buffers are still emptied. This results in a behavior that starts to show similarity to the exhaustive policies, thus the performance with respect to the exhaustive policies increases.

Total load	Flow time			Percentile difference to CLSA	
	SAV^a	CLW^a	CLSA	SAV^a	CLW^a
0.8	77.5	24.0	24.0	223%	0%
0.6	17.1	13.8	13.8	24%	0%
0.4	10.8	3.6	3.6	200%	0%

Table 4.3: The results of Experiment 3.

The results discussed above were all for symmetric systems. As discussed earlier CLW^a has the best performance for a symmetric system. CLSA has the same performance as CLW^a, since in a symmetric system it has the same behavior as CLW^a. The buffers containing the most work, will on average also have the largest scaled age. SAV^a has a bad performance compared to the other two policies. The bad performance is caused by the underutilization of the capacity present in the system.

4.2.2 Asymmetric system

In this section the results of the experiments as described in Section 4.1.2 are analyzed and discussed. The results are depicted in tables. For an explanation of these tables see Section 4.2.1.

Experiment 4: the size of the asymmetry in the arrival rates

The results of Experiment 4 are depicted in Table 4.4. As can be seen from column two to four the flow time decreases for an increasing asymmetry in the arrival rate. A possible explanation for this decrease is the following. The time between two production runs from the same buffer increases for buffers containing parts which have a low arrival rate. The time between production runs increases, because more time is needed to process from the buffers containing parts with an high arrival rate. The time between two production runs from the same buffer decreases for buffers containing parts which have an high arrival rate. The decrease is caused by the fact that less time is needed to process from the buffers containing parts with a low arrival rate. The average flow time decreases with an increasing asymmetry, because there are more parts with an high arrival rate, thus with a decreasing waiting time in

the buffers, than that are parts with a low arrival rate, thus with an increasing waiting time in the buffers.

The performance of SAV^a and CLW^a in comparison to CLSA degrades with an increasing asymmetric factor. An possible explanation for the decreasing performance for an increasing asymmetric factor is the following. All the average process times are identical, thus the buffer with the most work is also the buffer with most parts. Therefore, CLW^a chooses the largest buffer (at that exact moment) to perform a setup for. Contrary to CLSA, which chooses that buffer which is expected to have the largest scaled age after a setup has taken place. The result is that with CLW^a the buffers containing parts with a very low arrival rate are only visited when their buffer levels are equal or higher than the buffers levels of buffers containing parts with high arrival rates. As a result the buffers are not served often. With CLSA these buffers are visited more often, because their scaled age increases much more rapidly than their buffer size. SAV^a does not choose a buffer to process from, but follows a simple sequence. The larger the asymmetric factor the larger the difference between the buffers, causing it to be more efficient for the flow time to serve a certain buffer more often than others.

Factor	Flow time			Percentile difference to CLSA	
	SAV^a	CLW^a	CLSA	SAV^a	CLW^a
0	77.5	24.0	24.0	223%	0%
5	74.6	20.4	21.1	254%	-3%
10	73.2	19.4	18.6	294%	4%
25	72.0	18.8	15.3	371%	23%
50	71.6	18.4	13.3	438%	38%
100	71.6	18.0	12.4	477%	45%

Table 4.4: The results of Experiment 4.

Experiment 5: the size of the asymmetry in the process rates

Table 4.5 gives the result of Experiment 5. It can be seen that SAV^a and CLSA are not influenced by asymmetry in the process rates. The flow times stay the same with or without asymmetry present. A possible explanation for this is the following. The flow time of a product exists for the most part of the waiting time in the buffer while the machine is not processing parts from that buffer. Therefore, the process rate of a part has a minor influence on the average flow time. For SAV^a the process rate does not influence the choice of which buffer to serve next, since the buffers are visited in a fixed sequence. For CLSA the flow times of the part types are influenced by the asymmetry. However, the average flow time stays the same. The flow time of one group of part types increases with the same amount as the flow time of the other group of part types decreases. The influence of asymmetry in the process rates for a system under control of CLW^a is very large. The reason for this is that CLW^a takes decisions based on the buffer levels times the process times (thus workload). The buffers containing the parts with a small process time have to contain many parts before they are emptied resulting in a large flow time.

Factor	Flow time			Percentile difference to CLSA	
	SAV^a	CLW^a	CLSA	SAV^a	CLW^a
0	77.5	24.0	24.0	223%	0%
25	77.5	97.2	24.0	223%	305%

Table 4.5: The results of Experiment 5.

4.3 Resumé

Both asymmetric and symmetric systems have been studied. It can be concluded that CLSA has the best performance in all situations and that CLW^a has the same performance as CLSA in symmetric systems. In asymmetric systems the performance of CLW^a degrades where asymmetry in the process rates has a much larger influence than asymmetry in the arrival rates. The performance of SAV^a compared to CLSA is bad for all situations. As has been mentioned earlier this bad performance is caused by the policy being tuned for the maximum load factor possible, while the real load factor is lower. This causes the policy to under utilize the capacity present in the system, resulting in a lot of idling.

In this chapter only an acyclic single machine system and the performances of the different policies on this system have been studied. In the next chapter a more complicated system is studied. This system consists of more than one machine and non-acyclic production paths are present.

Chapter 5

Non-acyclic systems

5.1 Introduction

In Chapter 2 a flexible manufacturing system, which could produce different parts on the same machine, has been discussed. In Chapter 3, three policies, SAV, CLW and CLSA, have been explained that are suitable for controlling such a manufacturing system. If necessary, the policies have been adapted to be suitable for realistic flexible manufacturing systems. SAV and CLW have been adapted and the adapted versions are denoted with SAV^a and CLW^a. Furthermore, in the previous chapter simulation studies have been performed in order to compare the three policies, SAV^a, CLW^a and CLSA, on their performance. The performance of a policy was measured by the average flow time of all the parts in the system. The system under consideration consisted of only one machine and parts visited the machine only once. In this chapter a more complicated system is introduced, which has non-acyclic production paths. It is explained that a non-acyclic system can be unstable for certain control policies even if the capacity constraint (2.9) is met. Therefore, a stabilization technique called regulators [Per94] is introduced. Finally, simulation studies are performed to compare the performances of the three policies with and without regulators.

An acyclic system is stable for any DSI policy, defined in Chapter 2, if the capacity condition (2.9) is satisfied. However, stability is not guaranteed for a non-acyclic system satisfying the capacity condition (2.9). In the following example it is shown that a non-acyclic system, which satisfies (2.9), can be unstable.

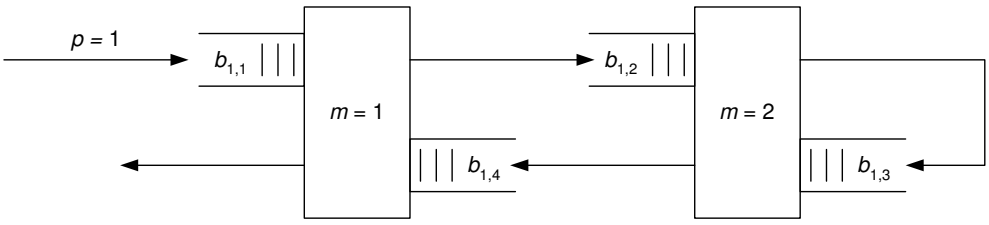

Figure 5.1: The system of Example 5.1.

Example 5.1. *[Kum90] Consider the system as depicted in Figure 5.1 in which no variability is present and defined as:*

$$\begin{aligned}
\mathcal{P} &= \{1\} & \mathcal{M} &= \{1,2\} \\
\alpha_{1,1} &= 1 & \alpha_{1,2} &= 2 \\
\alpha_{1,3} &= 2 & \alpha_{1,4} &= 1 \\
l^c_{1,i} &= 0 & \forall i & \\
\lambda^c_1 &= 1 & & \\
\mu^c_{1,1} &= \frac{10}{3} & \mu^c_{1,2} &= \frac{10}{6} \\
\mu^c_{1,3} &= \frac{10}{3} & \mu^c_{1,4} &= \frac{10}{6} \\
\theta_1 &= 50 & \theta_2 &= 50.
\end{aligned}$$

In words, there is only 1 part, which is processed on 2 machines. The part is first processed on machine one and then two times on machine two. The last process step is again on machine one. A setup is required when switching between processing the parts from different buffers. For simplicity the buffers are labeled b_i instead of $b_{p,i}$. The capacity condition (2.9) *is satisfied, since the average total load factor of both machines is smaller than one:*

$$\rho_1 = \frac{9}{10}, \quad \rho_2 = \frac{9}{10}.$$

Assume that the initial buffer level for b_1 equals 100 and zero for b_2, b_3 and b_4, thus

$$\begin{aligned}
x_{1,1}(0) &= 100 & x_{1,2}(0) &= 0 \\
x_{1,3}(0) &= 0 & x_{1,4}(0) &= 0.
\end{aligned}$$

In Figure 5.2 the behavior of the system of Example 5.1 with an exhaustive patient policy (for example, CLW[a] or CLSA) is depicted. The buffer levels of the four buffers in the system are shown. The machine starts at $t = 0$ with the initial buffer levels as defined above and machine one and two are ready to process from buffers 4 and 3 respectively. At $t = 0$ machine one starts with performing a setup for buffer b_1, since no work is present in b_4. Machine two stays idle at b_3 until works arrives in buffer b_2, which is when machine one starts processing work from b_1. From Figure 5.2 it can be seen that at a $t = T_1$ the system returns to the same state as it was at $t = 0$. Machine one and two are again ready to process work from the buffers 4 and 3 respectively. Furthermore, the buffer levels for buffers 2 until 4 equal $(x_{1,2}, x_{1,3}, x_{1,4}) = (0, 0, 0)$, which are the same levels as at $t = 0$. However, the level of buffer 1 at $t = T_1$ has increased compared to the level at $t = 0$. It has been shown in [Kum90] for this system that if the initial conditions are $(x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}) = (\xi, 0, 0, 0)$ that the buffer levels at $t = T_1 = ((\alpha + \mu^c_{1,2}/(1 - \mu^c_{1,2}))\xi + \gamma$ equal $(x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}) = (\alpha\xi + \beta, 0, 0, 0)$. For this system $\alpha = 1.5$, $\beta = 230$ and $\gamma = 530$. The system will repeat itself with an increasing period and increasing buffer level for buffer 1. Thus, the system has become unstable.

This instability is not caused by the time necessary to perform a setup, because α is independent of the time required to perform a setup and β equals zero when the time required to perform a setup equals zero. Consider the system of Example 5.2, which has setup times equal to zero.

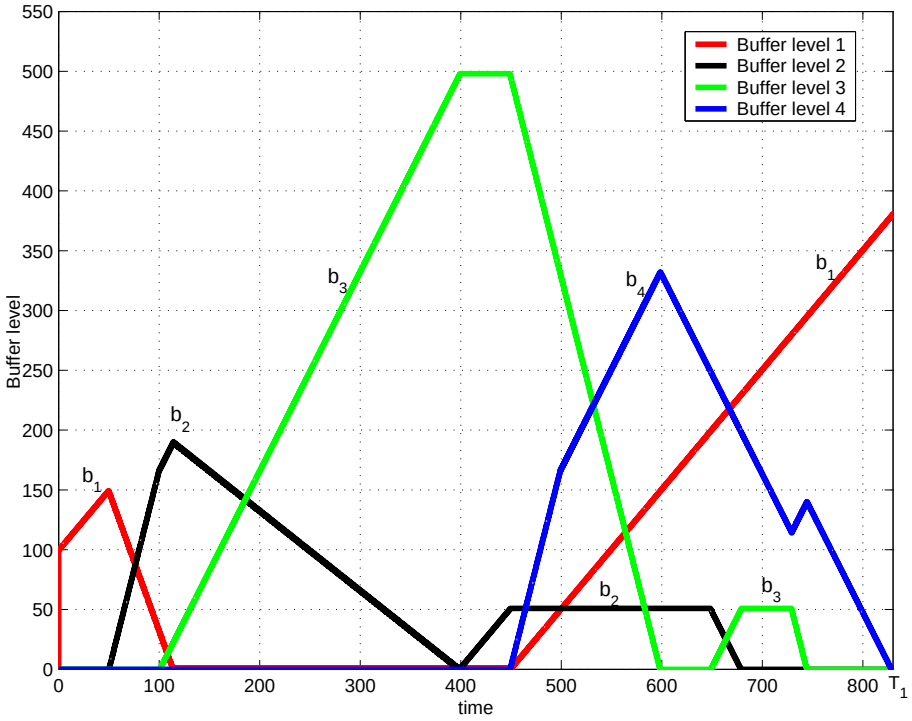

Figure 5.2: The behavior of the system of Example 5.1.

Example 5.2. *Consider the system as defined in Example 5.1, but with setup times equal to zero:*

$$\theta_1 = 0 \quad \theta_2 = 0 \ .$$

In Figure 5.3 the behavior of the system is depicted. The system has a behavior similarly to that of Example 5.1. At a time $t = T_1 = ((\alpha + \mu^c_{1,2}/(1 - \mu^c_{1,2}))\xi + \gamma$ the system returns to the same state as at time $t = 0$. Machine one and two are setup for the buffers 4 and 3 respectively and $(x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}) = (\alpha\xi + \beta, 0, 0, 0) = (150, 0, 0, 0)$. For this system $\alpha = 1.5$, $\beta = 0$ and $\gamma = 0$.

The system is again unstable, in spite of the setup time being equal to zero. It can be shown that a non-acyclic system can be unstable even when no part type ever revisits a machine, as in Figure 2.1, or that stability depends on the initial conditions [Kum90]. The instability results from the fact that the buffers for $i > 1$ can have instantaneous growth rates temporally exceeding the capacity of the machine. These grow rates result in large buffer levels and thus lead to very long production runs, which will block the other buffers at that machine, starving other machines. The cycles present in a non-acyclic system causes periods of overflow and starvation, thus resulting in instability.

5.1.1 Sufficient stability condition

In the previous section it was shown that a non-acyclic system can be unstable while the capacity condition (2.9) is satisfied. This section introduces a sufficient stability condition

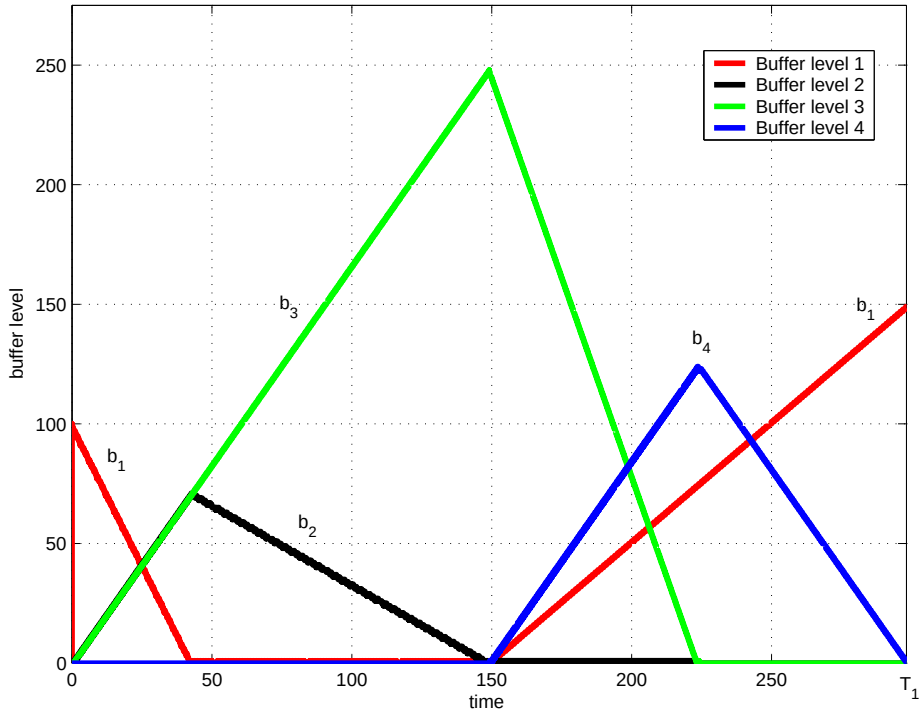

Figure 5.3: The behavior of the system of Example 5.2.

that guarantees the stability of a non-acyclic system for any DSI policy. In [Kum90] sufficient stability conditions for stability of non-acyclic systems using CAF policies are determined. However, the condition is only valid for non-acyclic systems in which parts do not flow from machine m directly back to m, a so called *self loop*. In [Hum94] a sufficient stability condition for any DSI policy is introduced. Thus, also for systems containing self loops.

To compute the sufficient stability condition of [Hum94] the following has to be defined. Define the *connection graph* of the system as a directed graph, where the nodes are the machines and the arc set $\mathcal{A}$ is:

$$\mathcal{A} := \{(m, m') | m \neq m' \text{ and } \exists (p, i); \alpha_{p,i} = m, \alpha_{p,i+1} = m'\}. \tag{5.1}$$

A machine m' is reachable from m if there is a directed path from m to m', which is written as $m \rightarrow m'$. The machines m and m' are *diconnected* if $m \rightarrow m'$ and $m' \rightarrow m$. Let $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_{\mathrm{K}}$ be the set of diconnected components of the connection graph. For all $(m, m') \in \mathcal{M}_{\mathrm{k}}$ there is a directed path from m to m' and back. For a buffer $b_{p,i}$ define $k_{p,i}$ as the index k for which $\alpha_{p,i} = m \in \mathcal{M}_{\mathrm{k}}$. There is a direct ordering on $\mathcal{M}$, because of the existence of directed paths from $m \in \mathcal{M}_{\mathrm{i}}$ to $m \in \mathcal{M}_{\mathrm{j}} (\mathrm{i} \neq \mathrm{j})$, which is written as $\mathcal{M}_{\mathrm{i}} \prec \mathcal{M}_{\mathrm{j}}$, or in short $i \prec j$. Thus a system is acyclic, if each $\mathcal{M}_{\mathrm{i}}$ consists of only one machine.

The system as depicted in Figure 5.1 is a clear example of two machines being diconnected. The parts flow from machine 1 to 2 and back thus there is a clear directed path present. The machines in Figure 2.1 are also diconnected. There is a path from machine 1 to 2 via the flow of part 1 and there is path from machine 2 to 1 via the flow of part 2. Thus, no clear ordering is present.

Definition 5.1. [Hum94] For each buffer, the *burst arrival rate* $\lambda'_{p,i}$ and the *conditional burst arrival rate* $\lambda''_{p,i}$ are defined as:

$$\lambda'_{p,i} = \begin{cases} \lambda_p & \text{if } i = 1 \\ \mu_{p,i-1} & \text{if } i \neq 1 \text{ and } \alpha_{p,i} \neq \alpha_{p,i-1} \\ \lambda'_{p,i-1} & \text{else } [\alpha_{p,i} = \alpha_{p,i-1}]. \end{cases} \tag{5.2}$$

$$\lambda''_{p,i} = \begin{cases} \lambda'_{p,i} & \text{if } i = 1 \text{ or } k_{p,i} = k_{p,i-1} \\ \lambda_p & \text{else } [k_{p,i-1} \prec k_{p,i}]. \end{cases} \tag{5.3}$$

The *conditional burst congestion level* [Hum94] is defined as:

$$\rho''_m = \sum_{b_{p,i} \in B_m} \frac{\lambda''_{p,i}}{\mu_{p,i}}. \tag{5.4}$$

A machine m is called *burst stable* [Hum94] if it satisfies the *burst capacity condition*:

$$\rho''_m = \sum_{b_{p,i} \in B_m} \frac{\lambda''_{p,i}}{\mu_{p,i}} < 1. \tag{5.5}$$

Definition 5.2. [Hum94] A system is stable under any DSI policy if each machine $m \in \mathcal{M}$ is burst stable, thus each $\rho''_m < 1$.

A non-acyclic system is guaranteed stable if condition (5.5) holds. However, it does not guarantee the system to be unstable if the condition does not hold. It can be shown that the system as depicted in Figure 5.1 is stable for certain parameters and initial conditions, for which condition (5.5) does not hold.

5.1.2 Regulators

In the previous section is was shown that non-acyclic systems can be unstable even when the capacity condition (2.9) holds. This section introduces a technique, called *regulators* [Per94]. These regulators modify a non-acyclic system such that it is stable for any DSI policy. A regulator splits a buffer in two (virtual) components, a regulator buffer and a regulated buffer, as depicted in Figure 5.4. The flow of parts from the regulator to the regulated buffer is restricted:

1. The cumulative input $\int_s^t u_{p,i}(t)dt$ to the regulated buffer $b_{p,i}$ satisfies

   $$\int_s^t u_{p,i}(t) \cdot dt \leq \hat{u}^c_{p,i} \cdot (t - s) + \hat{\gamma}_{p,i},$$

   in which $\hat{u}^c_{p,i}$ is a constant.

2. If buffer $b_{p,i-1}$ precedes $b_{p,i}$ then $\lambda_p \leq \hat{u}^c_{p,i-1} \leq \hat{u}^c_{p,i}$.

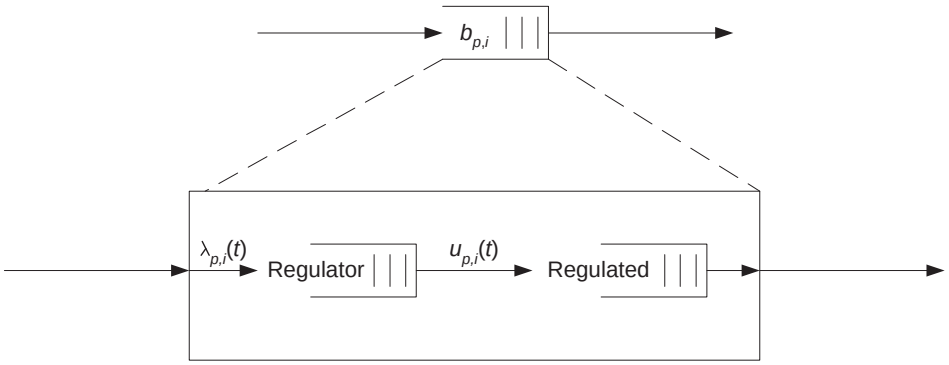

Figure 5.4: A regulated buffer.

3. For every machine m, the rates $\hat{u}^c_{p,i}$ satisfy the capacity constraint

$$\hat{\rho}_m = \sum_{b_{p,i} \in B_m} \frac{\hat{u}^c_{p,i}}{\mu_{p,i}} < 1.$$

Note that a regulator speed constant $\hat{u}^c_{p,i}$ that satisfies the conditions mentioned above is the arrival rate λ_p. A regulator is called smooth when $\hat{\gamma}_{p,i}$ equals zero and was introduced by [Hum94]. The use of smooth regulators can however result in an underutilization of the capacity of the machine. Non -smooth regulators can partially off set this problem [Per94].

The policy of Savkin is stable for both acyclic and non-acyclic systems, without the use of stabilizing techniques. The policy itself regulates the flow of parts throughout the system by limiting the time a machine processes parts from a buffer.

5.2 Simulation experiments

In this section simulation experiments are designed to compare the performance of CLSA, SAV^a and CLW^a. The results of the simulation experiments are presented and discussed in Section 5.3.

Experiment 6: An unstable non-acyclic system

In this experiment the performance of the three policies are compared with each other for a non-acyclic system with parameters such that the system is unstable under control of CLW^a or CLSA. In the previous section it was shown that a technique, called regulators, exists that stabilizes any DSI policy. Therefore, CLW^a and CLSA are equipped with regulators to stabilize them. Initially, smooth regulators are used and the regulator speeds are set equal to the average arrival rates, thus $\hat{\gamma}_{p,i} = 0$ and $\hat{u}^c_{p,i} = \lambda_p$. Variability is present in the process and arrival rates in the form of a triangular distribution. It is assumed that no variability is present in the regulator speed $u_{p,i}$. The system under consideration is similar to the one in Example 5.1. The process rates are however changed in order to set the average total load

factor (2.7) equal to the value used in the previous chapter, namely 0.8. For the process and arrival rates the same triangular distribution as in the previous chapter is used. The system is then defined as:

$$\begin{aligned}
\mathcal{P} &= \{1\} & \mathcal{M} &= \{1,2\} \\
\alpha_{1,1} &= 1 & \alpha_{1,2} &= 2 \\
\alpha_{1,3} &= 2 & \alpha_{1,4} &= 1 \\
l^c_{1,i}7 &= 0 & \forall i & \\
\lambda_1 &= 1 & & \\
\mu_{1,1} &= \frac{15}{4} & \mu_{1,2} &= \frac{15}{8} \\
\mu_{1,3} &= \frac{15}{4} & \mu_{1,4} &= \frac{15}{8} \\
\theta_1 &= 50 & \theta_2 &= 50.
\end{aligned}$$

The buffer levels are empty at the beginning of the simulation. Note that the capacity condition holds, since $\rho_m = 0.8 < 1$ for both machines. However, the burst capacity condition (5.5) does not hold, since $\rho''_1 = \frac{34}{15}$ and $\rho''_2 = 3$. Therefore, the system is not guaranteed to be stable for CLW^a and CLSA. According to [Kum90] for this system $\alpha = 8/7$, $\beta = 4010/21$ and $\gamma = 9560/21$. Thus, the system is not stable.

Experiment 7: A guaranteed stable non-acyclic system

The system of Experiment 6 did not satisfy the burst capacity constraint (5.5). In this experiment the performances of the three policies are compared with each other for a non-acyclic system, which is guaranteed stable under CLW^a or CLSA. The following non-acyclic system satisfies the burst capacity condition (5.5). The system is therefore guaranteed stable for any DSI policy.

$$\begin{aligned}
\mathcal{P} &= \{1\} & \mathcal{M} &= \{1,2\} \\
\alpha_{1,1} &= 1 & \alpha_{1,2} &= 2 \\
\alpha_{1,3} &= 2 & \alpha_{1,4} &= 1 \\
l^c_{1,i} &= 0 & \forall i & \\
\lambda_1 &= 1 & & \\
\mu_{1,1} &= \frac{10}{3} & \mu_{1,2} &= \frac{20}{3} \\
\mu_{1,3} &= \frac{100}{9} & \mu_{1,4} &= \frac{200}{9} \\
\theta_1 &= 50 & \theta_2 &= 50.
\end{aligned}$$

The buffer levels are empty at the beginning of the simulation. Note that both the capacity condition and burst capacity condition hold, since $\rho_1 = 0.345$, $\rho_2 = 0.24$ and $\rho''_m = 0.8$ for both machines. Two types of simulation are performed. One simulation where CLW^a and CLSA are not equipped with regulators and one simulation where they are equipped with regulators. Again SAV^a is not equipped with regulators, because SAV^a does not require them.

5.3 Simulation results and discussion

In Section 4.2 it is explained how the simulations are performed and how the results are depicted.

Experiment 6: An unstable non-acyclic system

The simulation results of Experiment 6 can be found in Table 5.1. The simulation tests confirm the instability for both policies. As mentioned earlier, regulators are added to stabilize the system for CLW^a and CLSA. No results are available for CLW^a and CLSA without regulators, since they are unstable. SAV^a is not equipped with regulators, since SAV^a is already stable in a non-acyclic system. It can been seen from the table that SAV^a does not perform well compared to CLW^a or CLSA. The average flow time is more than 2 times larger than that of CLW^a and CLSA. The large periods of idling present, result in an underutilization of the machine capacity and thus a bad performance. In section 3.3.1 it is explained that the idling is caused by the adaptation of SAV^a for a stochastic environment.

Regulators	Flow time			Percentile difference to CLSA	
	SAV^a	CLW^a	CLSA	SAV^a	CLW^a
No	3688.0	unstable	unstable	-%	-%
Yes	3688.0	1140.6	1141.9	223%	0%

Table 5.1: The results of Experiment 6.

Experiment 7: A guaranteed stable non-acyclic system

In Table 5.2 the results of the simulations are depicted. It can be seen from Table 5.2 that SAV^a performs better than the other policies with or without regulators. Applying the smooth regulators results in a larger average flow time for CLW^a and CLSA, thus in a larger percentile difference compared to SAV^a. A possible explanation for these results is the following. The system without regulators and with CLW^a or CLSA is stable. However, the non-acyclic production paths and the patient property of the policies result in an underutilization of the capacity of the machine. For example, buffer 3 is being emptied and buffer 2 is empty. Furthermore, machine one is working on buffer 4. When buffer 3 is empty, buffer 2 is still empty, because machine one has been working on buffer 4. Since the policy is patient the machine waits at the empty buffer 3, which will stay empty, because buffer 2 is empty. Machine two starts the setup for buffer 2 when machine one finally starts working on buffer 1. Smooth regulators only worsen the problem, by introducing more underutilization through more setups. SAV^a policy also under utilizes the machine capacity, which is caused by the adaption for an stochastic environment (see section 3.3.1). However, the underutilization present in SAV^a is, for this example, smaller than the underutilization present in the other policies.

Regulators	Flow time			Percentile difference to CLSA	
	SAV[a]	CLW[a]	CLSA	SAV[a]	CLW[a]
No	233.3	282.3	282.5	-17%	0%
Yes	233.3	469.0	473.9	-51%	-1%

Table 5.2: The results of Experiment 7.

5.4 Resumé

Although DSI policies can stabilize acyclic systems, they do not always stabilize non-acyclic systems. As has been demonstrated in this chapter the exhaustive policies CLW[a] and CLSA can be unstable in a non-acyclic system. However, techniques, such as regulators, exist that can stabilize these systems. The performance of the policies in a non-acyclic system depends on the parameters of the system. Contrary to the tested acyclic single machine system, CLSA does not always have the best performance. Furthermore, regulators can worsen the performance of an already stable non-acyclic system.

Chapter 6

Improving the Savkin policy

Chapters 2 and 3 introduced a flexible manufacturing system and three control policies which are suitable for controlling such a manufacturing system. These policies are the Savkin policy (SAV), the Clear the Largest Work policy (CLW) and the Clear the Largest Scaled Age policy (CLSA). In Chapters 4 and 5 it has been shown, by the use of simulation, that none of the policies has the overall best performance. It has also been shown that the Savkin policy is the only control policy which is guaranteed stable by itself for systems with non-acyclic production paths. This is a property that most control policies lack. Therefore, the question arises: is it possible to adapt the Savkin policy such that it has a best overall performance and still is stable for systems with non-acyclic production paths? To answer this question, first a better understanding of the Savkin policy is required. Therefore, this chapter starts with an analytical analysis of the Savkin policy. The analysis of the Savkin policy focusses on the levels of the buffers. The buffer levels are directly related to the average flow time of work present in the system. Thus, the buffer levels influence the performance of the system. Then, with help of this analysis it can be investigated if it is possible to improve the Savkin policy. The analysis is performed with the help of the logical-differential equations as introduced in Chapter 2. Thus, the flow of discrete parts is approximated as a continuous flow. Furthermore, it is assumed that no variability is present in the system. Techniques to analyze and improve this approximated system will provide a basis for analyzing and improving more realistic systems with variability and a discrete flow of parts. Note, that the original Savkin policy as introduced in Section 3.2 is used, since variability is not present.

6.1 Analyzing the Savkin Policy

6.1.1 The first buffer in a production path

The analysis of the Savkin policy starts by investigating the first buffer in which work arrives. Note, that T denotes the Savkin scheduling period where $T \geq T_0$. Furthermore, let j denote the j^{th} scheduling period from $t = (j-1)T$ until $t = jT$, where $j = 1, 2, 3, \ldots$.

Lemma 6.1. For buffer $b_{p,1}$ ($p \in \mathcal{P}$), which is the first buffer that work of type p encounters,

the following holds:

$$\begin{aligned} x_{p,1}(jT) &\geq x_{p,1}((j-1)T) \qquad \forall j = 1, 2, 3, \ldots \\ \lim_{j\to\infty} x_{p,1}(jT) &= x_{p,1}((j-1)T). \end{aligned}$$

In words, the buffer level of a buffer $b_{p,1}$ at the end of a scheduling period is equal or larger than the buffer level at the start of the scheduling period. Furthermore, when the number of scheduling periods goes to infinity the buffer level at the end of a scheduling period is equal or larger than the buffer level at the start of the scheduling period. Which means that the cumulative output from that buffer in one scheduling period equals the cumulative input.

Proof. Mass conversation implies the following for any buffer $b_{p,i}$ $(i \in \mathcal{I}_p)$:

$$x_{p,i}(jT) = x_{p,i}((j-1)T) + A^a_{p,i}((j-1)T, jT) - A^r_{p,i}((j-1)T, jT) \quad \forall j = 1, 2, \ldots, \tag{6.1}$$

in which $A^a_{p,i}((j-1)T, jT)$ denotes the cumulative input of buffer $b_{p,i}$ during the period $[(j-1)T, jT]$ and $A^r_{p,i}((j-1)T, jT)$ denotes the cumulative output from buffer $b_{p,i}$ during the period $[(j-1)T, jT]$. For the cumulative input of buffer $b_{p,1}$ the following holds:

$$A^a_{p,1}((j-1)T, jT) = \lambda^c_p T = d_p \qquad \forall j = 1, 2, 3, \ldots, \tag{6.2}$$

since work of type p arrives with a constant rate λ^c_p in the first buffer. For the cumulative output from this buffer the following holds:

$$\begin{aligned} A^r_{p,1}((j-1)T, jT) &\leq d_p \\ \lim_{j\to\infty} A^r_{p,1}((j-1)T, jT) &= d_p, \end{aligned} \tag{6.3}$$

for which the proof is delivered in [Sav98]. Lemma 6.1 now follows from (6.1), (6.2) and (6.3). □

Lemma 6.2. Let $t^j_{p,1}$ $(j \in \{1, 2, \ldots\},\ p \in \mathcal{P})$ be the time at which machine $\alpha_{p,1}$ starts with removing work from buffer $b_{p,1}$ during scheduling period j. Then, $A^r_{p,1}[(j-1)T, jT] = d_p$ if and only if:

$$x_{p,1}((j-1)T) \geq \frac{d_p}{\mu^c_{p,1}} \left(\mu^c_{p,1} - \lambda^c_p\right) - \lambda^c_p \left(t_{p,1} - (j-1)T\right). \tag{6.4}$$

In words, the cumulative output from buffer $b_{p,1}$ in scheduling period j equals the desired production level only if at the start of that scheduling period the level of buffer $b_{p,1}$ equals or is larger than then the amount as defined in (6.4).

Proof. Assume that $A^r_{p,1}((j-1)T, jT)$ equals the desired production level. The Savkin policy controls a workstation in such a way that during one scheduling period only one production run from buffer $b_{p,1}$ takes place. Furthermore, production from buffer $b_{p,1}$ always starts at the same moment in a scheduling period, thus $t^j_{p,1} = t^{j-1}_{p,1} + T$. Let $T^j_{p,1}$ be the time at which the production run from buffer $b_{p,1}$ in scheduling period j ends. Note, that a production run ends if the cumulative output in a scheduling period equals the desired production level or

when the buffer becomes empty. Then, $A^r_{p,1}((j-1)T, jT)$ can only be equal to the desired production level d_p if:

$$T^j_{p,1} - t^j_{p,1} = \frac{d_p}{\mu^c_{p,1}}. \tag{6.5}$$

In the period $[t^j_{p,1}, T^j_{p,1}]$ work is removed from buffer $b_{p,1}$ with a constant rate $\mu^c_{p,i}$. However, in that same period work also arrives in the buffer with a constant rate λ^c_p. Therefore, the effective rate at which work is removed during this period from buffer $b_{p,1}$ equals $\mu^c_{p,1} - \lambda^c_p$, then:

$$\begin{aligned} x_{p,1}(t^j_{p,1}) - x_{p,1}(T^j_{p,1}) &= \left(T^j_{p,1} - t^j_{p,1}\right)\left(\mu^c_{p,1} - \lambda^c_p\right) \qquad \textit{use (6.5)} \\ &= \frac{d_p}{\mu^c_{p,1}}\left(\mu^c_{p,1} - \lambda^c_p\right). \end{aligned} \tag{6.6}$$

This means that the cumulative output from buffer $b_{p,1}$ in one scheduling period equals the desired production level if and only if:

$$x_{p,1}(t^j_{p,1}) \geq \frac{d_p}{\mu^c_{p,1}}\left(\mu^c_{p,1} - \lambda^c_p\right). \tag{6.7}$$

In the period $[(j-1)T, t^j_{p,1}]$ work arrives with a constant arrival rate λ^c_p. With the result that the cumulative output from buffer $b_{p,1}$ in one scheduling period equals the desired production level d_p if and only if:

$$x_{p,1}((j-1)T) \geq \frac{d_p}{\mu^c_{p,1}}\left(\mu^c_{p,1} - \lambda^c_p\right) - \lambda^c_p\left(t^j_{p,1} - (j-1)T\right). \tag{6.8}$$

□

Assume that (6.4) holds for a scheduling period $j = \psi$ ($\psi \in \{1, 2, \ldots\}$). Then, it follows from Lemma 6.1 that Lemma 6.2 holds for any scheduling period $j \geq \psi$. Furthermore, from (6.1), (6.2) and (6.3) it follows that for any scheduling period $j \geq \psi$ that $x_{p,1}((j-1)T) = x_{p,1}(jT)$.

Lemma 6.3. Let ψ ($\psi \in \{1, 2, \ldots\}$) be a scheduling period for which (6.4) holds, then:

$$x_{p,1}(t+T) = x_{p,1}(t) \qquad \forall t \geq (\psi-1)T,\ \forall p \in \mathcal{P}. \tag{6.9}$$

In words, a buffer $b_{p,1}$ will exhibit periodic behavior for $t \geq (\psi-1)T$, thus for any scheduling period $j \geq \psi$.

Proof. It follows from Lemma 6.2 that for $j \geq \psi$

$$\begin{aligned} t^{j+1}_{p,1} &= t^j_{p,1} + T \\ T^{j+1}_{p,1} &= T^j_{p,1} + T, \end{aligned} \tag{6.10}$$

since the production period always starts at the same moment in a scheduling period, always has the same length and no variability is present. It can now be concluded from lemmas 6.1 and 6.2 that Lemma 6.3 holds. □

Proposition 6.1. For any buffer $b_{p,1}$ ($p \in \mathcal{P}$), which is the first buffer in a production path of a part type p, the following holds:

$$x_{p,1}(t+T) = x_{p,1}(t) \qquad \forall t \geq T \tag{6.11}$$
$$A^r_{p,1}[(j-1)T, jT] = d_p \qquad \forall j \geq 2 \tag{6.12}$$

In words, a buffer $b_{p,1}$ is guaranteed to exhibit periodic behavior for $t \geq T$. Furthermore, the cumulative output from a buffer $b_{p,1}$ for $t \geq T$ equals the desired production level.

Proof. It follows from lemmas 6.2 and 6.3 that Proposition 6.1 holds for any buffer $b_{p,1}$ for which:

$$x_{p,1}(0) \geq \frac{d_p}{\mu^c_{p,1}} \left(\mu^c_{p,1} - \lambda^c_p\right). \tag{6.13}$$

If $x_{p,1}(0)$ does not satisfy (6.13) then $A^r_{p,1}(0,T) \leq d_p$. Thus,

$$x_{p,1}(T^1_{p,1}) \geq 0 \tag{6.14}$$
$$T^1_{p,1} - t^1_{p,1} \leq \frac{d_p}{\mu^c_{p,1}}, \tag{6.15}$$

which means that for the cumulative input between the end of the production run in scheduling period 1 at $t = T^1_{p,1}$ and the start of the new production run in scheduling period 2 at $t = t^2_{p,1}$ the following holds

$$\begin{aligned} A^a_{p,1}(T^1_{p,1}, t^2_{p,1}) &= \left(t^2_{p,1} - T^1_{p,1}\right)\lambda^c_p && \textit{use } t^2_{p,1} = t^1_{p,1} + T \\ &= \left(t^1_{p,1} + T - T^1_{p,1}\right)\lambda^c_p \\ &= \left(T - \left(T^1_{p,1} - t^1_{p,1}\right)\right)\lambda^c_p && \textit{use } (6.15) \\ &\geq \left(T - \frac{d_p}{\mu^c_{p,1}}\right)\lambda^c_p && \textit{use } T = \frac{d_p}{\lambda^c_p} \\ &\geq \frac{d_p}{\mu^c_{p,1}}\left(\mu^c_{p,1} - \lambda^c_p\right). \end{aligned} \tag{6.16}$$

The level of the buffer at the start of the production run in scheduling period $j = 2$ becomes:

$$\begin{aligned} x_{p,1}(t^2_{p,1}) &= x_{p,1}(T^1_{p,1}) + A^a_{p,1}(T^1_{p,1}, t^2_{p,1}) \\ &\geq \frac{d_p}{\mu^c_{p,1}}\left(\mu^c_{p,1} - \lambda^c_p\right), \end{aligned} \tag{6.17}$$

The level of the buffer at the start of scheduling period $j = 2$ then becomes:

$$\begin{aligned} x_{p,1}(T) &= x_{p,1}(t^2_{p,1}) - A^a_{p,1}(T, t^2_{p,1}) \\ &\geq \frac{d_p}{\mu^c_{p,1}}\left(\mu^c_{p,1} - \lambda^c_p\right) - \lambda^c_p\left(t^2_{p,1} - T\right), \end{aligned} \tag{6.18}$$

which means that Proposition 6.1 holds according to lemmas 6.2 and 6.3. □

6.1.2 Analyzing the other buffers

In the previous section it was shown that the first buffer in a production path is guaranteed to exhibit periodic behavior for $t \geq T$. Furthermore, it was shown that the cumulative output from that buffer is guaranteed equal to the desired production level for $t \geq T$.

In this section the result of the previous section is extended to all the buffers in the system. Define for any part type $p \in \mathcal{P}$:

$$\begin{aligned} J_{p,i} &= 1 + J_{p,i-1} + \text{ceil}\left(\frac{l^c_{p,i-1}}{T}\right) \qquad \forall i = 1, 2, 3 \ldots, n_p \\ J_{p,0} &= 0 \\ l_{p,0} &= 0, \end{aligned} \tag{6.19}$$

in which the function ceil rounds up to the first integer. Note, that n_p denotes the last production step of work of type p.

Lemma 6.4. For any buffer $b_{p,i}$ ($i \in \mathcal{I}_p,\ p \in \mathcal{P}$) the following holds:

$$A^a_{p,i}((j-1), jT) = d_p \qquad \forall j \geq J_{p,i} \tag{6.20}$$

$$A^r_{p,i}((j-1), jT) = d_p \qquad \forall j \geq J_{p,i} + 1. \tag{6.21}$$

In words, the cumulative input of and cumulative output from buffer $b_{p,i}$ is guaranteed to be equal to the desired production level for a scheduling period $j \geq J_{p,i} + 1$.

Proof. Lemma 6.4 is proven with induction. From the previous section it can be seen that Lemma 6.4 holds for $i = 1$. Assume that Lemma 6.4 holds for buffer $b_{p,i}$. To complete the induction it needs to be shown that Lemma 6.4 also holds for $b_{p,i+1}$ The cumulative output from buffer $b_{p,i}$ during scheduling period $J_{p,i}$ is less than or equal to the desired production level d_p. The cumulative output is less than the desired production level if there is not enough work present in the buffer. As a result the following holds:

$$x_{p,i}(T^{J_{p,i}}_{p,i}) \geq 0 \tag{6.22}$$

$$T^{J_{p,i}}_{p,i} - t^{J_{p,i}}_{p,i} \leq \frac{d_p}{\mu^c_{p,i}}, \tag{6.23}$$

However, then work will still arrive after the end of the production run. Like for buffer $b_{p,1}$, see the proof of Proposition 6.1, the cumulative input of the buffer results in enough work to be present at the start of the new scheduling period $j = J_{p,i} + 1$ to guarantee that:

$$A^r_{p,i}((j-1), jT) = d_p \qquad \forall j \geq J_{p,i} + 1.$$

The work removed from buffer $b_{p,i}$ is transported with a transportation delay $l^c_{p,i}$ to buffer $b_{p,i+1}$. Note, that it is possible that this work arrives in $b_{p,i+1}$ in two successive scheduling periods if the transportation delay does not equal the length T of one scheduling period, see also Example 6.1. All the work that has been removed from a buffer $b_{p,i}$ in a scheduling period j is guaranteed to have arrived in buffer $b_{p,i+1}$ in scheduling period $j+\text{ceil}(l^c_{p,i}/T)$. This concludes the proof of Lemma 6.4 by induction, since the cumulative input in a scheduling

period of buffer $b_{p,i}$ is guaranteed equal to the desired production level for the scheduling period:

$$J_{p,i+1} = J_{p,i} + 1 + \text{ceil}\left(\frac{l^c_{p,i}}{T}\right).$$

□

Example 6.1. *This example illustrates a part of the proof of Lemma 6.4. In this example it is shown that if the transportation delay $l^c_{p,i}$ does not equal the length T of one scheduling period it is possible that work arrives in $b_{p,i+1}$ in two successive scheduling periods.*

Assume that in scheduling period j work is removed by machine $\alpha_{i,p}$ from buffer $b_{p,i}$ in the time period $[t^j_{p,i} T^j_{p,i}]$. This work is transported with a transportation delay of $l^c_{p,i}$ to the next buffer $b_{p,i+1}$. The work arrives in buffer $b_{p,i+1}$ during the time period $[t^j_{p,i} + l^c_{p,i}, T^j_{p,i} + l^c_{p,i}]$. Let $(jT - T^j_{p,i}) < l^c_{p,i} < T$, then work removed from buffer $b_{p,i}$ in period j arrives in buffer $b_{p,i+1}$ in the two successive scheduling periods j and $j+1$. The time periods in which work arrives in buffer $b_{p,i+1}$ are depicted in Figure 6.1 on a time line.

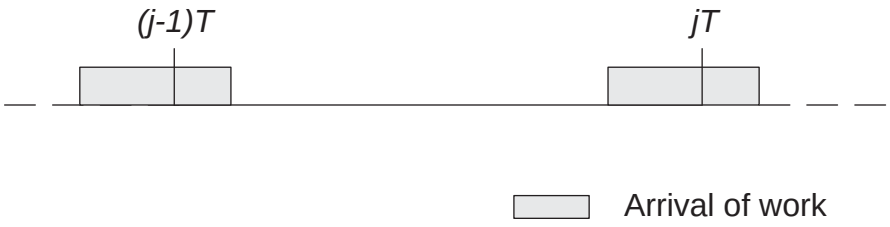

Figure 6.1: The time line of Example 6.1.

Proposition 6.2. For any buffer $b_{p,i}$ ($i \in \mathcal{I}_p,\ p \in \mathcal{P}$):

$$x_{p,i}(t+T) = x_{p,i}(t) \qquad \forall t \geq J_{p,i}T \tag{6.24}$$

$$A^r_{p,i}[(j-1)T, jT] = d_p \qquad \forall j \geq J_{p,i} + 1 \tag{6.25}$$

In words, a buffer $b_{p,i}$ is guaranteed to exhibit periodic behavior for $t \geq J_{p,i}T$. Furthermore, the cumulative output from buffer $b_{p,1}$ for $t \geq J_{p,i}T$ equals the desired production level. As a result the cumulative output from the system of part type p during a scheduling period is guaranteed equal to the desired production level for $t \geq J_{p,n_p}T$.

Proof. Define $[f^j_{p,i} F^j_{p,i}]$ as the time period in which work arrives in buffer $b_{p,i}$ where $f^j_{p,i}$ lays in scheduling period j. It follows from Lemma 6.4 that for $j \geq J_{p,i} + 1$:

$$\begin{aligned} t^{j+1}_{p,i} &= t^j_{p,i} + T \\ T^{j+1}_{p,i} &= T^j_{p,i} + T \\ f^j_{p,i} &= f^{j-1}_{p,i} + T \\ F^j_{p,i} &= F^{j-1}_{p,i} + T. \end{aligned} \tag{6.26}$$

It can now be concluded that Proposition 6.2 holds, since no variability is present. □

Lemma 6.5. For any buffer $b_{p,i}$ $(i \in \mathcal{I}_p,\ p \in \mathcal{P})$ the following holds

$$x_{p,i}(t) \geq \max[0, x_{p,i}(J_{p,i}T) - d_p] \qquad \forall t \geq J_{p,i}T. \tag{6.27}$$

In words, when a buffer exhibits periodic behavior for $t \geq J_{p,i}T$ then the minimal level of a buffer $b_{p,i}$ is at least the maximum of zero and $x_{p,i}(J_{p,i}T) - d_p$.

Proof. According to Lemma 6.4, the cumulative output from and the cumulative input of a buffer during a scheduling period $j \geq J_{p,i}+1$ are equal to the desired production level d_p. The level of a buffer is at its minimum if work is removed first before new work arrives. If this is the case then the buffer level at the start of the production run equals the buffer level at the start of the scheduling period. The minimal level of the buffer then equals $x_{p,i}(J_{p,i}T) - d_p$. However, work does not always arrive after the production run. Work can also arrive during of before the production run. As a result the minimal buffer level will be higher. The minimal level of the buffer can of course not become smaller than zero. □

From Lemma 6.5 it follows that the minimal level of a buffer is not guaranteed equal to zero.

Lemma 6.6. The level of a buffer $b_{p,i}$ at $t = J_{p,i}T$ is not independent of the levels of the buffers $b_{p,k}$ $(k \in \{= 1, 2, \ldots, i\})$ at $t = 0$. Thus, $x_{p,i}(J_{p,i}T)$ is not independent of the initial buffer levels of all the buffers that work of type p has visited.

Proof. Mass conversation implies the following:

$$x_{p,i}(J_{p,i}T) = x_{p,i}(0) + A^a_{p,i}[0, J_{p,i}T] - A^r_{p,i}[0, J_{p,i}T]. \tag{6.28}$$

In which for $i = 1$

$$A^a_{p,i}[0, J_{p,i}T] = d_p \tag{6.29}$$

and for $i \geq 2$

$$\begin{aligned} A^a_{p,i}[0, J_{p,i}T] &= A^r_{p,i-1}[0, J_{p,i}T - l_{p,i-1}] \\ &\leq A^r_{p,i-1}[0, J_{p,i}T] \end{aligned} \tag{6.30}$$

Furthermore,

$$A^r_{p,i}[0, J_{p,i}T] \leq \min\left[x_{p,i}(0) + A^a_{p,i}[0, J_{p,i}T], J_{p,i}d_p\right], \tag{6.31}$$

since the cumulative output of a buffer can not be larger than the level of the buffer at $t = 0$ plus the cumulative input or it can not be larger than the cumulative output limited by the Savkin policy. It can now be seen from (6.29), (6.30) and (6.31) that the level of the buffer at $t = J_{p,i}T$ is not independent of the buffer levels of the buffers $b_{p,k}$ $(k \in \{= 1, 2, \ldots, i\})$ at $t = 0$. □

Proposition 6.3. For any flexible manufacturing system under control of the Savkin policy the following holds for $t \geq \max\limits_{(p=1,2,\ldots,P)} J_{p,n_p}T$:

1. all buffers exhibit periodic behavior, thus:

$$x(t+T) = x(t) \tag{6.32}$$

2. the flow time of work of type p ($p \in \mathcal{P}$) is not independent of the initial buffer levels of the buffers $b_{p,i}$ ($i \in \mathcal{I}_p$) at $t = 0$.

3. the flow time of work of type p ($p \in \mathcal{P}$) has the lowest value if the minimal levels of all the buffers $b_{p,i}$ ($i \in \mathcal{I}_p$) are equal to zero. Thus the flow time of work of type p has the lowest value possible if the following holds:

$$\min(x_{p,i}(t)) = 0. \tag{6.33}$$

Proof. The proof for the first item follows directly from Proposition 6.2.

For the second item let φ_p denote the flow time of work of type p. Then the following holds for the flow time:

$$\begin{aligned} \varphi_p &\geq \sum_{i=1}^{n_p} \frac{\min\limits_{(t \geq J_{p,i}T)}(x_{p,i}(t))}{\mu^c_{p,i}} && \textit{use (6.27)} \\ &\geq \sum_{i=1}^{n_p} \frac{\max[0, x_{p,i}(J_{p,i}T) - d_p]}{\mu^c_{p,i}} \end{aligned} \tag{6.34}$$

From Lemma 6.6 it follows that $x_{p,i}(J_{p,i}T)$ is not independent of the of the buffer levels of all the buffers $b_{p,k}$ ($k \in \{=1, 2, \ldots, i\}$) at $t = 0$. Thus, it can be concluded that the flow time of work of type p is not independent of the initial buffer levels of all the buffers $b_{p,i}$ ($i \in \mathcal{I}_p$).

If the minimal level of a buffer is larger than zero there is always an unnecessary constant amount of work present in a buffer. This can also be seen from Figure 6.2. The upper line depicts the periodic behavior of a buffer in a system for $t \geq \max_{(p=1,2,\ldots,P)} J_{p,n_p}T$. The buffer level of the buffer never drops below c^1. This minimal amount of work that is always present in the buffer is not necessary to ensure the periodic behavior and if this amount of work would be removed manually the same behavior will still be present. However, the minimal buffer level will now be equal to zero as depicted with the bottom line in the figure. Removing this work results in a smaller flow time, since work is now not unnecessary waiting in the buffer. □

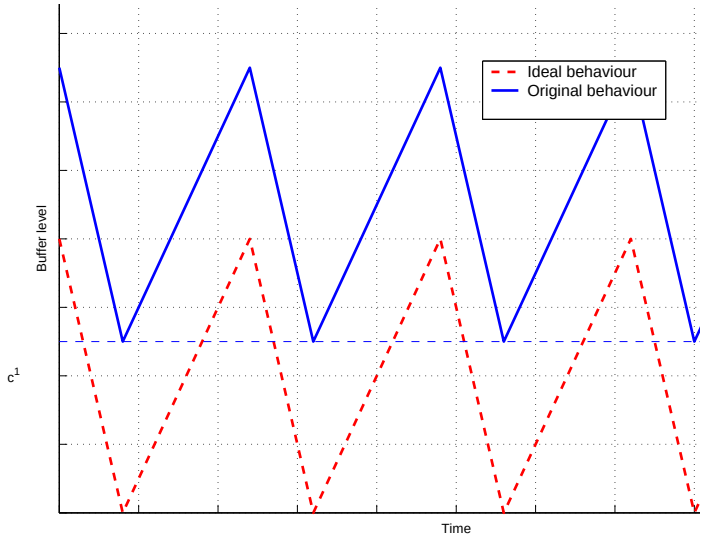

Figure 6.2: The behavior of a buffer under control of the Savkin policy.

6.1.3 Resumé

From Proposition 6.3 it can be concluded that the Savkin policy is not optimal with respect to minimizing the average flow time. Furthermore, the average flow time of a system under control of the Savkin policy is not independent of the initial buffer levels. A right choice of initial buffer levels will result in the lowest flow time possible for a manufacturing system controlled by the policy of Savkin. However, it is often not possible to arrange that a buffer has a specified initial buffer level. Therefore, the question arises: for a manufacturing system controlled by the Savkin policy, is it possible to adapt the system in such a way that the average flow time is minimized? This question is investigated in the next section.

6.2 Improving the Savkin policy for a simple case

In the previous section it has been shown that applying the Savkin policy on an approximated flexible manufacturing system does not guarantee the lowest average flow time possible for the Savkin policy. Furthermore, it has been shown that the average flow time is not independent of the initial buffer levels. A wrong choice of initial buffer levels results in the minimal level of a buffer being larger than zero. Thus, there is always a minimal amount of work present in a buffer, which results in an increased average flow time. A right choice of initial buffer levels results in the minimal level of a buffer being equal to zero. Then, no unnecessary work is present and this results in the lowest average flow time possible for the Savkin policy. In this section it is investigated if it is possible to improve the Savkin policy such that the minimal level of a buffer equals zero, regardless of the initial buffer levels. In other words, is it possible to adapt the Savkin policy such that the resulting average flow time is as low as possible for a system under control of the Savkin policy? The parameters of the Savkin policy that may be

adapted are the lengths of the production runs and the lengths of the setups. It is assumed that a machine can idle by extending a setup.

First, it is investigated if it is possible to improve the Savkin policy for a simple case. This case consists of only one machine processing two different part types. Furthermore, the parts in the systems are approximated as a continuous flow and it is assumed that no variability is present.

The buffer levels in a flexible manufacturing system under control of the Savkin policy are guaranteed to exhibit periodic behavior within a limited amount of time, see also Proposition 6.3. The periodic behavior stays the same for any initial buffer level. The initial buffer levels only influence the 'height' at which this period behavior takes place, see also Figure 6.2. For a simple two buffer case it is possible to depict the behavior of the buffers in a phase diagram.

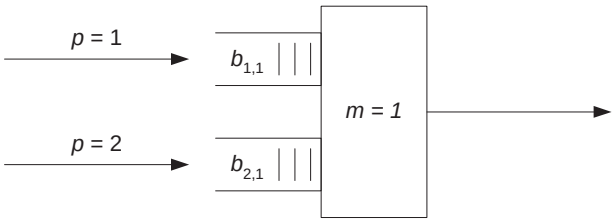

Figure 6.3: The flexible manufacturing system of Example 6.2.

Example 6.2. *Consider the flexible manufacturing system depicted in Figure 6.3 and described as:*

$$\begin{aligned} \mathcal{P} &= \{1,2\} & \mathcal{M} &= \{1\} \\ \alpha_{1,1} &= 1 & \alpha_{2,1} &= 1 \\ \lambda_1^c &= 1 & \lambda_2^c &= 2 \\ \mu_{1,1}^c &= 3 & \mu_{2,1}^c &= 6 \\ \theta_1 &= 2. \end{aligned}$$

The minimal length of the Savkin scheduling period equals:

$$T_0 = \frac{2 * 2}{1 - (\frac{1}{3} + \frac{2}{6})} = 12.$$

The scheduling period is set equal to the minimal scheduling period, thus $T = T_0$. Furthermore, for the desired production levels the following holds:

$$d_1 = \lambda_1 T_0 = 12, \quad d_2 = \lambda_2 T_0 = 24. \tag{6.35}$$

At time $t = 0$ the machine is ready to start processing from buffer $b_{1,1}$.

In Figure 6.4 two possible phase diagrams of the system as described in Example 6.2 are depicted. The horizontal axis represents the level of buffer $b_{1,1}$ and the vertical axis represents the level of buffer $b_{2,1}$. The start position of a trajectory is denoted with $*$. The solid line

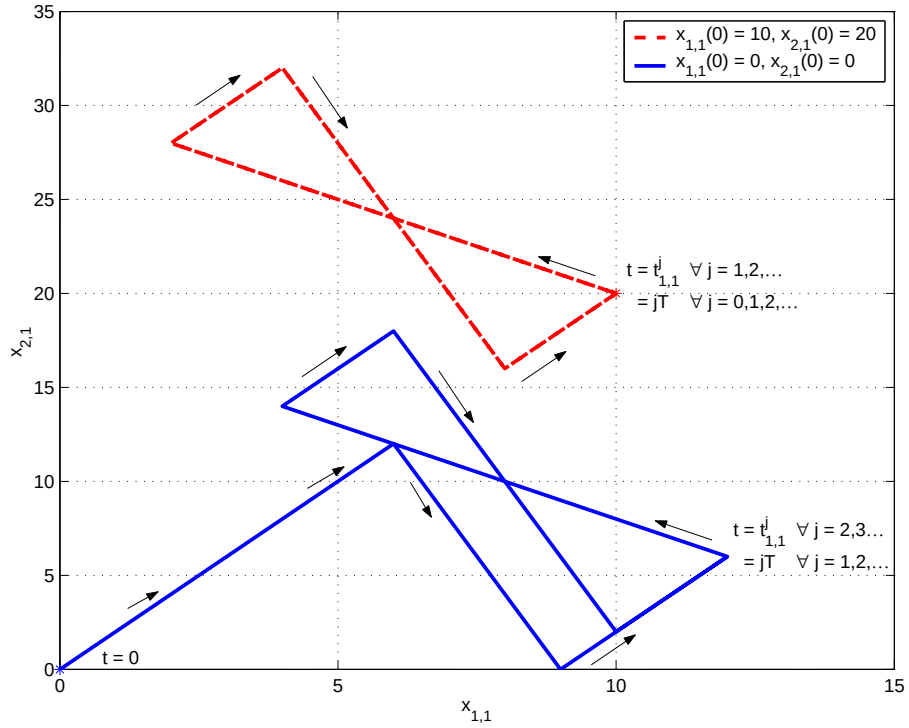

Figure 6.4: A phase diagram of Example 6.2 for different initial buffer levels.

depicts the trajectory of the buffer levels for empty buffers at $t = 0$. The dashed line represents the trajectory for the following initial buffer levels $x_{1,1}(0) = 10, x_{2,1}(0) = 20$. It can been seen that the trajectory of the buffer levels for the empty buffers at at $t = 0$ converges within one scheduling period T to an hourglass shape. For the initial buffer levels $x_{1,1}(0) = 10, x_{2,1}(0) = 20$ the trajectory immediately takes the form of an hourglass trajectory. The shape of this hourglass trajectory stays the same regardless of initial buffer levels. However, the position of this hourglass trajectory changes for different initial buffer levels. From Proposition 6.3 it follows that the average flow time of all the parts in the system is minimal if the minimal buffer levels equal zero. In the phase diagram this corresponds to the hourglass trajectory to be positioned as far in the lower left corner as possible. In Figure 6.5 a trajectory is depicted which results in a minimal average flow time for a two buffer machine system under control of the Savkin policy. Note that a buffer is now empty at the end of a production run. This trajectory, which results in a minimal average flow time for a two buffer machine system under control of the Savkin policy, will be denoted as the *ideal Savkin trajectory*.

Let the vector $x^{\mathrm{id}}(t) = [x^{\mathrm{id}}_{1,1}(t), x^{\mathrm{id}}_{2,1}(t)]$ denote the position on the ideal Savkin trajectory on the phase diagram at time t. Furthermore, let $x^{\mathrm{id}}(t^{\mathrm{id}}_{p,1})$ and $x^{\mathrm{id}}(T^{\mathrm{id}}_{p,1})$ denote the positions on the ideal Savkin trajectory at respectively the start and the end of a production run from buffer $b_{p,1}$. For this simple two buffer case the ideal Savkin trajectory can be computed easily, since there is a constant arrival rate and a buffer should be empty at the end of a production run. Thus, at $t = T^{\mathrm{id}}_{p,1}$ the buffer containing work of type p should be empty. The computed trajectory can be found in Table 6.1. As can be seen the level of the buffer containing work of type p equals zero at $t = T^{\mathrm{id}}_{p,1}$. The buffer level at $t = t^{\mathrm{id}}_{p,1}$ for the buffer containing work of

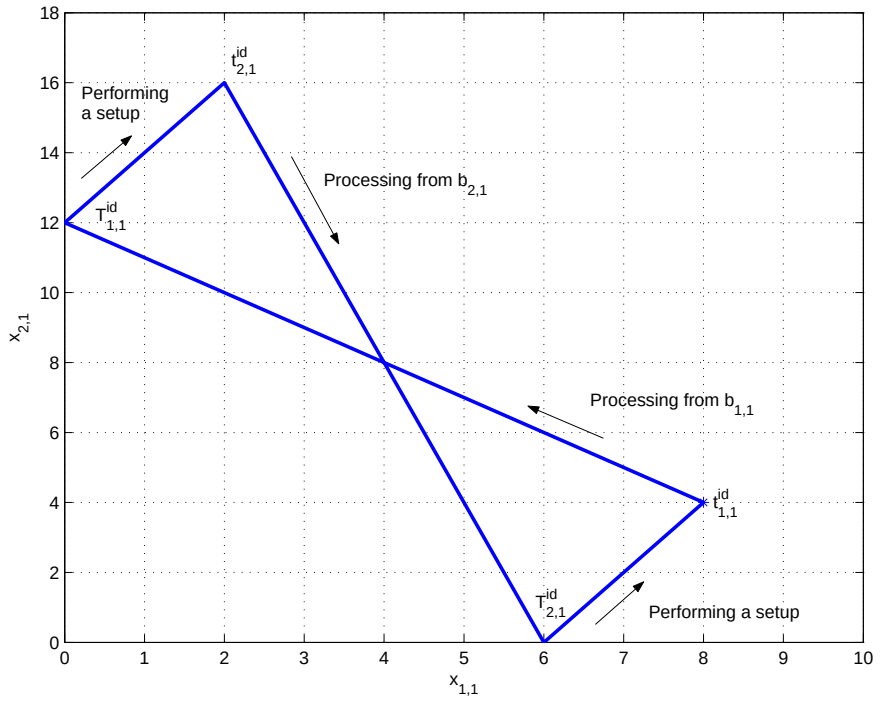

Figure 6.5: A phase diagram of Example 6.2 if minimal work is present in the buffers.

type p follows from (6.7). The levels of the buffer containing work of type p at time $s \leq t^{\mathrm{id}}_{p,1}$ can be computed by subtracting the cumulative input over the period $[s, t^{\mathrm{id}}_{p,1}]$ from the buffer level at time $t = t^{\mathrm{id}}_{p,1}$. These buffer levels are shown on the left side of the $x^{\mathrm{id}}_{1,1}(t)$ column and $x^{\mathrm{id}}_{2,1}(t)$ column. These values can be rewritten, which is shown on the right sides of the columns.

Table 6.1: The ideal Savkin trajectory.

t	$x^{\mathrm{id}}_{1,1}(t)$		$x^{\mathrm{id}}_{2,1}(t)$	
$t^{\mathrm{id}}_{1,1}$	$\frac{d_1}{\mu^c_{1,1}}(\mu^c_{1,1} - \lambda^c_1)$	$= \lambda^c_1 \left(2\theta_1 + \frac{d_2}{\mu^c_{2,1}}\right)$	$\frac{d_2}{\mu^c_{2,1}}(\mu^c_{2,1} - \lambda^c_2) - \lambda^c_2 \left(\theta_1 + \frac{d_1}{\mu^c_{1,1}}\right)$	$= \theta_1 \lambda^c_2$
$T^{\mathrm{id}}_{1,1}$	0	$= 0$	$\frac{d_2}{\mu^c_{2,1}}(\mu^c_{2,1} - \lambda^c_2) - \lambda^c_2 \theta_1$	$= \lambda^c_2 \left(\theta_1 + \frac{d_1}{\mu^c_{1,1}}\right)$
$t^{\mathrm{id}}_{2,1}$	$\frac{d_1}{\mu^c_{1,1}}(\mu^c_{1,1} - \lambda^c_1) - \lambda^c_1 \left(\theta_1 + \frac{d_2}{\mu^c_{2,1}}\right)$	$= \theta_1 \lambda^c_1$	$\frac{d_2}{\mu^c_{2,1}}(\mu^c_{2,1} - \lambda^c_2)$	$= \lambda^c_2 \left(2\theta_1 + \frac{d_1}{\mu^c_{1,1}}\right)$
$T^{\mathrm{id}}_{2,1}$	$\frac{d_1}{\mu^c_{1,1}}(\mu^c_{1,1} - \lambda^c_1) - \lambda^c_1 \theta_1$	$= \lambda^c_1 \left(\theta_1 + \frac{d_2}{\mu^c_{2,1}}\right)$	0	$= 0$

To minimize the average flow time the control goal of the improved Savkin policy becomes: guarantee that the trajectory of the buffer levels converges within a minimal amount of time to the ideal Savkin trajectory. Let $x(t^k_{1,1})$ denote the position of the improved Savkin policy at the start of cycle k. The idea for adapting the Savkin policy is the following. Assume that for every position on the phase diagram it is known what to do in order to satisfy the control goal. With this information the improved Savkin policy will be able to control a simple two

buffer case such that the control goal is satisfied. The expectation is that the phase diagram can be divided in different areas in which certain control actions should be taken. An example of such a division is given in Figure 6.6. The phase diagram is divided in three different areas. If positioned in the lower right area the machine should process from buffer $b_{1,1}$. In the middle area the machine should process from the buffer it is ready to process from. In the upper left area the machine should process from buffer $b_{2,1}$. The improved Savkin policy will be able to cope with variability if a division is possible, since at every position on the phase diagram it is known what the best action is.

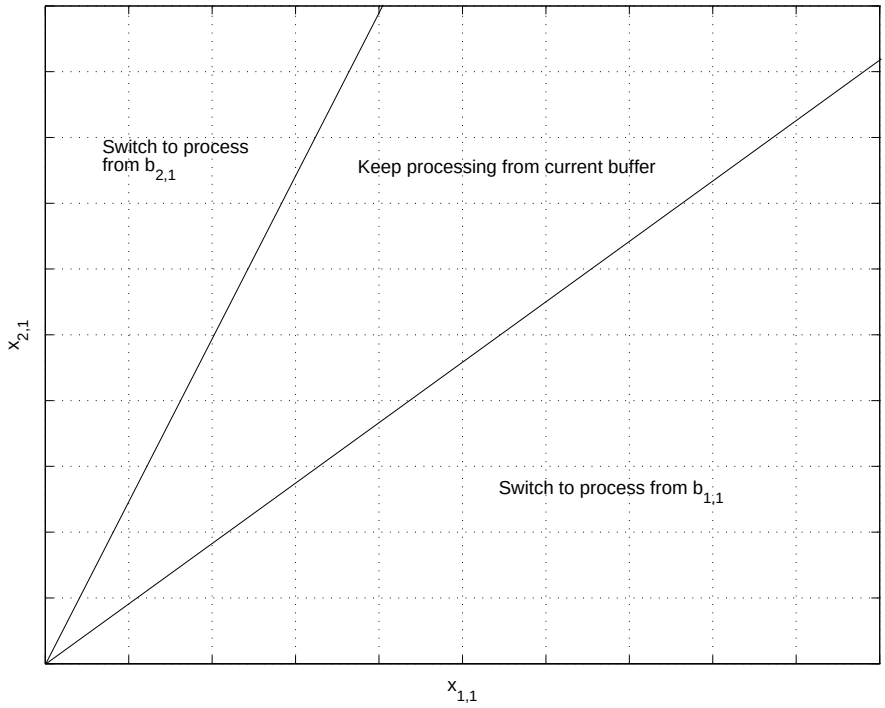

Figure 6.6: A phase diagram divided in different areas in which certain control actions should be taken.

6.2.1 Options for an improved Savkin policy

The previous section introduced a control goal for the improved Savkin policy. The control goal is defined as: guarantee that the trajectory of the buffer levels converges within a minimal amount of time to the ideal Savkin trajectory. This section discusses different options for an improved Savkin policy. The first option is a simple solution for the improved Savkin policy.

Cyclic clearing policy

Consider the following possible simple solution for the improved Savkin policy: process from $b_{1,1}$ until it is empty, setup for $b_{2,1}$, process from $b_{2,1}$ until it is empty, setup for $b_{1,1}$. Note

that this is a clearing policy, where the buffers are visited according to $\text{next}_{\mathbf{SAV}}$. This possible solution for the improved Savkin policy will be referred to as the *cyclic clearing policy*. Consider a line through the positions $x^{\text{id}}(T^{\text{id}}_{1,1})$ and $x^{\text{id}}(t^{\text{id}}_{1,1})$, denoted as a. Figure 6.7 depicts a. Note, that this line crosses the $x_{1,1}$ axis at $\left[(\frac{d_1}{\mu_1}+\theta)(\mu_1-\lambda_1), 0\right]$.

Proposition 6.4. Let $q_m(0) = b_{1,1}$, thus the machine is ready to process from buffer $b_{1,1}$. Then, for any initial position that does not coincide with a and when applying the cyclic clearing policy the resulting trajectory will converge to the ideal Savkin trajectory, but never coincide with it. For any initial position that does coincide with a the resulting trajectory will coincide with the ideal Savkin trajectory.

Proof. The proof of this proposition is given in Appendix B.1. Note that it can be seen from the proof in Appendix B.1 that Proposition 6.4 can easily be extended for $q_m(0) = b_{2,1}$. However, a will then run though the positions $x^{\text{id}}(T^{\text{id}}_{2,1})$ and $x^{\text{id}}(t^{\text{id}}_{2,1})$. □

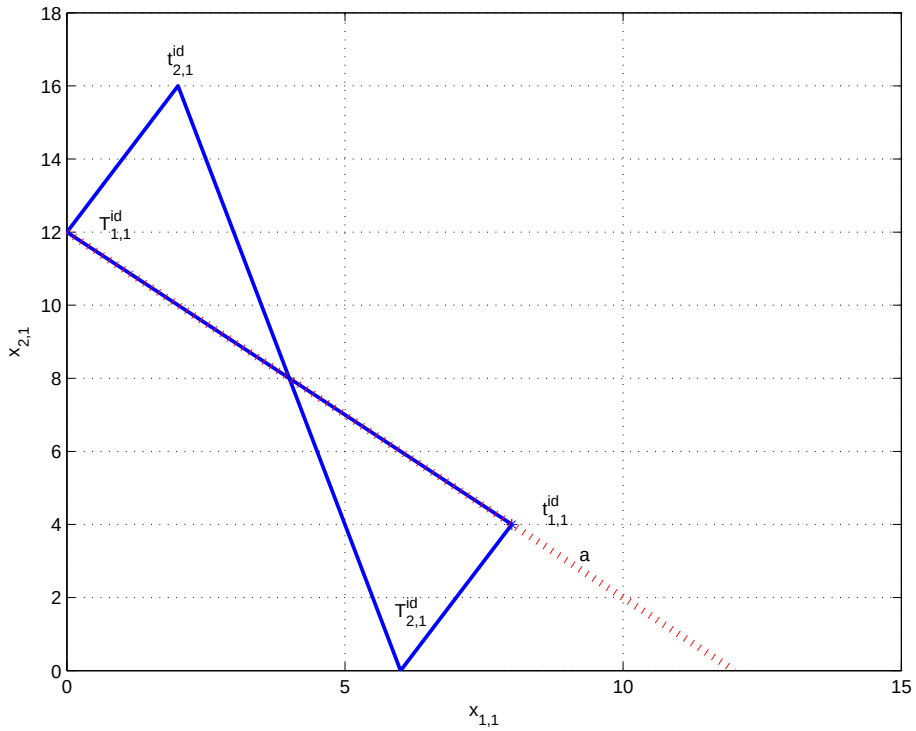

Figure 6.7: A phase diagram of Example 6.2 with a.

Searching for the optimal improved Savkin policy

The trajectory of the cyclic clearing policy only converges to the ideal Savkin trajectory, but never reaches it. Furthermore, it is not guaranteed that the resulting trajectory converges to the ideal Savkin trajectory in the smallest amount of time possible. Therefore, it is investigated if a better solution can be found. First, it is studied if the ideal Savkin trajectory can be reached and from which initial positions this is possible.

Proposition 6.5. Let $q_m(0) = b_{1,1}$, thus the machine is ready to process from buffer $b_{1,1}$. Then, the ideal Savkin trajectory can be precisely reached for any initial position that lays below or on a. The ideal Savkin trajectory can not be precisely reached for any initial position that lays above a.

Proof. The proof of this proposition is given in Appendix B.2. Note, in Appendix B.2 it is also shown from which initial positions the ideal Savkin trajectory can be reached if $q_m(0) = b_{2,1}$. □

In addition, the following proposition can be made.

Proposition 6.6. If a position on the phase diagram can be reached in K cycles without extending the setups then reaching that same position in $K + 1$ cycles will always take more time. Furthermore, if a position can be reached in K cycles without extending the setups then reaching that same position in K cycles with extending setups will always take more time.

Proof. The proof of this proposition is given in Appendix B.4. □

Initial positions below or on a

According to Proposition 6.5 the ideal Savkin trajectory can be reached from any initial position below or on a if $q_m(0) = b_{1,1}$.

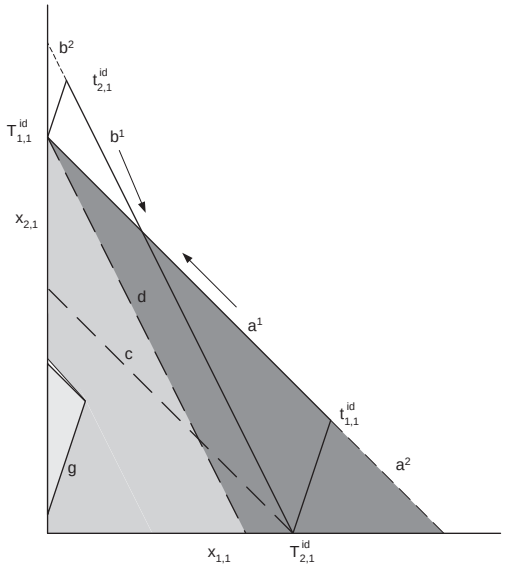

Figure 6.8: A possible phase diagram of the system depicted in Figure 6.3

Proposition 6.7. Consider the system depicted in Figure 6.3. Assume that the machine always follows the cycle: process from buffer $b_{1,1}$, setup for buffer $b_{2,1}$, process from $b_{2,1}$, setup for $b_{1,1}$. Furthermore, it is assumed that the time required to perform a setup when

the machine switches from $b_{1,1}$ to $b_{2,1}$, denoted with $\theta_{1\mapsto 2}$, does not have to equal the setup time when the machine switches from $b_{2,1}$ to $b_{1,1}$, denoted with $\theta_{2\mapsto 1}$. In Figure 6.8 a possible phase diagram of such a system is depicted. The structure if this phase diagram is explained in Appendix B.3. The phase diagram shows an ideal Savkin trajectory and the phase diagram is divided by lines and intervals in different areas. These areas all have a specific color. Let $q_m(0) = b_{1,1}$ and consider the following actions for the following different initial positions. For initial positions in the dark gray area process from $b_{1,1}$ work until d and then perform a setup to buffer $b_{2,1}$. For initial positions in the light gray area perform a setup to buffer $b_{2,1}$ and idle at $b_{2,1}$ to b^1. For initial positions in the very light gray area perform a setup to buffer $b_{2,1}$, perform a setup to buffer $b_{1,1}$ and idle at $b_{1,1}$ until a^1. And for initial positions in the white area perform a setup to buffer $b_{2,1}$, process work from $b_{2,1}$ until c, perform a setup to $b_{1,1}$. These actions, summarized in Table 6.2, result in the fastest trajectory to the ideal Savkin trajectory.

Proof. The proof of this proposition is given in Appendix B.3. Note, in Appendix B.3 it is also shown which action should be taken to reach the ideal Savkin trajectory in the smallest amount of time if $q_m(0) = b_{2,1}$. □

Table 6.2: Actions for any initial position with $q_m(0) = b_{1,1}$ below a, as depicted in Figure 6.8, that result in the fastest trajectory to the ideal Savkin trajectory.

Dark gray	process from $b_{1,1}$ work until d, perform a setup to buffer $b_{2,1}$
Light gray	perform a setup to buffer $b_{2,1}$ and idle at $b_{2,1}$ to b^1
Very light gray	perform a setup to buffer $b_{2,1}$, perform a setup to buffer $b_{1,1}$ and idle at $b_{1,1}$ until a^1
White	perform a setup to buffer $b_{2,1}$, process work from $b_{2,1}$ until c, perform a setup to $b_{1,1}$

Initial positions above a

According to Proposition 6.5 the ideal Savkin trajectory can not be reached from any initial position above a if $q_m(0) = b_{1,1}$. To measure if the improved Savkin policy has converged to the ideal Savkin trajectory the position $x^{\mathrm{id}}(t^{\mathrm{id}}_{1,1})$ of the ideal Savkin trajectory is compared with the start position $x(t^k_{1,1})$ of a cycle k of the improved Savkin policy. A possible option for the improved Savkin policy is the cyclic clearing policy, which was introduced earlier. However, it is not guaranteed that the cyclic clearing policy converges to the ideal Savkin trajectory in the smallest amount of time possible. The problem of finding the fastest route to some point can be formalized as an optimization problem. Therefore, an optimization algorithm is used in order to find an answer to this problem. First, the optimization problem needs to be correctly defined in order to use an optimization algorithm to solve it.

In Appendix B.5 it is shown for positions above a that idling the machine never results in faster convergence to the ideal Savkin trajectory than not idling the machine. Thus, increasing the lengths of the setups is never necessary in order to approach the ideal Savkin trajectory when

the initial positions lay above a. Therefore, the system variables that need to be optimized are only the lengths of the production runs. The optimization problem is defined as follows. Study if it is possible to sufficiently approach the ideal Savkin trajectory in one cycle. The trajectory has sufficiently approached the ideal Savkin trajectory when the distance between $x(t^k_{1,1})$ and $x^{\mathrm{id}}(t^{\mathrm{id}}_{1,1})$ is smaller than or equal to ϵ. If it is possible to reach the ideal Savkin trajectory in one cycle then minimize the amount of time necessary to reach the ideal Savkin trajectory. If the ideal Savkin trajectory can not be reached in one cycle study if it is possible to sufficiently approach the ideal Savkin trajectory in two cycles and minimize the amount of time necessary. If the ideal Savkin trajectory again can not be reached study if it is possible to sufficiently approach the ideal Savkin trajectory in three cycles and minimize the amount of time necessary. The number of cycles is increased until the ideal Savkin trajectory can be sufficiently approached and the system variables are optimized such that the amount of time necessary the reach the ideal Savkin trajectory is minimal. In Appendix B.4 it is shown that if a position can be reached in K cycles without extending the setups then reaching that same position in $K+1$ cycles will always take more time. Thus, a minimal number of cycles to sufficiently approach the ideal Savkin trajectory results in the fastest convergence to the ideal Savkin trajectory.

Note, the number of system variables increases with every cycle. For example, for the first cycle there are only two variables: the length of the production run from $b_{1,1}$ and the length of the production run from $b_{2,1}$. The second cycle again introduces two variables, thus there are now four variables which needs to be optimized. The lengths of the production runs are limited, because a buffer level can not become negative. For this simple case let D^k_p denote the length of the production from buffer $b_{p,1}$ during cycle k. Then, the following holds:

$$D^k_p \leq \frac{x_{p,1}(t^k_{p,1})}{\mu^c_{p,1} - \lambda^c_p}, \tag{6.36}$$

since a buffer level can not become negative. The level of a buffer a time t equals the level of the buffer at the start plus the cumulative input and minus the cumulative output. For this simple case the cumulative input equals the arrival rate times the time period. The cumulative output equals the processes rate times the lengths of the production runs. Then, (6.36) becomes:

$$D^k_p \leq \frac{x_{p,1}(0) + \lambda^c_p t^k_{p,1} - \mu^c_{p,1} \sum_{v=1}^{k-1} D^v_p}{\mu^c_{p,1} - \lambda^c_p}. \tag{6.37}$$

Note, that $t^k_{p,1}$ denotes the time at the start of the production run D^k_p and equals:

$$t^k_{p,1} = ((k-1)P + p - 1)\,\theta_1 + \sum_{v=1}^{k-1}\sum_{w=1}^{P} D^v_w + \sum_{w=1}^{p-1} D^k_w. \tag{6.38}$$

The length of a production run from buffer $b_{p,1}$ during cycle k is linearly dependent on the lengths of all the previous production runs from all the buffers at that machine. Appendix B.6 shows how (6.37) together with (6.38) can be rewritten in the form:

$$\mathbb{A}D \leq s. \tag{6.39}$$

In which D is a vector containing all the lengths of the production runs and $\leq$ is understood component wise. The size of $\mathbb{A}, D$ and s depends on the number of cycles.

The trajectory has sufficiently approached the ideal Savkin trajectory when the distance between $x(t_{1,1}^k)$ and $x^{\text{id}}(t_{1,1}^{\text{id}})$ is smaller than or equal to ϵ or formalized:

$$\|x^{\text{id}}(t_{1,1}^{\text{id}}) - x(t_{1,1}^k)\|_2 = \sqrt{\sum_{w=1}^{P} \left(x_{w,1}^{\text{id}}(t_{1,1}^{\text{id}}) - x_{w,1}(t_{1,1}^k)\right)^2} \leq \epsilon. \tag{6.40}$$

The position on the phase diagram after k cycles at $t = t_{1,1}^{k+1}$ is defined as:

$$x(t_{1,1}^{k+1}) = \begin{bmatrix} x_{1,1}(t_{1,1}^{k+1}) \\ x_{2,1}(t_{1,1}^{k+1}) \end{bmatrix} = \begin{bmatrix} x_{1,1}(0) \\ x_{2,1}(0) \end{bmatrix} + t_{1,1}^{k+1} \begin{bmatrix} \lambda_1^c \\ \lambda_2^c \end{bmatrix} - \begin{bmatrix} \mu_{1,1}^c \sum_{v=1}^{j} D_1^v \\ \mu_{2,1}^c \sum_{v=1}^{j} D_2^v. \end{bmatrix} \tag{6.41}$$

From (6.41) it can be seen that (6.40) is non-linearly dependent on the system variables. A question that rises is: what size should ϵ be? Let $\mathfrak{R}$ be the area in which all positions lay that have a distance ϵ or less from $x^{\text{id}}(t_{1,1}^{\text{id}})$. Then, ϵ should be so small such that it is guaranteed that if a position in $\mathfrak{R}$ can be reached in k cycles that a different position in that same area $\mathfrak{R}$ can not be reached faster in $k+1$ cycles. In Appendix B.7 it is shown that the following should hold:

$$\epsilon < \frac{\mu_{1,1}^c \mu_{2,1}^c \theta_1}{\sqrt{{\mu_{1,1}^c}^2 + {\mu_{2,1}^c}^2}}, \tag{6.42}$$

to guarantee the above. The fastest route to the ideal Savkin trajectory can now be found in the following manner: For $k = 1$ cycle study if it is possible to solve the following optimization problem:

$$\begin{aligned} \min_{D} \quad & t_{1,1}^{k+1} = kP\theta_1 + \|D\|_1 \\ \text{s.t.}: \quad & \mathbb{A}D \leq s \\ & D \geq 0 \\ & \|x^{\text{id}}(t_{1,1}^{\text{id}}) - x(t_{1,1}^{k+1})\|_2 \leq \epsilon. \end{aligned} \tag{6.43}$$

If the solution to this problem does not exist, because the feasible space is empty, then study if it is possible to solve (6.43) for $k+1$ cycles. Continue to increase the number of cycles until a feasible solution exist. This solution is the fastest route to the ideal Savkin trajectory. Note, that the objective function, the first and second constraint are linearly dependent on the system variables and that the third constraint is not. Due to the non-linear constraint and the increasing number of system variables which each cycle it is difficult to determine if the problem has a convex solution space.

Comparing the two options

The two possible options for the improved Savkin policy: clearing or optimization, are compared with each other. The optimization problem is solved in Matlab with `fmincon`, which uses a sequential quadratic programming (SQP) method to solve the problem. The file used for comparing both versions in Matlab can be found in Appendix E. The optimization version should always return a trajectory which takes the same or less amount of time than the cyclic clearing policy to sufficiently converge to the ideal Savkin trajectory. The reason for this is that if clearing should be the optimal solution than the optimization version has to return that solution. It turns out that for various initial positions the optimization version returns a trajectory that takes more time than applying the cyclic clearing policy. This is probably caused by the solution space not being convex, because (6.40) is non-linearly dependent on the system variables. To partially overcome this problem, the optimization is given the cyclic clearing trajectory as an initial guess for a possible solution. Then, the optimization version returns a trajectory almost similar to the cyclic clearing policy and which is negligible faster than the cyclic clearing policy. The results of various optimization problems for different initial positions are depicted in Table 6.3. The system as defined in Example 6.2 and an ϵ of 0.001 were used for generating these results. The table also depicts the results of the cyclic clearing policy. The table shows the remaining distance between the trajectory and the ideal Savkin trajectory and the time to sufficiently approach the ideal Savkin trajectory. It can been seen from the table that remaining distance of the optimization algorithm always lays around 0.001, which is equal to ϵ. This is the maximal remaining distance allowed. Apparently the optimization algorithm is not capable of finding a trajectory that has a smaller remaining distance and takes less time. Therefore, the optimization algorithm tries to find the fastest trajectory which results in a remaining distance of ϵ. The remaining distance of the cyclic clearing policy is always smaller than or equal to ϵ. If the remaining distance would be larger than ϵ, then another cycle would be performed resulting in a shorter remaining distance. It can also be seen from the table that the optimization algorithm is only slightly faster than the cyclic clearing policy. Summarized, the optimization algorithm returns a trajectory which is slightly faster than the trajectory of the cyclic clearing policy. However, the remaining distance between the optimized trajectory and the ideal Savkin trajectory is always larger or equal the remaining distance of the cyclic clearing policy. To overcome this problem the remaining distance of the cyclic clearing policy is used as the ϵ value for the optimization problem. Then, the optimization version returns a trajectory very similar to the cyclic clearing policy. The small difference between the trajectories is probably caused by a numerical errors.

Table 6.3: The results of the cyclic clearing policy and the optimization algorithm.

Initial position	Remaining distance		required time	
	Cyclic clearing	Optimization	Cyclic clearing	Optimization
$[10, 10]$	$4.8828 \cdot 10^{-4}$	$9.9992 \cdot 10^{-4}$	88.9995	88.9990
$[57, 23]$	$2.5940 \cdot 10^{-4}$	0.001	166.4997	166.4989
$[100, 357]$	$4.2439 \cdot 10^{-4}$	0.001	388.4996	388.4989

It can be concluded that the optimization algorithm is not capable of finding a significant

better trajectory then the cyclic clearing policy. It can not be concluded that the cyclic clearing policy is the optimal solution for an improved Savkin policy. However, for initial positions above a the cyclic clearing policy is easy implementable and it can cope with variability easily. Note that for this cyclic clearing policy the upper and lower switching lines of Figure 6.6 respectively coincide with the $x_{2,1}$ and $x_{1,1}$ axis. Further research is necessary to find the optimal solution for the improved Savkin policy for initial positions above a.

6.3 Resumé

In the first section of this chapter it has been shown for an approximated flexible manufacturing system that the average flow time is not independent on the initial buffer levels if the line is under control of the Savkin policy. In the previous section it has been investigated for a simple two buffer case if it is possible to improve the Savkin policy such that the lowest average flow time possible is guaranteed for such an line for any initial buffer level. Phase diagrams of the buffer levels have been used for finding the best improvement for the Savkin policy. First the ideal Savkin trajectory on the phase diagram for the system under control of the Savkin policy has been determined. It has been shown for certain initial positions on the phase diagram, which correspond to the initial buffer levels, that it is possible to reach the ideal Savkin trajectory precisely. For these initial positions the fastest trajectory to the ideal Savkin trajectory has been determined. For the other initial positions it is only possible to approach the trajectory, but never possible to precisely reach it. For these initial positions a trajectory has been determined that converges to the ideal Savkin trajectory, namely the cyclic clearing policy. It has not been determined if this trajectory is the fastest way to converge to the ideal Savkin trajectory. However, it is the fastest trajectory available, since a significant faster trajectory has not been found.

Chapter 7

Conclusions and recommendations

7.1 Conclusions

This thesis tried to answer various questions, which were:

- Which control policies are available for controlling a flexible manufacturing system?
- Are these policies suitable for stochastic environments and, if not, can they be adapted?
- For acyclic systems, how do these policies perform and what is the influence of certain characteristics of the system on the performance?
- For non-acyclic systems, which techniques are available for stabilizing unstable control policies and how do they perform compared to the just recently developed stable switching policy [Sav03]?
- When the previous questions are answered, is it possible to point out one policy with the best performance and if not, is it possible to improve the most promising policy in order to increase its performance?

In this report the average flow time of all parts has been used as a performance measure for the control policies. In Chapter 2 it has been shown that there are various policies available to control a flexible manufacturing system. Three promising policies have been chosen, namely the Savkin policy (SAV), the Clear the Largest Work policy (CLW) and the Clear the Largest Scaled Age policy (CLSA).

In Chapter 3 these policies have been studied further. It has been shown that the Savkin policy can be unstable in a stochastic discrete environment. Therefore, the Savkin policy has been adapted, in order to introduce variability in a system with discrete parts without resulting in instability. The adapted version is denoted with SAV^a. SAV^a is only able to cope with a special stochastic distribution, namely a triangular distribution for the arrival and process rate. Also CLW has been slightly adapted to make a comparison with CLSA more fair. The adapted CLW version is denoted with CLW^a.

Simulation experiments have been setup and performed in Chapter 4 in order to compare the three policies with each other for a simple single machine system with acyclic production

paths. The results have shown that CLSA has the best performance for a single machine system with acyclic production paths.

It has been explained in Chapter 5 that the CLSA and CLW^a policy are not always stable in a non-acyclic system. A technique called regulators stabilizes these systems. However, the performance of these two policies in a non-acyclic system depends on parameters of the system. Contrary to the tested acyclic single machine system, CLSA does not always have the best performance. Furthermore, regulators can worsen the performance of an already stable non-acyclic system. Therefore, none of the policies with or without regulators has the best overall performance.

The Savkin policy is the only control policy which is guaranteed stable by itself for systems with non-acyclic production paths if the so-called capacity condition holds. This is a property that the other two control policies lack. Therefore, the Savkin policy has been studied further to see if it is possible to increase its performance. In Chapter 6 it has been proven for a manufacturing system with a continuous flow of work and no variability that the Savkin policy is not optimal with respect to minimizing the average flow time. It has also been shown that the average flow time of a system under control of the Savkin policy is not independent of the initial buffer levels. For a two buffer single machine system improvements for the Savkin policy have been found, such that the average flow time converges to the lowest value possible for such a system, regardless of the initial buffer levels. Finally, it has been proven for the two buffer single machine system controlled by the improved Savkin policy that for certain initial conditions the average flow time converges to its minimum in the smallest amount of time possible.

7.2 Recommendations

New questions and ideas for improvements have arisen during this research.

Variability

The CLSA and CLW policies are both capable of handling variability. So far, the Savkin policy is only capable of handling special kinds of limited variability distributions. This limits the suitability of the Savkin policy, since in practice the variability is often unlimited. Therefore, further research is necessary to explore if it is possible to make the Savkin policy suitable for non limited variability distributions. A possible solution was already introduced in Chapter 6 for the two buffer single machine system. The system can handle variability if for every position and corresponding machine state on the phase diagram it is known what the best action is to reach an optimal position and corresponding machine state. Another possible solution that needs further research is the following. In Chapter 3 it was shown that the Savkin policy controls a manufacturing system in such a way that the utilization equals one, which results in instability in a stochastic system. It needs to be studied if it is possible to measure the utilization realtime and to adapt the parameters of the Savkin policy such that the utilization is smaller than 1.

Regulators

In Chapter 5 regulators were introduced in order to stabilize the CLW and CLSA policy for a non-acyclic system. For the simulations the arrival rate has been used as a setting for the regulator speed and the simulations have been performed with smooth regulators. However, the arrival rate does not have to be the optimal setting for the regulator speed. Furthermore, non-smooth regulators can partially off set the problem of underutilization. It needs to be explored if it is possible to improve the regulators. The performance of the system with the improved regulators should be greater than or equal to the performance of the system without regulators. Maybe it is possible to use feedback to control the regulator speed.

Discretization

In Chapter 6 is has been tried to improve the performance of SAV for a two buffer single machine system with a continuous flow of work. However, in Chapter 3 it has been shown that the SAV policy can be unstable in a system with work that consists of discrete parts if a machine has to finish a part. Therefore, it needs to be investigated what the effects are of discrete parts instead of continuous work for the improved Savkin policy.

Larger systems

In Chapter 6 the Savkin policy has been improved for a single machine system with two buffers, which is an acyclic system. However, the Savkin policy is in particular interesting, because it is stable for non-acyclic systems. Therefore, it needs be explored if it is possible to improve the Savkin policy for non-acyclic systems. A possible improvement for the Savkin policy is changing $\text{next}_{\mathbf{SAV}}$, which determines from which buffer work is processed by the machine. The Savkin policy visits each buffer once in one scheduling period. However, this does not have to be the optimal sequence. For example, is it possible that a Savkin policy with $\text{next}_{\mathbf{CLSA}}$ results in a lower average flow time? Therefore, it needs to be investigated if it is possible to change the $\text{next}_{\mathbf{SAV}}$ and what the influence of this change is.

Bibliography

[Ban97] J. Banks and J.G. Dai. Simulation studies of multiclass queueing networks. *IEEE Transactions*, 29:213–219, 1997.

[Bur97] Kevin Burgess and Kevin M. Passino. Stable scheduling policies for flexible manufacturing systems. *IEEE Transactions on Automatic Control*, 42(3):420–425, March 1997.

[Coo98] B. Cooper, Shun-Chen Niu, and Mandyam M. Srinivasan. When does forced idle time improve performance in polling models. *Management Science*, 44(8):1079–1086, August 1998.

[Coo99] B. Cooper, Shun-Chen Niu, and Mandyam M. Srinivasan. Setups in polling models: Does it make sense to setup if no work is waiting? *Journal of applied probability*, 36:585–592, 1999.

[Ger86] Stanley B. Gershwin. Stochastic scheduling and setups in flexible manufacturing systems. In K. Stecke and R. Suri, editors, *Proceedings of the second ORSA/TIMS conference on flexible manufacturing systems: operations research models and applications*, pages 431–422, Amsterdam, The Netherlands, 1986.

[Hof02] A.T. Hofkamp and J.E. Rooda. *Chi toolset reference manual*. Technische Universiteit Eindhoven, Department of Mechanical Engineering, Systems Engineering Group, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, November 2002. http://se.wtb.tue.nl/.

[Hop96] J.W. Hopp and M.L. Spearman. *Factory Physics: Foundations of manufacturing management*. Irwin, 1996.

[Hum94] Carlos Humes Jr. A regulator stabilization technique: Kumar Seidman revisited. *IEEE transaction on automatic control*, 39(1):191–196, January 1994.

[Kum90] P.R. Kumar and Thomas I. Seidman. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control*, 35(3):289–298, March 1990.

[Kum93] P.R. Kumar. Re-entrant lines. *Queueing Systems*, 13:87–110, 1993.

[Kum95] P.R. Kumar and S.P. Meyn. Stability of queuing-networks and scheduling policies. *IEEE Transactions on Automatic Control*, 40(2):251–260, February 1995.

[Law00] Averill M. Law and W. David Kelton. *Simulation modeling and analysis*. Mc-Graw-Hill, 2000.

[Liu92] Zhen Liu and Philippe Nain. On optimal polling policies. *Queueing Systems*, 11:59–83, 1992.

[Lu91] Steve H. Lu and P.R. Kumar. Distributed scheduling based on due dates and buffers priorities. *IEEE Transactions on Automatic Control*, 36(12):1406–1416, December 1991.

[Mat02] The MathWorks, Inc., Natick, Massachusetts, USA. *Matlab reference guide, version 6*, 2002. http://www.mathworks.com.

[Mon99] Douglas C. Montgomery and George C. Runger. *Applied statistics and probability for engineers*. John Wiley & Sons, Inc., 1999.

[Ols99] Tava Lennon Olsen. A practical scheduling method for multiclass production systems with setups. *Management Science*, 45(1):116–130, January 1999.

[Per89] James R. Perkins and P.R. Kumar. Stable, distributed, real-time scheduling of flexible manufacting/ assembly / disassembly systems. *IEEE Transactions on Automatic Control*, 34(2):139–148, February 1989.

[Per94] James R. Perkins, Carlos Humes Jr., and P.R. Kumar. Distributed scheduling of flexible manufacting systems: stability and performance. *IEEE Transactions on Automatic Control*, 10(2):133–141, April 1994.

[Ros04] G. van Rossum. *Python tutorial*, May 2004. Available via `http://www.python.org/doc/current/tut/`.

[Sav98] Andrey V. Savkin. Regularizability of complex switched server queueing networks modelled as hybrid dynamical systems. *Systems & Control Letters*, 35:291–299, 1998.

[Sav00] Andrey V. Savkin and Alexey S. Matvev. Cyclic linear differential automata: a simple class of hybrid dynamical systems. *Automatica*, 36:727–734, 2000.

[Sav01] Andrey V. Savkin and Alexey S. Matvev. A switched server system of order n with all its trajectories converging to (n-1)! limit cycles. *Automatica*, 37:303–306, 2001.

[Sav03] Andrey V. Savkin. Optimal distributed real-time scheduling of flexible manufacturing networks modelled as hybrid dynamical systems. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, Maui, Hawaii, December 2003.

[Sei94] T.I. Seidman. 'first come, first served' can be unstable! *IEEE Transactions on Automatic Control*, 39(10):2166–2171, October 1994.

[Tak86] H. Takagi. *Analysis of polling systems*. The MIT press, 1986.

Appendix A

Proof of the main result of [Sav03]

The problem statement and the control policy of Section 3.2.1 are introduced in [Sav03]. However, in [Sav03] no proof is provided that the introduced control policy solves the problem statement. In [Sav98] it is proven that the same control policy solves a similar problem statement for a switched server queueing network. However, the network as described in [Sav98] differs slightly from the manufacturing system described [Sav03]. In this appendix it is shown that the proof provided in [Sav98] also holds for the flexible manufacturing system of [Sav03].

Let $T > 0$ be a given time and $\lambda^{\max} > 0$ a given known constant which is called the maximum arrival rate. An arrival rate is part of the class $\mathcal{R}_T$ if the following condition holds:

$$\frac{1}{T} \int_{(j-1)T}^{jT} \lambda(t)dt \leq \lambda^{\max} \quad \forall j = 1, 2, 3, \ldots \quad . \tag{A.1}$$

An arrival rate belongs to the class $\mathcal{R}_T^c$, which is a subclass of $\mathcal{R}_T$, if there exists a constant λ^c such that

$$\frac{1}{T} \int_{(j-1)T}^{jT} \lambda(t)dt = \lambda^c \quad \forall j = 1, 2, 3, \ldots \quad . \tag{A.2}$$

In words it means that arrival rates from the class $\mathcal{R}_T^c$ have a constant average in the period $[jT, (j+1)T]$ which is less than or equal to $\lambda^{\max}$.

Definition A.1. [Sav98] A closed loop system as described in Section 3.2.1, but with non-constant (inter) arrival rates is said to exhibit *regular* behavior with a period T, if the following conditions hold:

1. All the trajectories of the closed loop system with (inter)arrival rates from the class $\mathcal{R}_T$ and initial conditions (2.1) are bounded on $[0, \infty)$.

2. For all the trajectories of the closed loop system with (inter)arrival rates from the class $\mathcal{R}_T^c$ and initial conditions (2.1), the following two conditions are satisfied:

(a) There exist a vector function $\hat{q}(t)$ such that:
$\hat{q}(t+T) = \hat{q}(t) \quad \forall t \geq 0$ and
$\lim_{j\to\infty} \mathbf{mes}\{t \in [(j-1)T, jT] : \ q(t) \neq \hat{q}(t)\} = 0.$

(b) $\lim_{j\to\infty} [x((j-1)T) - x(jT)] = 0.$

Here **mes** H denotes the Lebesque measure of the set H.

Savkin provided a proof in [Sav98] that a similar closed loop system, with non-constant (inter) arrival rates (belonging to the class $\mathcal{R}_T$), exhibits regular behavior. He did this by proving that the conditions defined by definition A.1 were met.

The arrival rates of the raw parts in the problem statement of [Sav03] belong to the class $\mathcal{R}_T^c$, since the arrival rates are constant. The process rate $\mu_{p,1}$ of machine $\alpha_{p,1}$, which processes a raw part of type p, is constant. Therefore, the output rate $\dot{y}_{p,i}$ will have a constant average over a scheduling period T. The output rate of machine $\alpha_{p,1}$ equals the arrival rate of machine $\alpha_{p,2}$ for a part of type p. Therefore, the inter arrival rate in the described system also belongs to the class $\mathcal{R}_T^c$. It can be concluded that with arrival rates defined by (3.9) and the proposed feedback policy (3.4), (3.5) (3.6), (3.7), (3.8) the system under consideration will be regular with the production levels and the scheduling period. Notice, that the minimal scheduling period does not depend on the the fixed transportation delays $l_{p,i}^c$.

Appendix B

Various proofs belonging to Chapter 6

In this appendix proofs are given for the various statements and propositions of Chapter 6.

B.1 Cyclic clearing

The proof in this section is for a system without variability. Therefore, the notation ^{c} is left out for simplicity.
The figures of to this section are depicted in Appendix B.8.
Consider the system depicted in Figure B.1. Let $q_m(0) = b_{1,1}$, thus the machine is ready to process from buffer $b_{1,1}$. For this simple two buffer case it is possible to depict the behavior of the buffers in a phase diagram, see also Figure B.2 or Figure 6.7. Let a be a line through the points $x^{\mathrm{id}}(T^{\mathrm{id}}_{1,1})$ and $x^{\mathrm{id}}(t^{\mathrm{id}}_{1,1})$, this also depicted in Figure B.2. Note that this line crosses the $x_{1,1}$ axis at $[(\frac{d_1}{\mu_1} + \theta)(\mu_1 - \lambda_1), 0]$. In this section it is shown for any initial position that does not coincide with a and when applying the cyclic clearing policy that the resulting trajectory will converge to the ideal Savkin trajectory, but never coincide with it. Furthermore, it is shown that for any initial position that does coincide with a that the resulting trajectory will coincide with the ideal Savkin trajectory.

Consider an initial position at $t = t_0$ that does not coincide with a. This initial position is also depicted in Figure B.2. Assume that the system is under control of the cyclic clearing policy. Then, at $t = t_a$ buffer $b_{1,1}$ will be empty and the buffer level of buffer $b_{2,1}$ equals $x_{2,1}(t_a)$. The trajectory of the cyclic clearing policy until the next time that buffer $b_{2,1}$ is emptied can be computed:

At $t = t_a$:
$x_{1,1}(t_a) = 0$
$x_{2,1}(t_a) = x_{2,1}(t_a)$

At $t = t_b = t_a + \theta_1$:
$x_{1,1}(t_b) = x_{1,1}(t_a) + (t_b - t_a)\lambda_1$
$x_{2,1}(t_b) = x_{2,1}(t_a) + (t_b - t_a)\lambda_2$

At $t = t_c = t_b + \frac{x_{2,1}(t_b)}{\mu_{2,1}-\lambda_1}$:
$x_{1,1}(t_c) = x_{1,1}(t_b) + (t_c - t_b)\lambda_1$
$x_{2,1}(t_c) = 0$

At $t = t_d = t_c + \theta_1$:
$x_{1,1}(t_d) = x_{1,1}(t_c) + (t_d - t_c)\lambda_1$
$x_{2,1}(t_d) = x_{2,1}(t_c) + (t_d - t_c)\lambda_2$

At $t = t_e = t_d + \frac{x_{1,1}(t_d)}{\mu_{1,1}-\lambda_1}$:
$x_{1,1}(t_e) = 0$
$x_{2,1}(t_e) = x_{2,1}(t_d) + (t_e - t_d)\lambda_2.$

Working this out results for $x_{2,1}(t_e)$ in:

$$x_{2,1}(t_e) = c_a + c_b x_{2,1}(t_a) \tag{B.1}$$

In which:

$$c_a = \theta_1\lambda_2 \left(1 + \frac{2\lambda_1}{\mu_{1,1} - \lambda_1} + \frac{\lambda_1\lambda_2}{(\mu_{1,1} - \lambda_1)(\mu_{2,1} - \lambda_2)}\right) \tag{B.2}$$

$$c_b = \frac{\lambda_1\lambda_2}{(\mu_{1,1} - \lambda_1)(\mu_{2,1} - \lambda_2)}. \tag{B.3}$$

Working c_b out results in:

$$c_b = \frac{\lambda_1\lambda_2}{\mu_{1,1}\mu_{2,1} - \mu_{1,1}\lambda_2 - \mu_{2,1}\lambda_1 + \lambda_1\lambda_2}. \tag{B.4}$$

The capacity condition (2.9) implies the following:

$$\frac{\lambda_1}{\mu_{1,1}} + \frac{\lambda_2}{\mu_{2,1}} < 1.$$

Multiplying the left and right side with $\mu_{1,1}\mu_{2,1}$ results in:

$$\mu_{2,1}\lambda_1 + \mu_{1,1}\lambda_2 < \mu_{1,1}\mu_{2,1}. \tag{B.5}$$

From (B.4) and (B.5) it can be seen that:

$$0 < c_b < 1. \tag{B.6}$$

Let $x_{2,1}^z$ denote the z^{th} time that buffer $b_{2,1}$ becomes empty. It can be seen from (B.1) that the following holds:

$$x_{2,1}^{z+1} = c_a + c_b x_{2,1}^z \qquad \forall z \geq 1. \tag{B.7}$$

This can be rewritten:

$$x_{2,1}^{z+1} = \sum_{k=1}^{z} c_a c_b^{k-1} + c_b^z x_{2,1}^1 \qquad \forall z \geq 1. \tag{B.8}$$

Then:

$$\begin{aligned}\lim_{z\to\infty} x_{2,1}^{z+1} &= \sum_{k=1}^{\infty} c_a c_b^{k-1} + c_b^{\infty} x_{2,1}^1 \\ &= \frac{c_a}{1-c_b} \\ &= \frac{\lambda_2\theta_1\left(\mu_{1,1}\mu_{2,1} + \mu_{2,1}\lambda_1 - \mu_{1,1}\lambda_1\right)}{\mu_{1,1}\mu_{2,1} - \mu_{1,1}\lambda_2 - \mu_{2,1}\lambda_1}.\end{aligned} \tag{B.9}$$

From (3.19) is follows that:

$$\lambda_1 = \mu_{1,1} - \frac{2\theta_1\mu_{1,1}}{T_0} - \frac{\lambda_2\mu_{1,1}}{\mu_{2,1}}. \tag{B.10}$$

Then (B.9) can be rewritten with (B.10) into:

$$\begin{aligned}\lim_{z\to\infty} x_{2,1}^{z+1} &= \lambda_2 T_0 - \lambda_2\theta_1 - \frac{\lambda_2^2 T_0}{\mu_{2,1}} \\ &= \frac{\lambda_2 T_0}{\mu_{2,1}}(\mu_{2,1} - \lambda_2) - \lambda_2\theta_1 && d_p = \lambda_p T \\ &= \frac{d_2}{\mu_{2,1}}(\mu_{2,1} - \lambda_2) - \lambda_2\theta_1 \\ &= x_{2,1}^{\mathrm{id}}(T_{1,1}) && \textit{see Table 6.1}\end{aligned} \tag{B.11}$$

From (B.11) it can be concluded that for any initial position that does not coincide with a and when applying the cyclic clearing policy that the resulting trajectory will converge to the ideal Savkin trajectory, but never coincide with it. With (B.7) it is easy to show that for any initial position that does coincide with a that the resulting trajectory will coincide with the ideal Savkin trajectory. For any initial position that coincides with a it can be seen from Figure B.2 that the resulting trajectory with the cyclic clearing policy eventually will reach position $x_{2,1}^{\mathrm{id}}(T_{1,1}^{\mathrm{id}})$. Using $x_{2,1}^{\mathrm{id}}(T_{1,1}^{\mathrm{id}})$ for $x_{2,1}^{i}$ in (B.7) results in:

$$\begin{aligned}x_{2,1}^{i+1} &= c_a + c_b x_{2,1}^{\mathrm{id}}(T_{1,1}^{\mathrm{id}}) \\ &= \frac{\lambda_2\left(\mu_{2,1}\lambda_1\theta_1 + \mu_{2,1}\mu_{1,1}\theta_1 + \lambda_1\lambda_2 T_0\right)}{\mu_{2,1}(\mu_{1,1} - \lambda_1)} && \textit{use (B.10)} \\ &= \lambda_2 T_0 - \lambda_2\theta_1 - \frac{\lambda_2^2 T_0}{\mu_{2,1}} \\ &= \frac{\lambda_2 T_0}{\mu_{2,1}}(\mu_{2,1} - \lambda_2) - \lambda_2\theta_1 && d_p = \lambda_p T \\ &= \frac{d_2}{\mu_{2,1}}(\mu_{2,1} - \lambda_2) - \lambda_2\theta_1 \\ &= x_{2,1}^{\mathrm{id}}(T_{1,1}^{\mathrm{id}}) && \textit{see Tabel 6.1.}\end{aligned} \tag{B.12}$$

Thus, it can be concluded that for any initial position that does coincide with a that the resulting trajectory will coincide with the ideal Savkin trajectory.

B.2 Reaching the ideal Savkin trajectory

The proof in this section is for a system without variability. Therefore, the notation '^{c}' is left out for simplicity.
The figures of to this section are depicted in Appendix B.8.
Consider the system depicted in Figure B.1. It is assumed that the time required to perform a setup when the machine switches from $b_{1,1}$ to $b_{2,1}$, denoted with $\theta_{1\mapsto 2}$, does not have to be equal to the setup time when the machine switches from $b_{2,1}$ to $b_{1,1}$, denoted with $\theta_{2\mapsto 1}$. For this simple two buffer case it is possible to depict the behavior of the buffers in a phase diagram. In Figure B.3 a trajectory is depicted on a phase diagram which results in a minimal average flow time for a two buffer machine system under control of the Savkin policy. Note that a buffer is now empty at the end of a production run. This trajectory, which results in a minimal average flow time for a two buffer machine system under control of the Savkin policy, will be denoted as the *ideal Savkin trajectory*. In this section it is shown for the system depicted in Figure B.1 from which initial positions on the phase diagram the ideal Savkin trajectory can be reached.

Consider the interval on the ideal Savkin trajectory between the positions $x(t^{\text{id}}_{1,1})$ and $x(T^{\text{id}}_{1,1})$. From now on this interval will be indicated as a^1, as depicted in Figure B.4. The machine state has to be $q_m = b_{1,1}$ in order to follow the ideal Savkin trajectory when positioned on a^1. Thus, the machine has to be ready to process from buffer $b_{1,1}$. There are two areas on the phase diagram from which a^1 can be reached, such that $q_m = b_{1,1}$ when a^1 has been reached. When on a^1 $q_m = b_{1,1}$ can only hold if either the previous machine state q^{prev}_m equals: $q^{\text{prev}}_m = b_{1,1}$ or the machine was performing a setup to buffer $b_{1,1}$, denoted as $q^{\text{prev}}_m = b_{2,1} \mapsto b_{1,1}$. When $q^{\text{prev}}_m = b_{1,1}$ it can be seen from the phase diagram that a^1 can only be reached from a^2. The interval a^2 is an extension of a^1 and crosses the $x_{1,1}$ axis at $[(\frac{d_1}{\mu_{1,1}} + \theta_{2\mapsto 1})(\mu_{1,1} - \lambda_1), 0]$. When $q^{\text{prev}}_m = b_{2,1} \mapsto b_{1,1}$ the machine is performing a setup and this setup takes up $\theta_{2\mapsto 1}$ amount of time. Therefore, when $q^{\text{prev}}_m = b_{2,1} \mapsto b_{1,1}$ it can be seen from the phase diagram that a^1 can only be reached from c. Summarized, a^1 can be reached from two areas on the phase diagram, namely from the interval a^2 and line c if the machine is in the proper machine state.

Interval a^1 can be reached from all positions from where it is possible to reach c or a^2 with the proper machine state. It can be seen from the phase diagram that there are no positions, not on a^2, from where a^2 can be reached with the proper machine state. However, there are positions from where it is possible to reach c. At c a setup from buffer $b_{2,1}$ to buffer $b_{1,2}$ should commence. Thus, prior to that the machine had to be processing work from $b_{2,1}$. It can be seen from the phase diagram that for positions in the dark gray area c can be reached by processing work from buffer $b_{2,1}$. Furthermore, for positions in the light gray area c can be reached by idling the machine. Note that idling the machine results in the same trajectory as when a setup is performed. In this situation it can be seen as extending a setup.

Consider the interval on the ideal Savkin trajectory between the positions $x(t^{\text{id}}_{2,1})$ and $x(T^{\text{id}}_{2,1})$. From now on this interval will be indicated as b^1. In order to follow the ideal Savkin trajectory when positioned on b^1 the machine state has to be $q_m = b_{2,1}$. As for a^1 two areas can be determined from which b^1 can be reached with a proper machine state. The two areas are depicted in Figure B.5.

All initial positions from which it is possible to reach the ideal Savkin trajectory have now been determined. In Figure B.6 the area from which it is possible to reach the ideal Savkin

trajectory is depicted. It can be concluded that the ideal Savkin trajectory can never be reached from initial positions that do not lay within the gray area.

B.3 The optimal trajectory to the ideal Savkin trajectory

The figures of to this section are depicted in Appendix B.8.
Consider the system depicted in Figure B.1. Assume that the machine always follows the cycle: process from buffer $b_{1,1}$, setup for buffer $b_{2,1}$, process from $b_{2,1}$, setup for $b_{1,1}$. Furthermore, it is assumed that the time required to perform a setup when the machine switches from $b_{1,1}$ to $b_{2,1}$, denoted with $\theta_{1\mapsto 2}$, does not have to be equal to the setup time when the machine switches from $b_{2,1}$ to $b_{1,1}$, denoted with $\theta_{2\mapsto 1}$. Finally, it is assumed that a machine can idle by extending a setup. For this simple two buffer case it is possible to depict the behavior of the buffers in a phase diagram. In Figure B.3 a trajectory is depicted on a phase diagram which results in a minimal average flow time for a two buffer machine system under control of the Savkin policy. Note that a buffer is now empty at the end of a production run. This trajectory, which results in a minimal average flow time for a two buffer machine system under control of the Savkin policy, will be denoted as the *ideal Savkin trajectory*. In this section it is shown for initial positions from which it is possible to precisely reach the ideal Savkin trajectory what the fastest trajectory to the ideal Savkin trajectory is. In Section B.2 it has been shown from which initial positions and corresponding machine state the ideal Savkin trajectory can be reached.

Let $q_m(0) = b_{1,1}$, thus the machine is ready to process from buffer $b_{1,1}$. The initial positions from which the ideal Savkin trajectory can be reached are depicted in Figure B.5. For any initial position that lays between or on a and d, which is the dark gray area, the fastest way to precisely reach the ideal Savkin trajectory is: process from buffer $b_{1,1}$ until d has been reached, perform a setup to buffer $b_{2,1}$. It can be seen from the phase diagram in Figure B.5 that no other methods exist to reach to ideal Savkin trajectory from initial positions in the dark gray area, since idling is only allowed during a setup.

Consider e as depicted in Figure B.7. Line e runs parallel with d and they are $\theta_{2\mapsto 1}$ apart from one another. Let $q_m(0) = b_{1,1}$, for any initial position that lays between or on d and e, which is the light gray area. For an initial position with $q_m(0) = b_{1,1}$ in the light gray area there are three options to reach the ideal Savkin trajectory:

1. setup for $b_{2,1}$ and idle at $b_{2,1}$ until b^1.
2. setup for $b_{2,1}$, setup for $b_{1,1}$, if the resulting position falls within the dark gray area, process work from $b_{1,1}$ until d, setup for $b_{2,1}$. If the resulting position falls outside the dark gray the ideal trajectory can not be reached with this option.
3. setup for $b_{2,1}$, setup for $b_{1,1}$, if the resulting position falls within the dark gray area, idle at $b_{1,1}$ until a^1. If the resulting position falls outside the dark gray the ideal trajectory can not be reached with this option.

It can be seen from Figure B.7 that for option 1 the time needed to reach b^1 by idling is smaller than or equal to $\theta_{2\mapsto 1}$. Thus, for option 1 the time needed to reach b^1 is smaller than or equal to: $\theta_{1\mapsto 1} + \theta_{2\mapsto 1}$. The time needed to reach the ideal Savkin trajectory for option 2

and 3 is equal to or larger than: $\theta_{1\mapsto 1} + \theta_{2\mapsto 1}$, since both options exist of performing a setup for buffer $b_{2,1}$ and buffer $b_{1,1}$. Therefore, it can be concluded that fastest way to reach the ideal Savkin trajectory is option 1.

Consider g as depicted in Figure B.8. Interval g runs parallel with a line indicating a setup or idling and runs through the position where a^1 and b^1 cross. Let $q_m(0) = b_{1,1}$, for any initial position that lays between or on the g, e, the $x_{2,1}$ axis and the $x_{1,1}$ axis, which is the very light gray area. For an initial position with $q_m(0) = b_{1,1}$ in the very light gray area the fastest way to reach the light gray area is to perform a setup to $b_{2,1}$ and idle until the light gray area. In the light gray area it it known that idling at buffer $b_{2,1}$ until b^1 is the fastest to reach the ideal Savkin trajectory. Therefore, it can be concluded that for initial positions in the very light gray area the fastest way to reach the ideal Savkin trajectory is to perform a setup for buffer $b_{2,1}$ and idle at $b_{2,1}$ until b^1 has been reached.

Consider f as depicted in Figure B.9. Line f runs parallel with c, which was introduced in Figure B.4. Line f and interval c are $\theta_{1\mapsto 2}$ apart from one another. Line f is $\theta_{1\mapsto 2} + \theta_{2\mapsto 1}$ apart from a^1. Let $q_m(0) = b_{1,1}$, for any initial position that lays between or on the f, c and the $x_{2,1}$ axis, which is the very dark gray area. It can be seen from the phase diagram that at least a setup to $b_{2,1}$ and back to $b_{1,1}$ needs to be performed in order to reach the ideal Savkin trajectory for any initial position with $q_m(0) = b_{1,1}$ in the very dark gray area. The remaining distance after these setups can be covered with the following options:

1. idle at $b_{1,1}$ to a^1.
2. if positioned in the dark gray area, process work from $b_{1,1}$ until d, perform a setup to buffer $b_{2,1}$. If the resulting position falls outside the dark gray the ideal trajectory can not be reached with this option.
3. if positioned in the light gray area, setup for $b_{2,1}$ and idle at $b_{2,1}$ until b^1. If the resulting position falls outside the light gray the ideal trajectory can not be reached with this option.

It can be seen from the phase diagram that option 2 and 3 take more time than option 1. Therefore, it can be concluded that for any initial position with $q_m(0) = b_{1,1}$ in the very dark gray area the fastest way to the ideal Savkin trajectory is: perform a setup to buffer $b_{2,1}$, perform a setup to buffer $b_{1,1}$ and idle at $b_{1,1}$ until a^1.

The white triangle in the phase diagram of Figure B.9 is the only area for which no actions have been defined yet. Let $q_m(0) = b_{1,1}$. For any initial position in the white triangle the following two options exist

1. perform a setup to buffer $b_{2,1}$ and idle at $b_{2,1}$ until b^1.
2. perform a setup to buffer $b_{2,1}$, process work from $b_{2,1}$ until c, perform a setup to $b_{1,1}$.

Which option results in the fastest trajectory depends on the position in the white triangle. Consider the upper white triangle in the phase diagram of Figure B.10. This triangle depicts the area which can be reached after performing a setup to buffer $b_{2,1}$. Note that h, the upper line of the triangle, runs parallel to b^1 and runs $\theta_{2\mapsto 1}$ apart from b^1. Thus, idling from h to b^1 takes the same amount of time as performing a setup to $b_{1,1}$ from c. Consider position

1 on h, which can be reached from anywhere on the dashed line by idling. Instead of idling work could have been processed from buffer $b_{2,1}$. A vertical line from position 1 to c depicts the line which could have been reached if the time spend on idling was used for processing work. Consider a line parallel to b^1 and through the position where the vertical line and c cross. The position where this line crosses the dashed line indicates the position from which idling to h or processing work from $b_{2,1}$ to c takes the same amount of time. Thus, from this position it takes the same amount of time to reach the ideal Savkin trajectory. Consider a line through this position and the position where c and h cross, as depicted in Figure B.11. This line can be copied to the lower white triangle. Then, option 1 is the fastest way to the ideal Savkin trajectory for all the initial positions above this line and option 2 is the fastest way to the ideal Savkin trajectory for all the initial positions below this line.

The actions for an initial position on or below a with $q_m(0) = b_{1,1}$ that results in the fastest trajectory to the ideal Savkin trajectory are summarized in Figure B.12 and defined in Table B.1. In the same manner as for initial positions with $q_m(0) = b_{1,1}$ all actions for initial positions below or on b with $q_m(0) = b_{2,1}$ can be determined. The actions are summarized in Figure B.13 and defined in Table B.2.

Table B.1: Actions for any initial position with $q_m(0) = b_{1,1}$ below a, as depicted in Figure B.12, that result in the fastest trajectory to the ideal Savkin trajectory.

Dark gray	process from $b_{1,1}$ work until d, perform a setup to buffer $b_{2,1}$
Light gray	perform a setup to buffer $b_{2,1}$ and idle at $b_{2,1}$ to b^1
Very light gray	perform a setup to buffer $b_{2,1}$, perform a setup to buffer $b_{1,1}$ and idle at $b_{1,1}$ until a^1
White	perform a setup to buffer $b_{2,1}$, process work from $b_{2,1}$ until c, perform a setup to $b_{1,1}$

Table B.2: Actions for any initial position with $q_m(0) = b_{2,1}$ below b, as depicted in Figure B.13, that result in the fastest trajectory to the ideal Savkin trajectory.

Dark gray	process work from $b_{2,1}$ until c, perform a setup to buffer $b_{1,1}$
Light gray	perform a setup to buffer $b_{1,1}$ and idle at $b_{1,1}$ to a^1
Very light gray	perform a setup to buffer $b_{1,1}$, perform a setup to buffer $b_{2,1}$ and idle a $b_{2,1}$ until b^1
White	perform a setup to buffer $b_{1,1}$, process work from $b_{1,1}$ until d, perform a setup to $b_{2,1}$

B.4 Number of cycles

The proof in this section is for a system without variability. Therefore, the notation '^{c}' is left out for simplicity.
The figures of to this section are depicted in Appendix B.8.
Consider the system depicted in Figure B.1. Assume that the machine always follows the cycle: process for a from buffer $b_{1,1}$, setup for buffer $b_{2,1}$, process from $b_{2,1}$, setup for $b_{1,1}$. For this simple two buffer case let D_p^k denote the amount of time work is removed from buffer $b_{p,1}$ during cycle j. Assume that $q_m(0) = b_{1,1}$, thus the machine is ready to process from buffer $b_{1,1}$. In addition, it is assumed that a machine can idle by extending a setup. Furthermore, it is assumed that the time required to perform a setup when the machine switches from $b_{1,1}$ to $b_{2,1}$, denoted with $\theta_{1\mapsto 2}$, does not have to be equal to the setup time when the machine switches from $b_{2,1}$ to $b_{1,1}$, denoted with $\theta_{2\mapsto 1}$. Then, for this simple two buffer case the minimal scheduling period (3.19) becomes:

$$T_0 = \frac{\theta_{1\mapsto 2} + \theta_{2\mapsto 1}}{1 - \frac{\lambda_1}{\mu_{1,1}} \frac{\lambda_2}{\mu_{2,1}}}. \tag{B.13}$$

Let $\theta_{p,1}^{\mathrm{e}}(k)$ denote the time that the setup for buffer $b_{p,1}$ is extended in cycle k and let $\theta_{p,1}^{\mathrm{t}}(k)$ denote the total length of setup with extension in cycle k, thus:

$$\theta_{p,1}^{\mathrm{t}}(k) = \theta_{p\mapsto \mathbf{next_{SAV}}[b]} + \theta_{p,1}^{\mathrm{e}}(k) \tag{B.14}$$

Finally, let T^k denote the length that one cycles lasts, then:

$$T^k = \theta_{1,1}^{\mathrm{t}}(k) + \theta_{2,1}^{\mathrm{t}}(k) + D_1^k + D_2^k. \tag{B.15}$$

The amount of time that work is removed from buffer $b_{1,1}$ during cycle k can be computed with:

$$D_1^k = \frac{x_{1,1}(t_{1,1}^k) - x_{1,1}(T_{1,1}^k)}{\mu_{1,1} - \lambda_1}, \tag{B.16}$$

in which:

$$\begin{aligned} x_{1,1}(T_{1,1}^k) &= x_{1,1}(t_{1,1}^{k+1}) - \lambda_1 \left(t_{1,1}^{k+1} - T_{1,1}^k \right) \\ &= x_{1,1}(t_{1,1}^{k+1}) - \lambda_1 \left(\theta_{1,1}^{\mathrm{t}}(k) + \theta_{2,1}^{\mathrm{t}}(k) + D_2^k \right). \end{aligned} \tag{B.17}$$

The amount of time that work is removed from buffer $b_{2,1}$ during cycle k can be computed with:

$$D_2^k = \frac{x_{2,1}(t_{2,1}^k) - x_{2,1}(T_{2,1}^k)}{\mu_{2,1} - \lambda_2}, \tag{B.18}$$

in which:

$$\begin{aligned} x_{2,1}(t^k_{2,1}) &= x_{2,1}(t^k_{1,1}) + \lambda_2 \left(t^{k+1}_{1,1} - t^k_{2,1} \right) \\ &= x_{2,1}(t^k_{1,1}) + \lambda_2 \left(D^k_1 + \theta^{\mathrm{t}}_{2,1}(k) \right) \qquad \text{(B.19)} \\ x_{2,1}(T^k_{2,1}) &= x_{2,1}(t^{k+1}_{1,1}) - \lambda_2 \left(t^{k+1}_{1,1} - T^k_{2,1} \right) \\ &= x_{2,1}(t^{k+1}_{1,1}) - \lambda_2 \theta^{\mathrm{t}}_{1,1}(k). \qquad \text{(B.20)} \end{aligned}$$

Then, with (B.16) to (B.20) and after simplification (B.15) becomes:

$$\begin{aligned} T^k = {} & \theta^{\mathrm{t}}_{1,1}(k) + \theta^{\mathrm{t}}_{2,1}(k) + \\ & + \frac{x_{1,1}(t^k_{1,1})\mu_{2,1} + \lambda_1\mu_{2,1}(\theta^{\mathrm{t}}_{1,1}(k) + \theta^{\mathrm{t}}_{2,1}(k)) - x_{1,1}(t^{k+1}_{1,1})\mu_{2,1}}{\mu_{1,1}\mu_{2,1} - \lambda_1\mu_{2,1} - \lambda_2\mu_{1,1}} + \\ & + \frac{x_{2,1}(t^k_{1,1})\mu_{1,1} + \lambda_2\mu_{1,1}(\theta^{\mathrm{t}}_{1,1}(k) + \theta^{\mathrm{t}}_{2,1}(k)) - x_{2,1}(t^{k+1}_{1,1})\mu_{2,1}}{\mu_{1,1}\mu_{2,1} - \lambda_1\mu_{2,1} - \lambda_2\mu_{1,1}}. \end{aligned} \qquad \text{(B.21)}$$

With (B.14) this can be simplified into:

$$\begin{aligned} T^k = {} & \alpha + \xi \left(\theta^{\mathrm{e}}_{1,1}(k) + \theta^{\mathrm{e}}_{2,1}(k) \right) + \\ & + \beta_1 \left(x_{1,1}(t^k_{1,1}) - x_{1,1}(t^{k+1}_{1,1}) \right) + \beta_2 \left(x_{2,1}(t^k_{1,1}) - x_{2,1}(t^{k+1}_{1,1}) \right), \end{aligned} \qquad \text{(B.22)}$$

in which:

$$\begin{aligned} \alpha &= \theta_{1\mapsto 2} + \theta_{2\mapsto 1} + \frac{\theta_{1\mapsto 2} + \theta_{2\mapsto 1}(\lambda_1\mu_{2,1} + \lambda_2\mu_{1,1})}{\mu_{1,1}\mu_{2,1} - \lambda_1\mu_{2,1} - \lambda_2\mu_{1,1}} \\ &= \frac{\theta_{1\mapsto 2} + \theta_{2\mapsto 1}}{1 - \frac{\lambda_1}{\mu_{1,1}} - \frac{\lambda_2}{\mu_{2,1}}} = T_0 \qquad \textit{see (B.13)} \qquad \text{(B.23)} \\ \xi &= \frac{\lambda_1\mu_{2,1} + \lambda_2\mu_{1,1}}{\mu_{1,1}\mu_{2,1} - \lambda_1\mu_{2,1} - \lambda_2\mu_{1,1}} \qquad \text{(B.24)} \\ \beta_1 &= \frac{\mu_{2,1}}{\mu_{1,1}\mu_{2,1} - \lambda_1\mu_{2,1} - \lambda_2\mu_{1,1}} \qquad \text{(B.25)} \\ \beta_2 &= \frac{\mu_{1,1}}{\mu_{1,1}\mu_{2,1} - \lambda_1\mu_{2,1} - \lambda_2\mu_{1,1}}. \qquad \text{(B.26)} \end{aligned}$$

From (B.5) it follows that $\xi, \beta_p > 0$. Let $\hat{T}^k$ denote the time after k cycles, then:

$$\hat{T}^k = \hat{T}^{k-1} + T^k \qquad \forall k \geq 1, \qquad \text{(B.27)}$$

in which $\hat{T}^0 = 0$. This can be rewritten:

$$\begin{aligned}
\hat{T}^k &= \sum_{k=1}^{k} T^k \qquad \textit{use (B.21) and simplify} \\
&= kT_0 + \xi \sum_{k=1}^{k} \left(\theta^{\mathrm{e}}_{1,1}(k) + \theta^{\mathrm{e}}_{2,1}(k)\right) + \\
&+ \beta_1 \left(x_{1,1}(t^1_{1,1}) - x_{1,1}(t^{k+1}_{1,1})\right) + \beta_2 \left(x_{2,1}(t^1_{1,1}) - x_{2,1}(t^{k+1}_{1,1})\right) \\
&= kT_0 + \xi \sum_{k=1}^{k} \left(\theta^{\mathrm{e}}_{1,1}(k) + \theta^{\mathrm{e}}_{2,1}(k)\right) + \\
&+ \beta \left(x(t^1_{1,1}) - x(t^{k+1}_{1,1})\right).
\end{aligned} \tag{B.28}$$

Note, that $x(t^1_{1,1})$ denotes the position on the phase diagram at $t = 0$ and that $x(t^{k+1}_{1,1})$ denotes the position on the phase diagram at the end of cycle k.

Let $x(t^{K+1}_{1,1}) = X$ be a position on the phase diagram that can be reached in K cycles without extending the setups and assume that the same position X can also be reached in $K + 1$ cycles without extending the setups, then:

$$\begin{aligned}
\hat{T}^K &= KT_0 + \beta\left(x(0) - X\right) \\
\hat{T}^{K+1} &= (K+1)T_0 + \beta\left(x(0) - X\right) = \hat{T}^K + T_0.
\end{aligned}$$

It can be concluded that if a position on the phase diagram can be reached in K cycles without extending the setups then reaching that same position in $K + 1$ cycles will always take more time.

Let $x(t^{K+1}_{1,1}) = X$ be a position on the phase diagram that can be reached in K cycles without extending the setups and assume that the same position X can also be reached in K cycles with extending the setups. It can be seen from (B.28) that reaching the position X with extending the setups will last:

$$\xi \sum_{k=1}^{K} \left(\theta^{\mathrm{e}}_{1,1}(k) + \theta^{\mathrm{e}}_{2,1}(k)\right)$$

longer than reaching the position X without extending the setups.

B.5 Idling

Consider the system depicted in Figure B.1 for which a phase diagram is shown in Figure B.14. It is assumed that the time required to perform a setup when the machine switches from $b_{1,1}$ to $b_{2,1}$, denoted with $\theta_{1\mapsto 2}$, does not have to be equal to the setup time when the machine switches from $b_{2,1}$ to $b_{1,1}$, denoted with $\theta_{2\mapsto 1}$. In this section it shown for positions that lay above the gray area that idling is never an option. The question that needs to be answered is: At this specific moment in time does it have an advantage to idle the machine? An example is studied in order to answer this question. Assume that the machine is processing work from

buffer $b_{1,1}$ and reaches a position 1 on the phasediagram of Figure B.14. The question now is: will idling the machine at this specific moment result in a trajectory and state on the phase diagram faster than not idling the machine. Assume that the machine idles for some time and then continues processing work from $b_{1,1}$. This is depicted with the solid arrows in the phase diagram. The trajectory continues along arrow z. Now assume that the machine did not idle at position 1. To reach the z arrow the machine can idle at a later moment in time, as depicted with the dashed arrows. Both options, idling now or idling later, last the same amount of time and reach the same trajectory. It can be concluded that idling now does not have an advantage, idling later will result in the same trajectory and state. Therefore, idling is never an option for reaching the ideal Savkin trajectory from positions above the gray area, since the trajectory can not be reached within a limited amount of time.

B.6 $\mathbb{A}\boldsymbol{D} \leq \boldsymbol{s}$

The proof in this section is for a system without variability. Therefore, the notation '^{c}' is left out for simplicity.
In this section is shown how (6.37) and (6.38) can be rewritten in the following form:

$$\mathbb{A}D \leq s. \tag{B.29}$$

In which $\mathbb{A}$ is a $[kP \times kP]$ matrix, the vectors D, s have a length of kP and $\leq$ is understood component wise. Note that k denotes the number of cycles of the improved Savkin policy. First, (6.37) and (6.38) are repeated:

(6.37) :

$$D_p^k \leq \frac{x_{p,1}(0) + \lambda_p^c t_{p,1}^k - \mu_{p,1}^c \sum_{v=1}^{k-1} D_p^v}{\mu_{p,1}^c - \lambda_p^c}. \tag{B.30}$$

Note, that $t_{p,1}^k$ denotes the time at the start of the production run D_p^k and equals:

(6.38) :

$$t_{p,1}^k = ((k-1)P + p - 1)\,\theta_1 + \sum_{v=1}^{k-1}\sum_{w=1}^{P} D_w^v + \sum_{w=1}^{p-1} D_w^k. \tag{B.31}$$

Using (B.31) in (B.30) results in:

$$\begin{aligned} D_p^k \leq & \frac{x_{p,1}(0) + \lambda_p\left((k-1)\,P + p - 1\right)\theta_1}{\mu_{p,1} - \lambda_p} + \\ & + \frac{\lambda_p\left(\sum_{v=1}^{k-1}\sum_{w=1}^{P} D_w^v + \sum_{w=1}^{p-1} D_w^k\right) - \mu_{p,1}\sum_{v=1}^{k-1} D_p^v}{\mu_{p,1} - \lambda_p}. \end{aligned} \tag{B.32}$$

This can be rewritten:

$$D_p^k \leq \frac{x_{p,1}(0) + \lambda_p \left((k-1)\, P + p - 1\right) \theta_1}{\mu_{p,1} - \lambda_p} + \\ + \alpha_p \left(\sum_{v=1}^{k-1} \left(\sum_{w=1}^{p-1} D_w^v + \sum_{w=p+1}^{P} D_w^v \right) + \sum_{w=1}^{p-1} D_w^k \right) - \sum_{v=1}^{k-1} D_p^v, \tag{B.33}$$

in which:

$$\alpha_p = \frac{\lambda_p}{\mu_{p,1} - \lambda_p}.$$

It can be seen from (B.33) that D_p^k is linear depended on all the previous production runs. Let $\mathbb{A}_1$,$\mathbb{A}_2$ and $\emptyset$ be $[P \times P]$ matrices defined as:

$$\mathbb{A}_1 = \begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ -\alpha_2 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -\alpha_{P-1} & \dots & -\alpha_{P-1} & 1 & 0 \\ -\alpha_P & \dots & -\alpha_P & -\alpha_P & 1 \end{bmatrix}, \tag{B.34}$$

$$\mathbb{A}_2 = \begin{bmatrix} 1 & -\alpha_1 & \dots & \dots & -\alpha_1 \\ -\alpha_2 & 1 & -\alpha_2 & \dots & -\alpha_2 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -\alpha_{P-1} & \dots & -\alpha_{P-1} & 1 & -\alpha_{P-1} \\ -\alpha_P & \dots & -\alpha_P & -\alpha_P & 1 \end{bmatrix}, \tag{B.35}$$

and

$$\emptyset = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}. \tag{B.36}$$

Furthermore, let D^k, s^k be vectors with a length of P and defined as:

$$s^k = \begin{bmatrix} \frac{x_{1,1}(0) + \lambda_1((k-1)P+1-1)\theta_1}{\mu_{1,1} - \lambda_1} \\ \frac{x_{2,1}(0) + \lambda_2((k-1)P+2-1)\theta_1}{\mu_{2,1} - \lambda_2} \\ \vdots \\ \frac{x_{P,1}(0) + \lambda_1((k-1)P+P-1)\theta_1}{\mu_{P,1} - \lambda_P} \end{bmatrix} \tag{B.37}$$

and

$$D^k = \begin{bmatrix} D_1^k \\ D_2^k \\ \vdots \\ D_P^k \end{bmatrix}. \tag{B.38}$$

It can be seen from (B.33) that the following holds:

$$\mathbb{A}_2 \sum_{v=1}^{k-1} D^v + \mathbb{A}_1 D^k \leq s^k. \tag{B.39}$$

This can be rewritten in the form of (B.29) in which:

$$\mathbb{A} = \left.\begin{bmatrix} \mathbb{A}_1 & \emptyset & \dots & \dots & \emptyset \\ \mathbb{A}_2 & \mathbb{A}_1 & \emptyset & \dots & \emptyset \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbb{A}_2 & \dots & \mathbb{A}_2 & \mathbb{A}_1 & \emptyset \\ \mathbb{A}_2 & \dots & \mathbb{A}_2 & \mathbb{A}_2 & \mathbb{A}_1 \end{bmatrix}\right\} [k \times k], \tag{B.40}$$

$$D = \begin{bmatrix} D^1 \\ D^2 \\ \vdots \\ D^{k-1} \\ D^k \end{bmatrix} \tag{B.41}$$

and

$$s = \begin{bmatrix} s^1 \\ s^2 \\ \vdots \\ s^{k-1} \\ s^k \end{bmatrix}. \tag{B.42}$$

B.7 Determining the maximum value of ϵ

The proof in this section is for a system without variability. Therefore, the notation '^{c}' is left out for simplicity.
In this section it is investigated what size ϵ should be for the optimization problem as described in Section 6.2. Let $\mathfrak{R}$ be the area in which all positions lay that have a distance ϵ or less from $x^{\text{id}}(t^{\text{id}}_{1,1})$. Then ϵ should be so small such that it is guaranteed that if a position in $\mathfrak{R}$ can be reached in k cycles that a different position in that same area $\mathfrak{R}$ can not be reached faster in $k+1$ cycles.

Note, it is shown in Section 6.2 that extending setups is not necessary for the optimization problem. Furthermore, the length of the setup is equal for switching from $b_{1,1}$ to $b_{2,1}$ and back. Let x^{ϵ_1} be a position that can be reached in k cycles and falls within $\mathfrak{R}$ and x^{ϵ_2} be a position that can be reached in $k+1$ cycles and also falls within $\mathfrak{R}$. Furthermore, let $T^k_{\epsilon_1}$

and $T_{\epsilon_2}^{k+1}$ respectively denote the time in which position x^{ϵ_1} and x^{ϵ_2} are reached. Then, the following should hold according to (B.28):

$$\begin{aligned} \hat{T}_{\epsilon_1}^{k} &< \hat{T}_{\epsilon_2}^{k+1} \\ kT_0 + \beta\left(x(0) - x^{\epsilon_1}\right) &< (k+1)T_0 + \beta\left(x(0) - x^{\epsilon_2}\right) \\ \beta\left(x(0) - x^{\epsilon_1}\right) &< T_0 + \beta\left(x(0) - x^{\epsilon_2}\right) \\ \beta\left(x^{\epsilon_2} - x^{\epsilon_1}\right) &< T_0 \\ \beta_1\eta_1 + \beta_2\eta_2 &< T_0, \end{aligned} \tag{B.43}$$

in which:

$$\eta_i = x_{i,1}^{\epsilon_2} - x_{i,1}^{\epsilon_1}. \tag{B.44}$$

The maximal distance between two positions both within $\mathfrak{R}$ is equal to: 2ϵ and then the following holds:

$$\eta_1 = 2\epsilon\kappa \qquad 0 \leq \kappa \leq 1. \tag{B.45}$$

The distance between two positions is defined as:

$$\begin{aligned} \sqrt{\eta_1^2 + \eta_2^2} &= 2\epsilon \\ \eta_1^2 + \eta_2^2 &= 4\epsilon^2 \\ 4\epsilon^2\kappa^2 + \eta_2^2 &= 2\epsilon \end{aligned} \tag{B.46}$$

Solving η_2 from (B.46) results in:

$$\eta_2 = \sqrt{4 - 4\kappa^2}\epsilon. \tag{B.47}$$

Together with (B.43) this results in:

$$f(\kappa) < T_0, \tag{B.48}$$

in which:

$$f(\kappa) = 2\beta_1\epsilon\kappa + 2\beta_2\epsilon\sqrt{1 - \kappa^2}. \tag{B.49}$$

The maximum value of $f(\kappa)$ can be found with its derivative.

$$f'(\kappa) = 2\beta_1\epsilon - \frac{2\beta_2\epsilon\kappa}{\sqrt{1 - \kappa^2}} = 0. \tag{B.50}$$

Solving (B.50) for κ results in:

$$\kappa^* = \frac{\beta_1}{\sqrt{\beta_1^2 + \beta_2^2}}. \tag{B.51}$$

Using (B.51) in (B.48) and simplifying results in:

$$2\epsilon\sqrt{\beta_1^2+\beta_2^2} < T_0. \tag{B.52}$$

Solving ϵ from (B.52):

$$\begin{aligned} \epsilon &< \frac{T_0}{2\sqrt{\beta_1^2+\beta_2^2}} \qquad \textit{use (3.19), (B.25) and (B.25)} \\ &< \frac{\mu_{1,1}\mu_{2,1}\theta_1}{\sqrt{\mu_{1,1}^2+\mu_{2,1}^2}}. \end{aligned} \tag{B.53}$$

B.8 Figures

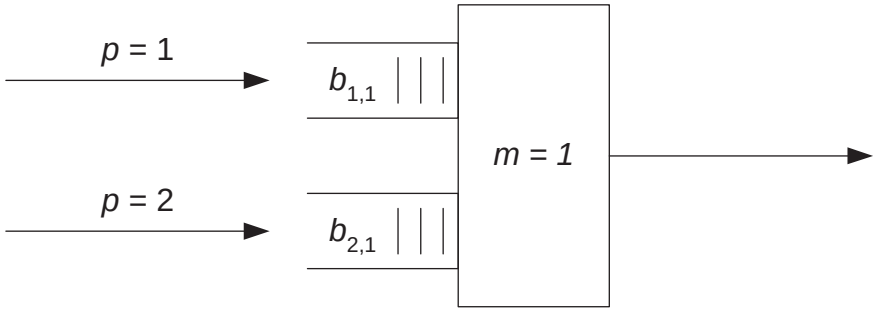

Figure B.1: A flexible manufacturing system.

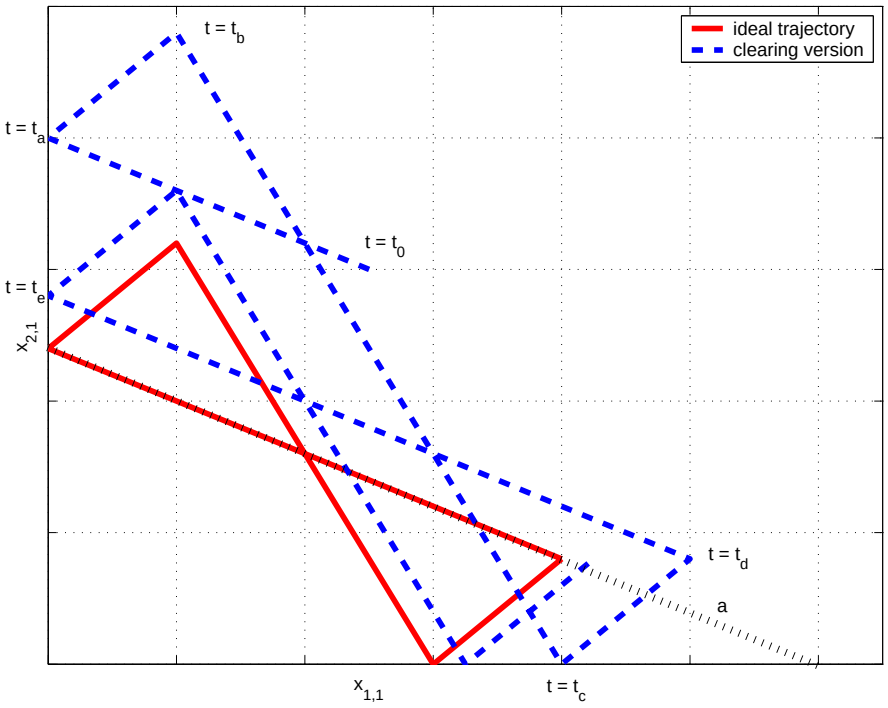

Figure B.2: A possible phase diagram of the buffer levels for the system depicted in Figure B.1 when the cyclic clearing policy is applied.

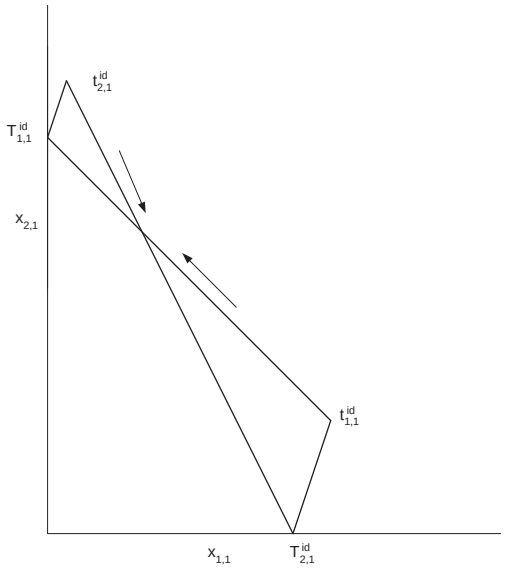

Figure B.3: A possible phase diagram of the buffer levels for the system depicted in Figure B.1.

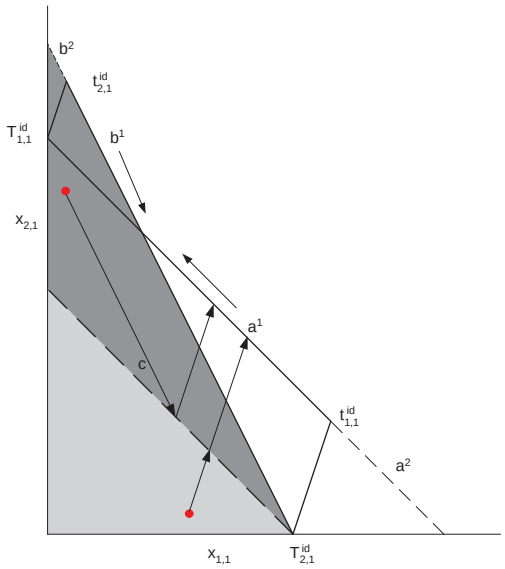

Figure B.4: A possible phase diagram of the buffer levels for the system depicted in Figure B.1.

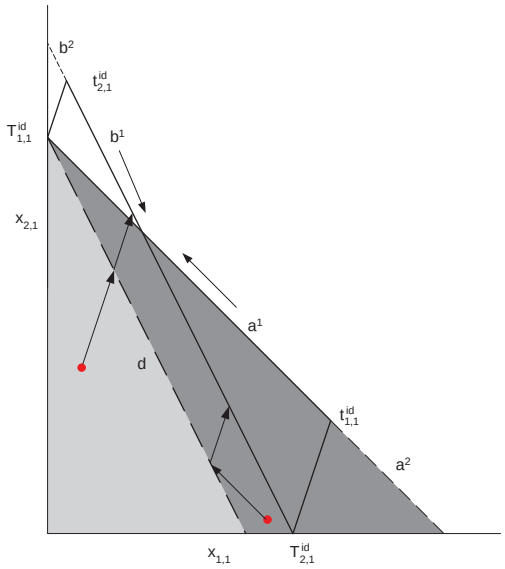

Figure B.5: A possible phase diagram of the buffer levels for the system depicted in Figure B.1.

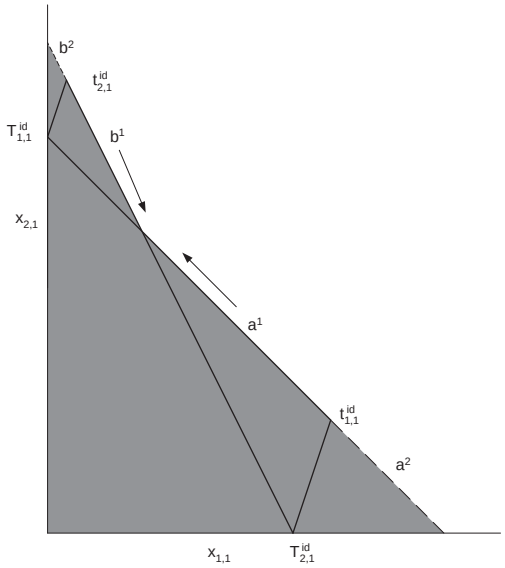

Figure B.6: All initial positions from which it is possible to reach the ideal Savkin trajectory for the system depicted in Figure B.1 if the machine has the proper machine state.

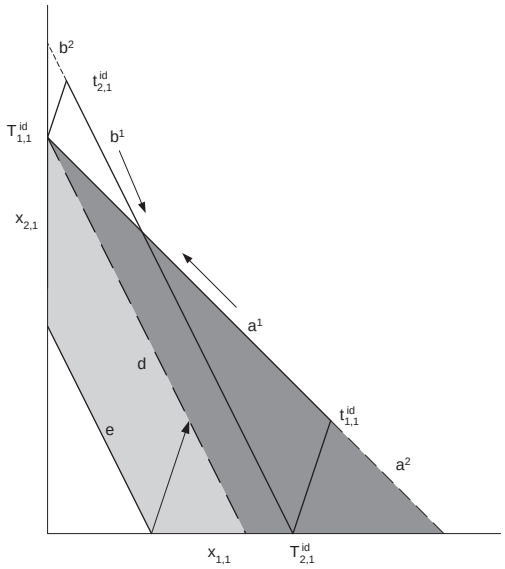

Figure B.7: A possible phase diagram of the system depicted in Figure B.1.

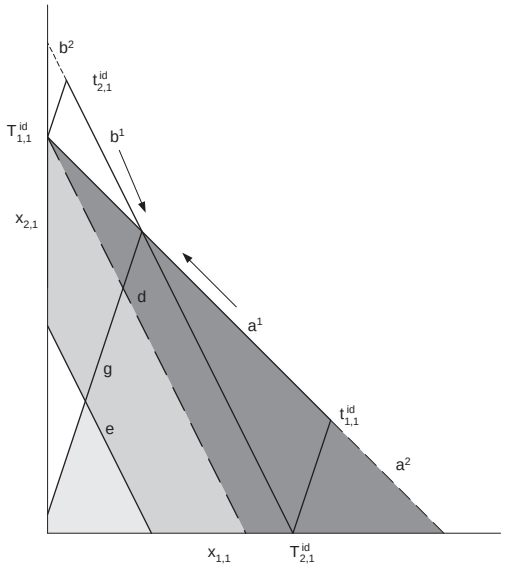

Figure B.8: A possible phase diagram of the system depicted in Figure B.1.

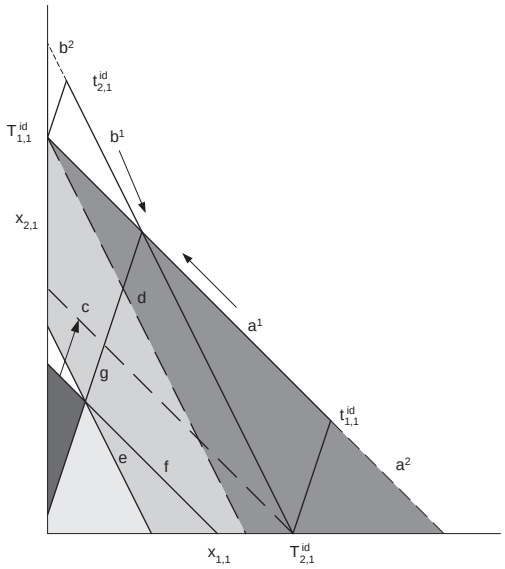

Figure B.9: A possible phase diagram of the system depicted in Figure B.1.

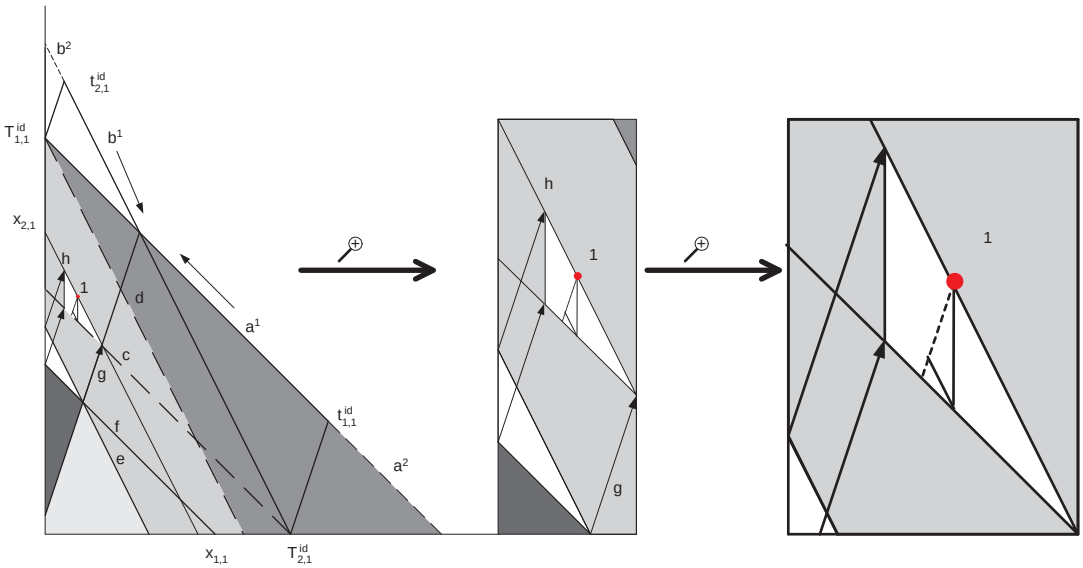

Figure B.10: A possible phase diagram of the system depicted in Figure B.1.

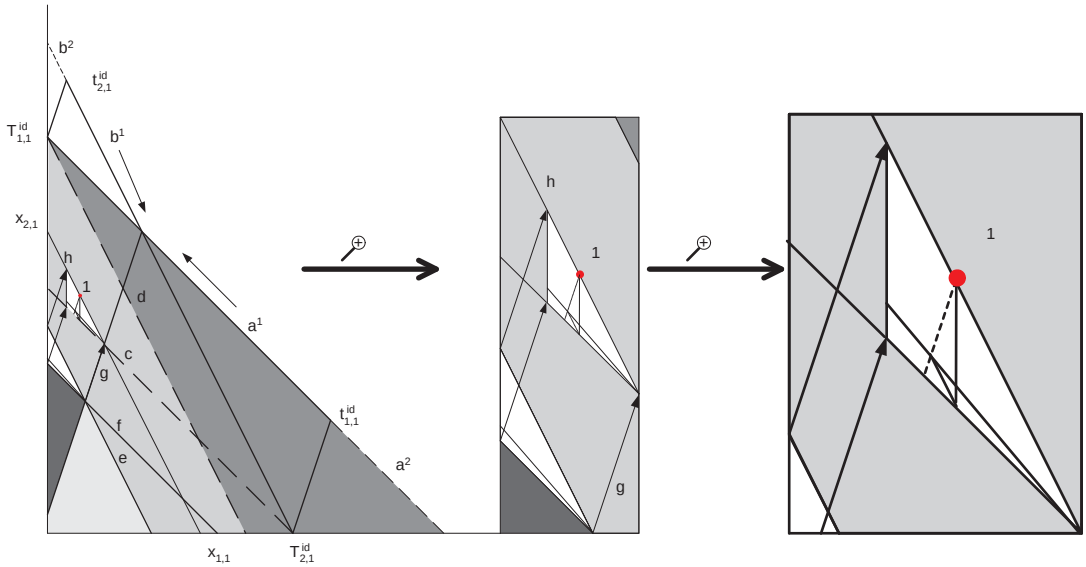

Figure B.11: A possible phase diagram of the system depicted in Figure B.1.

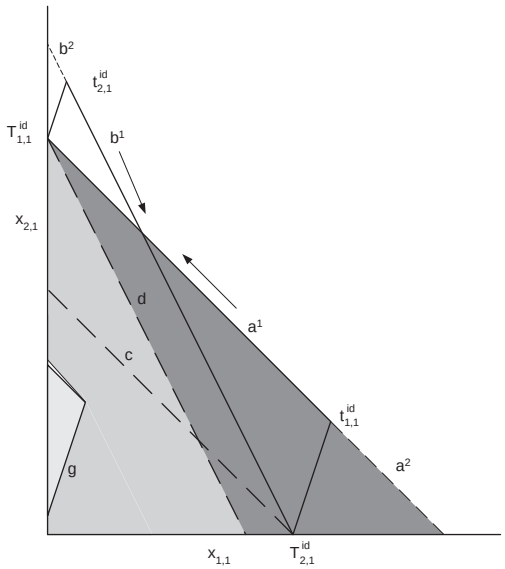

Figure B.12: A possible phase diagram of the system depicted in Figure B.1.

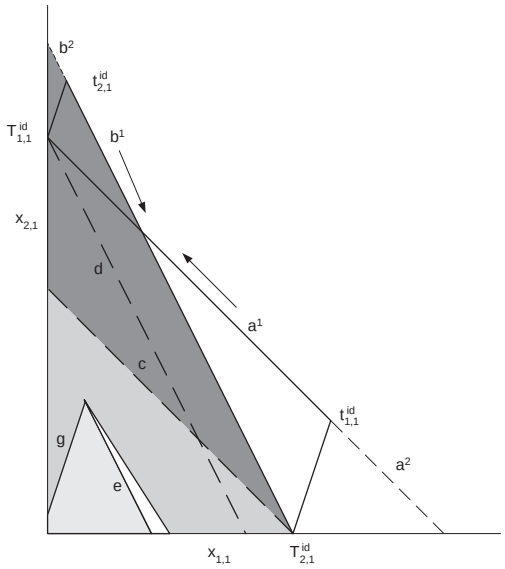

Figure B.13: A possible phase diagram of the system depicted in Figure B.1.

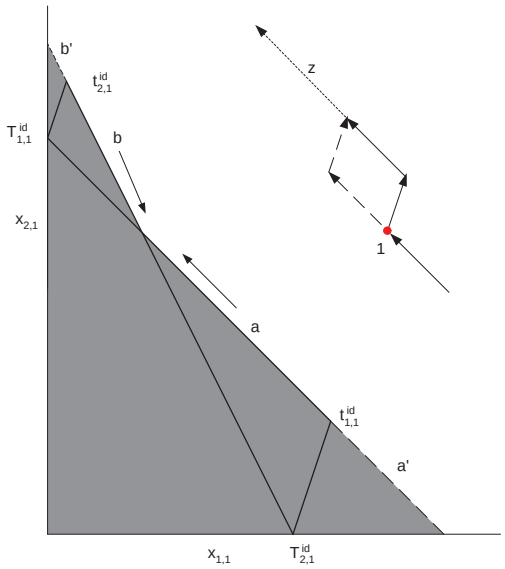

Figure B.14: Does idling introduces an advantage?

Appendix C

Simulation

C.1 Introduction

Different results are obtained when a simulation of a stochastic model is reproduced under the same conditions. Therefore, the results of just one simulation can not be used to draw conclusions on. The differences in the results are caused by the variability present. Furthermore, the results of one simulation run are not independent, which means that formulas from the classical statistics can not be used directly, since the classical statistics is based on independent variables. Two random variables are said to be *independent* if the value of X has no influence on the value of Y and vice versa. In a simulation the variables are not independent, for example if X is flow time of a part and Y is the flow time of the next part in a buffered production line. If the first part incurs a large delay and thus a large flow time the probability that the next part has a large flow time increases. In other words the value of X influences the value of Y. To solve the two problems mentioned above the following technique is used.

Let $Y_1, Y_2, \ldots, Y_m$ be random variables resulting from a simulation with a run length of m observations and let $y_{11}, y_{12}, \ldots, y_{1m}$ be the numerical values of these variables obtained from a simulation. If the simulation is reproduced independent from the previous one, but under the same conditions, other numerical values $y_{21}, y_{22}, \ldots, y_m$ are obtained for the variables $Y_1, Y_2, \ldots, Y_m$. Now suppose the simulation is reproduced independently n times resulting in the following observations:

$$\begin{array}{ccccc} y_{11}, & \cdots, & y_{1i}, & \cdots, & y_{1m} \\ \vdots & & \vdots & & \vdots \\ y_{j1}, & \cdots, & y_{ji}, & \cdots, & y_{jm} \\ \vdots & & \vdots & & \vdots \\ y_{n1}, & \cdots, & y_{ni}, & \cdots, & y_{nm} \end{array} .$$

The observations in one row are not independent from each other, but the observations $y_{1i}, y_{2i}, \ldots, y_{ni}$ in one column are and are independent observations for the variable Y_i. This property is called *independency across runs* and because of this property classical statistics can be applied.

C.2 Statistical analysis of steady-state systems

Let $Y_1, Y_2, \ldots$ be the output of a simulation and let $F_i(y|I)$ be the cumulative distribution function of Y_i, where y is a real number and I the initial conditions used in the simulation. Suppose that the simulation does not have a terminating event and if $F_i(y|I) \to F(y)$ when i goes to infinity then $F(y)$ is called the *steady-state distribution* of the output proces $Y_1, Y_2, \ldots$. Let Y denote this random steady-state variable with a distribution $F(y)$ and let $\nu = E(Y)$ be the mean of this distribution.

C.2.1 The transient phase

Suppose a simulation produces the output $Y_1, Y_2, \ldots, Y_m$. The sample mean $\overline{Y}(m) = \frac{\sum_{i=1}^{m} Y_i}{m}$ is a *biased* estimator of $\nu = E(Y)$ for any finite m. The sample mean is a biased estimator because at the beginning the observations, where the steady-state not yet has been reached, "corrupt" the sample mean. This problem is called the *problem of the initial transient* and a technique often used to solve this problem is called *initial data deletion*. This technique deletes the first l observations and uses the remaining observations to estimate ν, thus:

$$\overline{Y}(m,l) = \frac{\sum\limits_{i=1+l}^{m} Y_m}{m-l}. \tag{C.1}$$

The technique used in this report for determining l is as follows. A number of n simulations are made of a relative large length m, where Y_{ji} is an observation as defined earlier. Let $\overline{Y}_i = \frac{\sum_{j=1}^{n} y_j}{n}$ for $i = 1, 2, \ldots, m$, which are plotted against i. From the resulting figure choose l' as the value beyond which $\overline{Y}_i$ appears to have converged. The value l' is multiplied by 1.5 to ensure that the value l is not chosen to low, then $l = 1.5 \cdot l'$. Perform more simulations or make use of moving averages if the resulting plot oscillates too much to make a good choose for l'. If l' is picked correctly then the sample mean $\overline{Y}(m,l)$ will be approximately an unbiased estimator of ν, thus $E(\overline{Y}(m,l)) \approx \nu$.

C.2.2 Replication/Deletion approach for means

There are several ways to estimate the steady-state mean ν of a process with output $Y_1, Y_2, \ldots$. The technique used in this report is the Replication/Deletion approach for means. The mean reason this technique is used is because it is the easiest technique to understand and implement.

Let X_j be the sample mean of the steady-state observations in one simulation run:

$$X_j = \frac{\sum\limits_{i=l+1}^{m} Y_{ji}}{m-l} \quad \forall j = 1, 2, \ldots, n. \tag{C.2}$$

The X_j's are independent random variables with $E(X_j) \approx \nu$, because of the independency across runs and the deletion of the transient phase. If the number of independent runs n is large, the sampling distribution of the sample mean $\overline{X}(n)$ will be approximately normal with a mean μ and variance σ^2. This theorem is called the *Central Limit Theorem*.

Definition C.1 (Central Limit Theorem [Mon99])**.** If $X_1, X_2, \ldots, X_n$ is a random sample of size n taken from a population (either finite or infinite) with mean μ and finite variance σ^2, and if $\overline{X}(n)$ is the sample mean, then the limiting form of the distribution of

$$Z = \frac{\overline{X}(n) - \mu}{\sigma/\sqrt{n}} \tag{C.3}$$

as $n \to \infty$, is the standard normal distribution.

In many cases of practical interest, if $n \geq 30$, the normal approximation will be satisfactory regardless of the shape of the population. However since the variance σ^2 of the approximated normal distribution is unknown, equation (C.3) can not be used. When σ^2 is unknown, a logical procedure is to replace σ with the sample variance $S(n)$.

Definition C.2. [Mon99] Let $X_1, X_2, \ldots, X_n$ be a random sample from a normal distribution with unknown mean μ and unknown variance σ^2. The quantity

$$T = \frac{\overline{X}(n) - \mu}{S(n)/\sqrt{n}} \tag{C.4}$$

has a t (student-t) distribution with $n - 1$ degrees of freedom.

Definition C.2 applies for populations with a normal distribution. The distribution of X_j is unknown, but the distribution of the sample mean $\overline{X}(n)$ is approximately normal if n is large. Therefore, the sample mean $\overline{X}(n)$ is an approximately unbiased point estimator for μ and an approximate $100(1-\alpha)$ confidence interval for μ is given by [Mon99]:

$$\overline{X}(n) - t_{\alpha/2,n-1}\frac{S(n)}{\sqrt{n}} \leq \mu \leq \overline{X}(n) + t_{\alpha/2,n-1}\frac{S(n)}{\sqrt{n}}. \tag{C.5}$$

C.2.3 Number of replications

As can been seen from (C.5) the size of the confidence interval depends on the number of replications n. At least 30 replications are needed to approximate the sampling distribution of the sampling mean as normal. Only 30 replications can result in a large confidence interval, thus in a large error of $\overline{X}(n)$. To decrease the error, the number of replications needs to be increased. In this section a procedure to the determine the number of replications required to reach a certain maximum error is introduced.

The relative error of a sample is the ratio between the error $|\overline{X}(n) - \mu|$ and the mean and is defined by [Law00]:

$$\gamma = \frac{|\overline{X}(n) - \mu|}{\mu}. \tag{C.6}$$

However, the mean μ is unknown, thus the relative error can not be computed directly. Suppose n replications are performed until the following is true:

$$\frac{\overline{X}(n) + t_{\alpha/2,n-1}\frac{S(n)}{\sqrt{n}}}{|\overline{X}(n)|} \leq \gamma. \tag{C.7}$$

Then [Law00]:

$$1-\alpha \approx P(\frac{|\overline{X}(n)-\mu|}{|\overline{X}(n)|} \leq \frac{\text{half-length}}{|\overline{X}(n)|}), \tag{C.8}$$

in which *half-length* equals the half size of the confidence interval: $\overline{X}(n) + t_{\alpha/2,n-1}\frac{S(n)}{\sqrt{n}}$. It can be shown that:

$$1-\alpha \approx P(\frac{|\overline{X}(n)-\mu|}{|\overline{X}(n)|} \leq \frac{\text{half-length}}{|\overline{X}(n)|}) \leq P(\frac{|\overline{X}(n)-\mu|}{|\mu|} \leq \frac{\gamma}{(1-\gamma)}). \tag{C.9}$$

Thus, if the relative error is estimated by (C.7), then $\overline{X}(n)$ has a relative error of at most $\gamma/(1-\gamma)$ with a probability of approximately $1-\alpha$. In other words, if 100 independent 90 percent confidence intervals are constructed, it is expected that $\overline{X}(n)$ has a relative error of at most $\gamma/(1-\gamma)$ in about 90 of the 100 cases. In about 10 of the 100 cases the relative error is greater than $\gamma/(1-\gamma)$. The relative error is larger than γ, because the sample mean μ is estimated by $\overline{X}(n)$. A sequential procedure for obtaining an estimate of μ with a predefined relative error that only takes as many replications as needed is as follows:

1. Make n_0 replications (in this case $n_0 = 30$).
2. Compute $\overline{X}(n)$ and the half-length of the confidence interval where n is equal to n_0 and $100(1-\alpha)$ is the confidence level.
3. If $\frac{\text{half-length}}{|\overline{X}(n)|} \leq \gamma$ stop and use $\overline{X}(n)$ as point estimate for μ. If $\frac{\text{half-length}}{|\overline{X}(n)|} > \gamma$ then make a new simulation, replace n with $n+1$ and start again at step 2.

Thus at least n_0 replications are performed to approximate the sample mean with a normal distribution. If the computed approximated relative error is larger than desired then additional replications are made until the desired relative precision is met.

C.2.4 Simulation run length

The technique used to estimate l only provides an approximation of the length of the transient phase. It is possible that a number of observations beyond l are still corrupted by the transient phase. However if the simulation run length m is large enough these observations will have little influence on X_j or $\overline{X}(n)$. The initial choice of the run length m for the simulation is this report is $m = 5 \cdot l$. The run length m is increased if from the figure, from which a choice for l is made, can be observed that the initial choice for m is too short. The run length is not based on a certain criterion, however there is a trade off between the run length and the number of replications. The number of replications needed to reach a certain relative error decreases when the simulation length increases.

Appendix D

Simulation files

In this appendix the files are presented used for the simulation experiments of chapters 4 and 5. The simulations have been performed according to the replication / deletion approach, which has been explained in Appendix C.

D.1 Determining the transient phase

The following Matlab files have been used to determine the transient phase. The χ files can be found in D.3.

transientphase1.m

```
% transientphase1.m is used to determine the transient phase. The file is made to
% run in a unix environment. This file simulates the chi-file 10 times until
% 30000 parts have exit the system. It saves the relevant data to the workspace.
% This data can be used in transientphase2.m. The chi file that this file
% runs continue produces a lot of output, which is written in to files. To
% ensure fast simulation do not run this file on a file-server, but
% directly on a computer.
clear all
clc

in1=input('Which python file? ','s');       % The python file with the data
in2=input('Which chi file? ','s');          % The chi file
in=['python2 ' in1 ' ' in2 ' 30000 0 0']    % Defines the Unix command

for i=1:10
    unix (in);                              % Executes 'in' in Unix
    d=load('E.txt');                        % Loads the file containing the flow
                                            % times of each part
    if i==1;
        E=d;                                % Stores the flow times in E
    else
        E=[E d(:,2)];                       % Stores the flow times in E
    end
end

A=[];                                       % Matrix A contains on (i,j) the av.
for i=1:length(E)                           % flow time of of the parts 1 until i for
    A(i,2:11)=mean(E(1:i,2:11));            % the jth simulation. The last column
    A(i,12)=mean(A(i,2:11));                % contains the av. of the av. flow time.
```

```
end
clear d i
save workspace
exit
```

transientphase2.m

```
% transientphase2.m is used to determine the transient phase.
% It needs data from transientphase1.m.
close all
clear all
clc

load workspace                          % Load data from transientphase1.m

ME=[E(:,1) mean(E(:,2:end),2)];         % ME contains the av. flow time of part i.


    % From figure 1 and 2 the transientphase can be determined.
figure (1)
stairs(ME(:,1),ME(:,2),'b')
grid
xlabel('Part number')
ylabel('Average flow time')
title(['The av. flow time of part i for ' in1 ' with ' in2], 'Interpreter','none')

    % Moving average, for more inforation see [Law00].
w=40;                                   % Window
Y=[];
for i=1:w
    s=-(i-1);
    se=i-1;
    a=sum((ME((i+s):(i+se),2)),1);
    Y=[Y (a/(2*i-1))];
end
for i=(w+1):(length(ME)-w)
    s=-w;
    se=w;
    a=sum((ME((i+s):(i+se),2)),1);
    Y=[Y (a/(2*w+1))];
end

figure (2)
stairs(Y)
grid
xlabel('Part number')
ylabel('Moving average flow time')
title(['The moving av. flow time of part i for ' in1 ' with ' in2], 'Interpreter','none')

    % From the figures above the transientphase can be determined.
    % From Figure 3 it can be determined if a simulation length of
    % m=5*l is long enough.

figure (3)
plot(E(:,1),A(:,2:11))
hold on
plot(E(:,1),A(:,12),'k','LineWidth',3)
grid
xlabel('Part number')
ylabel('Total Average flow time of part 1 to part i')
title(['Total Average flow time of part 1 to part i for ' in1 ' with ' in2], 'Interpreter','none')
legend('Simulation 1','Simulation 2','Simulation 3','Simulation 4','Simulation 5',...
    'Simulation 6','Simulation 6','Simulation 8','Simulation 9','Simulation 10',...
    'Average of all sim.',4)
```

D.2 Python

runner.py

```
# runner.py facilitates the startup of simulations.
import sys,os,string,math

pfile = raw_input("Which python file? ")
chifile = raw_input("Which chi file? ")
runl = raw_input("Length of simulation? ")
ini = raw_input("Transient phase?")
do = raw_input("Just one run(0) or multiple (1)?")
filename = raw_input("Which file name for possible output?")

    # The unix command for running the simulations is formed:
para='nohup python2 '+pfile+' '+chifile+' '+runl+' '+ini+' '+do+' >'+filename+' &'
print para

os.system("%s"  %(para))            # Inputs the command para
os.system('top -i')                 # Shows running processes on the server.
```

Problem_x.py

```
# Problem_x.py facilitates the input of data in the chi file.
# Note that Chi can not cope with a value of 1/2, this should
# be 0.5. Python transforms 1.0/2.0 automatically in a real number.

import sys,os,string,math

    # For deterministic simulations, the input in chi can only
    # exists of naturals. A value of 0.333 causes problems for
    # chi. Therefore, all the values have to be naturals.
    # This is done by multiplying them with a multiplication factor
    # Note that this factor is also present in the output.
m = 1

    # For stochastic simulation the following data is needed for
    # the triangulardistribution of the arrival and process rate:
mh = 94.0/87.0        # The right corner of the triangle
ml = 80.0/87.0        # The left corner of the triangle

    # When simulating the Savkin policy, the scheduling period
    # needs to be defined:
T = 1000.0/53.0



    # The time required to perform a setup is defined:
setup0= 1             # The minimal setup time for workstation 0
setup1= 1             # The minimal setup time for workstation 1
setupi= x             # The minimal setup time for workstation i
                      # etc,
    # The values are transformed into a string:
setup= '<| '+str(setup0*m)+' '+str(setup1*m)+' '+str(setupi*m)+' |>'

    # The following data is for the part types.
# ##############################################################################################
#                                  Part type 0
# ##############################################################################################

ta=25.0               # The inter arrival time
    # The interarrival time is transformed into a string
    # and max. and min. values of the triangle are computed
```

```
    # and transformed.
inter0= '<| '+str(ta*m*ml)+' '+str(ta*m)+' '+str(ta*m*mh)+' |>'

# ---------Production step 1
W= 0                  # Workstation at which the production step takes place
v= 1                  # This is the v th visit to this workstation
tr= 0.0               # The transportation delay to the workstation for this production step
bt= 1                 # The batchsize of transportation for this production step
pr= 1.0               # The process time for this production step
    # All the data is transformed into a string and the min
    # and max. value of the process time for the triangle
    # are computed.
step1= '< '+str(W)+' '+str(v)+' '+str(tr*m)+' '+str(bt)+' <| '+str(pr*m*ml)+' '+str(pr*m)+
' '+str(pr*m*mh)+' |> >'

# ---------Production step i
W= 5                  # Workstation at which the production step takes place
v= 1                  # This is the v th visit to this workstation
tr= 0.0               # The transportation delay to the workstation for this production step
bt= 1                 # The batchsize of transportation for this production step
pr= 1.0               # The process time for this production step
    # All the data is transformed into a string and the min
    # and max. value of the process time for the triangle
    # are computed.
stepi= '< '+str(W)+' '+str(v)+' '+str(tr*m)+' '+str(bt)+' <| '+str(pr*m*ml)+' '+str(pr*m)+
' '+str(pr*m*mh)+' |> >'

    # Etc., until all production steps of part type 0 have been put in.

    # All the steps are combined
# ------------Total part type 0----------------
part0= '[ '+step1+' '+stepi+' ]'

# ##############################################################################################
#                                 Part type 1
# ##############################################################################################
    # All data of part type 1


# ##############################################################################################
#                                 Part type i
# ##############################################################################################
    # All data of part type i

# ##############################################################################################
#                                 Total part types
# ##############################################################################################
    # The following string contains all interarrival times and there min. and max. values
interarrival= '<| '+inter0+' '+inter1+' '+interi+' |>'
    # The following string contains all other data of the part types
totparts= '<| '+part0+' '+part1+' '+parti+' |>'

# ##############################################################################################
    # The data below does not have to be changed

    # The following string contains all the information about the system.
TOT= '< '+str(T*m)+' '+interarrival+' '+totparts+' '+setup+' >'

    # The following defines the unix command.
para1=(sys.argv[1])+" '%s' "%(TOT)
para='./'+para1+' '+(sys.argv[2])+' '+(sys.argv[3])+' '+(sys.argv[4])
print para

    # If the fourth arg. is 0, then just run the chi file once
    # if the fourth arg. is 1, then use RUN.py to run the chi file
    # multiple times.
```

```
if sys.argv[4] == '0':
    os.system("%s"  %(para))
elif sys.argv[4] == '1':
    os.system("python2 RUN.py %s" %(para))
print 'stopped \n'
```

RUN.py

```
# RUN.py  facilitates the execution of multiple simulations
# and simulating until the relative error of the
# estimated average is 0.05/(1-0.05) with a probability
# of approximately 1-0.05=0.95%
import sys,os,string,math

    #  Values of the student-t distribution for n>= 30, 0.05/2=0.025
tvalue=[2.042, 2.040, 2.037, 2.035, 2.032, 2.030, 2.028, 2.026, 2.024, 2.023,
        2.021, 2.020, 2.018, 2.017, 2.015, 2.014, 2.013, 2.012, 2.011, 2.010,
        2.009, 2.008, 2.007, 2.006, 2.005, 2.004, 2.003, 2.002, 2.002, 2.001,
        2.000, 2.000, 1.999, 1.998, 1.998, 1.997, 1.997, 1.996, 1.995, 1.995,
        1.994, 1.994, 1.993, 1.993, 1.993, 1.992, 1.992, 1.991, 1.991, 1.990,
        1.990, 1.990, 1.989, 1.989, 1.989, 1.988, 1.988, 1.988, 1.987, 1.987,
        1.987, 1.986, 1.986, 1.986, 1.986, 1.985, 1.985, 1.985, 1.984, 1.984]

    # First, the definition Readfile is defined.
    # The definition Readfile reads the data file from a chi simulation, which
    # is one big giant string and has the following form
    # ['1\t 4\t 2\n'], this is an example, \t = tab, \n =  new line
    # These are the different variables that the chi files give at the end of the file.
    # For example the throughput, flow time, WIP, etc
    # This is transformed in one long line.
    #   1   4   2
    # Numlist changes it back in a list with strings
    #   ['1', '4', '2']
    # Finally, the values are changed in reals.
    #   [1.0, 4.0, 2.0]
def Readfile(fname):
    fp=open(fname,'r')
    line=string.strip(fp.readlines()[-1])
    fp.close
    x=[]
    numlist=string.split(line,'\t')
    for i in numlist:
        x.append(float(i))
    return x

# ##############################################################################
#                           Main program
# ##############################################################################
    # The following loop examines the given command line with which
    # RUN.py was run and stores the arguments in paramstr.
paramstr=''
for i in range(len(sys.argv[1:])):
    paramstr=paramstr+' '+repr(sys.argv[i+1])

    # paramstr. is extended and now contains the command line
    # to run the chi file with and stores screen output in to
    # the file data.
paramstr=paramstr+ ' >> data.txt'

    # If a file data allready exists it is deleted.
if os.path.isfile('data.txt'):
    os.unlink('data.txt')


n=0                                 # The simulation number
```

```
while 1:                                # Do forever until break:
    os.system('%s' % paramstr)          # Runs the unix command paramstr
    n=n+1                               # Updates the simulation number
    print n                             # Prints the simulation number
    lastx=[]                            # Makes an empty list
    lastline=Readfile('data.txt')       # Reads the last line of the data file
    for i in range(len(lastline)):      # The data is transferred to lastx
        lastx.append(lastline[i])

    if n==1:                            # If this is the first simulation run
        xm=[]                           # Make an empty list xm, at xm(i) it will
                                        # contain the average of value data(i)
                                        # over n simulations
        xv=[]                           # Make an empty list xm, at xv(i) it will
                                        # contain the st. deviation of value data(i)
                                        # over n simulations
        conf=[]                         # Half lenght confidence interval
        c=[]                            # Estimate relative error

        for i in range(len(lastx)):
            xm.append(lastx[i])
            xv.append(0)
            conf.append(0)
            c.append(0)

    else:                               # If this is not the first simulation run
        for i in range(len(lastx)):
            xv[i]=(n-2)*xv[i]/(n-1)+(lastx[i]-xm[i])*(lastx[i]-xm[i])/n   # Computes the st. deviations
            xm[i]=((n-1)*xm[i]+lastx[i])/n                                # Computes the averages

    if n>29:                            # If more then 29 simulations have been performed
        for i in range(len(lastx)):
            conf[i]=math.sqrt(xv[i])*tvalue[n-30]/math.sqrt(n)            # Computes the half-lengths
            c[i]=conf[i]/xm[i]                                            # Computes the relative errors

        maxconf= max(c)                                                   # Returns the max. rel. error

        if maxconf<0.05:                # if all relative errors are smaller than 0.05:
            xm_output=''
            xv_output=''
            for i in range(len(xm)):                                      # The results are transformed
                xm_output = xm_output+str(xm[i])+'\t'                     # in a string and printed.
                xv_output = xv_output+str(xv[i])+'\t'
            print 'xm: \n'
            print xm_output
            print 'xv: \n'
            print xv_output
            break                                                         # Stop!
```

wrfile.ext

This file is used for exporting data from the χ-files to Python.

```
language "python"
file "wrfile"

ext fileini(w, b, p :nat) -> nat
ext outfilebuffer(t:real, w, b , s :nat) -> nat
ext outfilecyct(n :nat, avc :real ) ->nat
```

wrfile.py

```
# wrfile.py writes data from chi to files.
```

```
def fileini(w,b,p):
    f=open('W'+str(w)+'B'+str(b)+'.txt', 'w')
    f.write('0'+'\t'+'0\n')
    f.close()
    i=open('E.txt','w')
    i.close()
    return 0

def outfilebuffer(t,w,b,s):
    f=open('W'+str(w)+'B'+str(b)+'.txt', 'a')
    f.write(str(t)+'\t'+str(s)+'\n')
    f.close()
    return 0

def outfilecyct(n, avc):
    f=open('E.txt','a')
    f.write(str(n)+'\t'+str(avc)+'\n')
    f.close()
    return 0
def fileini(w,b,p):
    f=open('W'+str(w)+'B'+str(b)+'.txt', 'w')
    f.write('0'+'\t'+'0\n')
    f.close()
    i=open('E.txt','w')
    i.close()
    return 0

def outfilebuffer(t,w,b,s):
    f=open('W'+str(w)+'B'+str(b)+'.txt', 'a')
    f.write(str(t)+'\t'+str(s)+'\n')
    f.close()
    return 0

def outfilecyct(n, avc):
    f=open('E.txt','a')
    f.write(str(n)+'\t'+str(avc)+'\n')
    f.close()
    return 0
```

D.3 χ files

SAVvar.chi

```
// SAVvar.chi simulates the SAV policy for a system with a triangular distribution
// for the process and arrival rates.

from std import *
from fileio import *
from random import *
from wrfile import *

type job         = nat # nat # real # path
                 // A job contains one part that is used in this simulation.
                 // < part-type, ID, time job enters the system, the production path >
   , path        = (nat # nat # real # nat # real^3)*
                 // A path describes the production path that a part follows.
                 // [ < workstation, the #th visit to the workst., transp. delay to the workst.
                 // , batch size transp., proc. time > ]
   , infor       = nat # real # real
                 // This is information send between controller and machine.
                 // < buffer, proc. rate, set-up time >
   , batchbuffer = ( ( (job*)^maxbuf )^numpart )^numworkst
```

```
                // Parts are "buffered" in this tuple for transport batching. This tuple is larger
                // than needed. and is used by the transporter process.
   , transplist  = (job* # real)*
                //  A list of parts that are in transport.
                // [ < job, transport delay > ],
   , datatype    = real # (real^3)^numpart # path^numpart # real^numworkst
                // This contains almost all the data that is needed for the simulation.
                // < period T, <| the inter arrival time of a part type  |>
                // , <| the production paths of the different part-types| >, setup times workstation >

                // Information about the systems enters the simulation via the variable
                // data: datatype.
const  dt     : nat = 0      // The position where period T information can be found in data
     , dta    : nat = 1      // The position where inter arrival time information can be found in data
     , dpath  : nat = 2      // The position where prod. path information can be found in data
     , dsetup : nat = 3      // The position where setup information can be found in data

// ------------------------------------------------------------
// The following parameters need to be provided
// ------------------------------------------------------------
     , maxbuf    : nat = 10      // The max. number of buffers found at a workstation
     , numpart   : nat = 10      // The number of different part-types
     , numworkst : nat = 1       // The number of workstations

// ------------------------------------------------------------
// Generator cluster
// ------------------------------------------------------------
// ------------- Generator
// The generator releases parts into the system.
// There are as many generators as part-types

proc G (o: !job, part:nat, data: datatype) =
|[ id: nat, ta: real^3
 , u: -> real
 | id:= 1                                        // G loads the inter-arrival time tuple containing
 ; ta:= data.dta.part                            // the values for triangular distribution.
 ; u:= triangle(ta.0, ta.1, ta.2)                // The triangular distribution
 ; *[ true ; delta sample(u)                     // Stochastic case: do forever: wait sample and send lot
// ; *[ true ; delta sample(u)                   // Deterministic case: wait ta and send lot
      -> o!< part, id, time, data.dpath.part >
       ; id:= id + 1
    ]
]|


// ------------- generator cluster
clus GEN (o: (!job)^numpart, data: datatype) =
|[ j:nat <- 0..numpart: G (o.j, j, data)]|

// ------------------------------------------------------------
// Workstation cluster
// ------------------------------------------------------------
// ------------- Controller
// The controller has a double function:
// 1) It makes sure that the parts arrive in the proper buffer
// 2) It controls which buffer is used by the machine.

proc C (i: ?job*, im: ?void, ib: (?void)^maxbuf, ob1: (!job*)^maxbuf
       , om: !infor, workstation: nat, data: datatype) =
// ------------- Type definition
|[ switch, included: bool, j, buffer, visit, n
 , q: nat, bufseq: nat*, T, msetup, sutm, ts, te, sut: real
 , xs: path
 , u: (-> real)^maxbuf                           // Stochastic case
// , u: (real)^maxbuf                           // Deterministic case
 , tp: (real^3)
```

```
 , pt: (real)^maxbuf
 , lotbuf: (nat)^maxbuf
 , bufinfo: (nat^2)^maxbuf, lots: job*
 , info: infor
// ------------- Initialization
// The controller runs through data and checks which part-type visits
// this workstation and how many times. For each time that a part-type
// visits this workstation a buffer is included in the buffer sequence.
 | switch:= false; j:=0; buffer:= 0; T:=data.dt
 ; msetup:= data.dsetup.workstation
 ; bufseq:= []
 ; *[ j <  numpart                                // The number of part types that need to be checked
      -> xs:= data.dpath.j; visit:= 1             // A part type can never visit a workstation more then
       ; *[ visit <= maxbuf                       // the max. num. of buffers present at a workst.
            -> < included, n >:= workstationincluded(xs, workstation, visit)
             ; [ included                         // If a part type visits the workstation it:
                 -> bufseq:= bufseq ++ [buffer]   // Adds a buffer to the buffer sequence
                  ; tp:= idx(xs, n).4             // loads the info of that production step
                  ; u.buffer:= triangle(tp.0, tp.1, tp.2) // if stochastic, stores the proc. distr.
//                  ; u.buffer:= tp.1             //  if deterministic, stores the proc. time
                                                  // computes max. length of a run of a part-type
                  ; pt.buffer:=data.dt * (tp.2) / (data.dta.j.0)  // stochastic case
//                  ; pt.buffer:= data.dt * tp.1 / (data.dta.j.1) // deterministic case
                  ; bufinfo.buffer:= <| j, visit |>    // it is stored which part visits what buffer
                  ; q:=fileini(workstation, buffer, j) // Info. on the buffers send to Python
                  ; buffer:= buffer + 1
               | not included                      // If a part type does not visit this workst.
                 ->  skip                         // skip
               ]
             ; visit:= visit + 1
         ]
       ; j:= j + 1
   ]
 ; sutm:= fsutm(bufseq, pt, T)
 ; [ sutm <  msetup                               // A build-in check
    -> !"Error, did you compute T right?\n","sutm: ",sutm,"\t msetup: ",msetup,"\n"
  | sutm >= msetup
    -> skip
  ]
 ; buffer:= 0                                     // The machine is set ready for buffer 0
 ; info:= < buffer, sample(u.buffer), sut >       // Stochastic case
// ; info:= < buffer, u.buffer, sut >             // Deterministic case
 ; om!info                                        // With this info. the machine can be initialized
                                                  // It contains info about from which buf. should
                                                  // be processed, the process time and the time
                                                  // of a possible setup
 ; ts:=time; te:= ts + pt.buffer                  // The end time of production is computed and
                                                  // stored, also the start time is stored.
// ------- End of initialization
 ; *[ true; i?lots
      -> j:= bufchan(hd(lots), bufseq, bufinfo); ob1.j!lots
       ; lotbuf.j:= lotbuf.j + len(lots)
       // Incoming parts are send to the proper buffer by checking there part
       // type and there #th visit. The controller registers the increase of
       // parts in the buffer
    | b: nat <-0..maxbuf: true; ib.b?
      -> lotbuf.b:= lotbuf.b - 1
       // The controller registers when a part leaves a buffer
    | true; im?
      -> [ switch                                 // Checks if a setup has taken place
           -> ts:=time; te:= ts + pt.buffer       // yes: computes new values
            ; switch:= false
         | not switch                             // not: does nothing
           -> skip
         ]
       ; [ te > time and lotbuf.buffer > 0        // Checks if it is time to setup
```

```
          -> skip                                  // not time: does nothing
        | te <= time or lotbuf.buffer <= 0
          -> switch:= true                         // When a setup needs to performed it
           ; sut:=sutm + pt.buffer - time + ts     // it is computed how long to the setup
           ; [ sut <  msetup                       // should be. If due to variability the
               -> sut:=msetup                      // computed setup is smaller than the minimal
             | sut >= msetup                       // setup this is corrected
               -> skip
             ]
           ; bufseq:= swap(bufseq)                 // The buf. seq. is updated
           ; buffer:=hd(bufseq)
        ]
      // The controller checks if it is time to perform a setup or if the
      // buffer being served is empty. If this is not true, the machine
      // stays processing parts from the current buffer. If this is true
      // the controller computes the necessary setup time and sends this
      // to the machine.
      ; info:= < buffer, sample(u.buffer), sut > // Stochastic case
//     ; info:= < buffer, u.buffer, sut >        // Deterministic case
      ; om!info
      // The controller sends information to the machine.
      // It contains info about from which buf. should  be processed,
      // the process time and the time of a possible setup.
   ]
]|

// ------------- Buffer
// The buffer processes and receives parts from the transport proces, via
// the workstation controller. If the parts were batched during transport,
// these batches are separated. Information about the buffer is written to a file.

proc B (ic1: ?job*, om: !job, oc: !void, buffer: nat, workstation: nat)=
|[ LOTS, lots: job*, x: nat, info: bool
 | LOTS:= []; info:= false
 ; *[ true; ic1?lots                                        // Receives parts
      -> *[ len (lots) > 0                                  // Stores parts
            -> LOTS:= LOTS ++ [hd(lots)]; lots:= tl(lots)]
       ; x:=outfilebuffer(time,workstation,buffer,len(LOTS)) // Writes info. to file
    | not info and len(LOTS) > 0; om!hd(LOTS)               // Sends parts
      -> LOTS:= tl(LOTS)                                    // Updates buf. info.
       ; x:=outfilebuffer(time,workstation,buffer,len(LOTS))
       ; info:=true
    | info; oc!                                             // Informs Controller
      -> info:= false
    ]
]|

// ------------- Machine
// The machine sends a requests to the controller that it is ready
// From the controller it receives information about the buffer it needs
// to use and information needed for processing a part. When the buffer
// it needs to use, is different then the previous one, the machine switches
// with the setup time it receives from the controller.

proc M ( ib: (?job)^maxbuf, ic: ?infor
       , o: !job, oc: !void, workstation: nat)=
|[ buffer: nat, lot: job, info: infor
// ------------- Initialization
 | ic?info
 ; buffer:= info.0
// ------- End of initialization
 ; *[ true -> oc!; ic?info                          // Requests and receives info.
           ; [ buffer = info.0                      // Switching is not needed
               ->  ib.buffer?lot; delta info.1      // Receives part from buffer and processes it
                ; lot.3:= tl(lot.3)                 // Updates the production path of the part
                ; o!lot                            // Part is send to transport process.
```

```
              | buffer /= info.0                  // Switching is needed
                -> delta info.2; buffer:= info.0  // Switch is performed.
              ]
    ]
 ]|

// ------------- Workstation cluster
clus W (i: ?job*, o: !job, workstation: nat, data: datatype) =
|[ cb1: (-job*)^maxbuf, bm: (-job)^maxbuf, cm:-infor
 , mc:-void, bc: (-void)^maxbuf
 | j:nat <- 0..maxbuf: B (cb1.j, bm.j, bc.j, j, workstation)
|| M (bm, cm, o, mc, workstation)
|| C (i, mc, bc, cb1, cm, workstation, data)
]|

// ------------------------------------------------------------
// The main cluster
// ------------------------------------------------------------
// ------------- Transport process
// The transport process, receives every job that is send from generator to workstation,
// from workstation to workstation and workstation to exit. If batching is required for transport,
// this process batches the jobs. Also transportation delays are taken into account.

proc Tr (ig: (?job)^numpart, i: (?job)^numworkst, o: (!job*)^numworkst, oe: !job) =
|[ xs: transplist, lot: job, LOTS: batchbuffer, lots: job*, workstation: nat
 | xs:=[]
 ; *[ j:nat <-0..numpart: true; ig.j?lot
      -> < xs, LOTS >:= batching(lot, xs, LOTS, time)
      // Receives jobs from the gen. and batches them and includes them in the transport list
    | k:nat <-0..numworkst: true; i.k?lot
      -> [ not finished (lot)
           -> < xs, LOTS >:= batching(lot, xs, LOTS, time)
         | finished (lot)
           -> oe!lot
         ]
      // Receives jobs from the workst. and batches them and includes them in the transport list
      // if the part is not finished, otherwise the part is send to the exit
    | len (xs) > 0; delta (hd(xs).1 - time)
      -> lots:= hd(xs).0; workstation:= hd(hd(lots).3).0
       ; o.workstation!lots; xs:= tl(xs)
    // When the transportation delay has finished, it sends the batch to the desired
    // workstation.
    ]
 ]|

// ------------- Exit proces
// The exit proces removes the parts from the system and it computes
// the average flow time. If necessary it writes the flow time of each
// part to a file.

proc E (i: ?job, numlots: nat, initial: nat, sim: nat) =
|[ tottp: nat, cyctot:  real, lot: job, x: nat
 | tottp:=0; cyctot:=0.0
 ; *[ tottp < numlots; i?lot                     // When less parts have exit the system then run length
      -> tottp:= tottp + 1                       // Update number of parts
       ; [ tottp <= initial
           -> skip
         | tottp > initial
           -> cyctot:= cycle(lot, tottp-initial, cyctot, time) // Computes av. flow tume
            ; [ sim = 0 -> x:=outfilecyct(tottp, time-lot.2)   // Writes the flow time of each product
              | sim > 0 -> skip
              ]
         ]
    ]
 ; !cyctot,"\n"
 ; terminate                                    // Terminates the simulation
```

```
]|

// ------------- Main cluster
clus GTSWBE(data: datatype, simtime: nat, initial: nat, sim: nat) =
|[ gt: (-job)^numpart, wt: (-job)^numworkst, tw: (-job*)^numworkst, te: -job
|  GEN (gt, data)
|| Tr (gt, wt, tw, te)
|| j: nat <-0..numworkst: W (tw.j, wt.j,j, data)
|| E (te, simtime, initial, sim)
]|

xper (data:datatype, simtime: nat, initial: nat, sim: nat) = |[GTSWBE(data, simtime, initial, sim)]|

// ------------------------------------------------------------
// Functions
// ------------------------------------------------------------
// ------------- swap function
// Updates the buffer sequence
func swap(s: nat*) -> nat* =
|[ ret tl(s) ++ [hd(s)] ]|

// ------------- fsutm function
// Calculates the minimal set-up time according to Savkin
func fsutm(bufseq: nat* , pt: real^maxbuf, T: real) -> real =
|[ x: real, n : nat
 | x:=0.0; n:=0
 ; *[n < len(bufseq) -> x:= x + pt.n; n:=n+1]
 ; ret (T-x)/n
]|

// ------------- bufchan function
// The function returns to the controller to which buffer a part
// should be send when it arrives at the workstation. It investigates
// if the part type and the visit number correspond with the information
// stored in bufinfo. When there is a match, the buffer number is returned.
func bufchan (lot: job, bufseq: nat*, bufinfo: (nat^2)^maxbuf ) -> nat =
|[ n: nat
 | n:= 0
 ; *[ len (bufseq) > 0 and (lot.0 /= bufinfo.n.0 or hd(lot.3).1 /= bufinfo.n.1)
      -> n:= n+1; bufseq:= tl(bufseq)
    ]
 ; ret n
]|

// ------------- cycle function
// Calculates the flow time of a part
func cycle(x: job, y: nat, cyc: real, t: real) -> real =
|[ ret (y-1) / y * cyc + (t - x.2) / y]|

// ------------- finished function
// Checks if a part is finished or if it still has to undergo
// production steps.
func finished (lot:job) -> bool =
|[ [ len (lot.3) = 0 -> ret true
   | len (lot.3) > 0 -> ret false
   ]
]|

// ------------- batching function
// This function is used by the transport process and batches the parts
// for transport if this is needed. It first adds the incoming part to list of
// the same parts. If this list contains enough parts they are made ready for transport.
// The function returns the tuple containing all parts which are waiting to be batched and
// a list containing the batches which are in transport and the time when there transport is
// finished.
func batching (lot: job, xs: transplist, LOTS: batchbuffer, t: real)
```

```
             -> transplist # batchbuffer =
|[ part, workstation, visit, batchsize: nat, lots: job*
 | part:=lot.0; workstation:= hd(lot.3).0
 ; visit:= hd(lot.3).1-1; batchsize:= (hd(lot.3)).3
 ; LOTS.workstation.part.visit:= LOTS.workstation.part.visit ++ [lot]
 ; lots:= LOTS.workstation.part.visit
 ; [ len (lots) = batchsize -> xs:= transport(xs, lots, t); LOTS.workstation.part.visit:= []
   | len (lots) < batchsize -> skip
   ]
  ; ret <xs,LOTS>
]|

// ------------- transport function
// Returns a list of batches in transport sorted by transport delay.
func transport (xs: transplist, lots: job*, t:real) -> transplist=
|[ ret insert(xs,<lots,t+hd(hd(lots).3).2>,inc) ]|

// ------------- inc function
// This function is necessary for the transport function.
func inc(x, y: job*#real) -> bool=
|[ ret x.1 < y.1 ]|

// ------------- workstationincluded function
// Checks if the workstation is included in the production path of a part
// and if the visit corresponds to the given visit.
func workstationincluded(xs:path, workstation:nat, visit:nat) -> bool#nat =
|[ b: bool, n: nat
 | b:=false ; n:= 0
 ;*[ not b and len(xs) > 0
     ->  b:= (hd(xs).0 = workstation and hd(xs).1 = visit)
      ;  xs:=tl(xs); n:= n+1]
 ; ret < b, n-1 >
 ]|
```

CLWvar.chi

Most of the CLWvar.chi file is similar to the SAVvar.chi file. Only code that differs from SAVvar.chi is denoted below. The controller process and the `bufchan` function differ as the ones defined in SAVvar.chi and the functions `LW`, `CLSA` and `sum` are added. Furthermore, the `swap` and `fsutm` function are not necessary in CLWvar.chi.

```
// ------------------------------------------------------------
// Workstation cluster
// ------------------------------------------------------------
// ------------- Controller
// The controller has a double function:
// 1) It makes sure that the parts arrive in the proper buffer
// 2) It controls which buffer is used by the machine.

proc C (i: ?job*, im: ?void, ib: (?void)^maxbuf   // Channels
       , ob1: (!job*)^maxbuf
       , om: !infor, workstation: nat
       , data: datatype) =
// ------------- Type definition
|[ included: bool, j, buffer, visit, n
 , nbuf, q: nat, msetup: real, xs: path
 , u: (-> real)^maxbuf                           // Stochastic case
// , u: (real)^maxbuf                           // Deterministic case
 , tp1: (real^3)
 , tp, ta : (real)^maxbuf
 , lotbuf: (real*)^maxbuf
 , bufinfo: (nat^2)^maxbuf, lots: job*
 , info: infor
// ------------- Initialization
```

```
// The controller runs through data and checks which part-type visits
// this workstation and how many times. For each time that a part-type
// visits this workstation a buffer is assigned to it.
 | j:=0; buffer:= 0; nbuf:= 0
 ; msetup:= data.dsetup.workstation
 ; *[ j <  numpart                                // The number of part types that need to be checked
      -> xs:= data.dpath.j; visit:= 1             // A part type can never visit a workstation more then
       ; *[ visit <= maxbuf                       // the max. num. of buffers present at a workst.
            -> < included, n >:= workstationincluded(xs, workstation, visit)
             ; [ included                         // If a part type visits the workstation it:
                 -> tp1:= idx(xs, n).4            // loads the info of that production step
                  ; u.buffer:= triangle(tp1.0, tp1.1, tp1.2) // if stochastic, stores the proc. distr.
//                  ; u.buffer:= tp1.1            //  if deterministic, stores the proc. time
                  ; tp.buffer:=tp1.1
                  ; ta.buffer:= data.dta.j.1
                  ; bufinfo.buffer:= <| j, visit |>    // it is stored which part visits what buffer
                  ; q:=fileini(workstation, buffer, j) // Info. on the buffers send to Python
                  ; buffer:= buffer + 1
               | not included                      // If a part type does not visit this workst.
                 ->  skip                         // skip
               ]
             ; visit:= visit + 1
         ]
       ; j:= j + 1
   ]
 ; nbuf:= buffer                                  // The number of buf. at this workst.
 ; buffer:= 0                                     // The machine is set ready for buffer 0
 ; info:= < buffer, sample(u.buffer), msetup >    // Stochastic case
// ; info:= < buffer, u.buffer, msetup >          // Deterministic case
 ; om!info                                        // With this info. the machine can be initialized
                                                  // It contains info about from which buf. should
                                                  // be processed, the process time and the time
                                                  // of a possible setup
// ------- End of initialization
 ; *[ true; i?lots
      -> j:= bufchan(hd(lots), nbuf, bufinfo); ob1.j!lots
       ; *[ len (lots) > 0
            -> lotbuf.j:= lotbuf.j ++ [time]; lots:= tl(lots) ]
       // Incoming parts are send to the proper buffer by checking there part
       // type and there #th visit. The controller registers the increase of
       // parts in the buffer
    | b: nat <-0..maxbuf: true; ib.b?
      -> lotbuf.b:= tl(lotbuf.b)
       // The controller registers when a part leaves a buffer
    | c: nat <-0..maxbuf: len(lotbuf.c) > 0; im?
      -> [ len(lotbuf.buffer) > 0
           -> skip
         | len(lotbuf.buffer) <= 0
           -> buffer:= LW(lotbuf, tp, ta, nbuf, msetup, time)
         ]
       // The controller checks if there are buffers which are not empty.
       // If the buffer from which the machine is processing is not empty
       // do nothing, but if this buffer is empty, then it is computed which
       // of the non-empty buffers has the largest scaled age.
       ; info:= < buffer, sample(u.buffer), msetup > // Stochastic case
//       ; info:= < buffer, u.buffer, msetup >        // Deterministic case
       ; om!info
       // The controller sends information to the machine.
       // It contains info about from which buf. should  be processed,
       // the process time and the time of a possible setup.
   ]
]|

// ---------------------------------------------------------
// Functions
// ---------------------------------------------------------
```

```
// ------------- LW function
// Calculates which of the the buffers contains the most work.
func LW (lotbuf: (real*)^maxbuf, tp, ta: real^maxbuf, nbuf: nat, msetup, t: real) -> nat =
|[ i, a: nat, x:  nat # real
 | i:= 0; x:= < 0, 0.0>
 ; *[ i <  nbuf
      -> [ x.1 > (len(lotbuf.i)) * tp.i
           -> skip
         | x.1 = (len(lotbuf.i)) * tp.i
           -> a:= CLSA( <| lotbuf.(x.0), lotbuf.i |>,  <| tp.(x.0), tp.i |>, <| ta.(x.0), ta.i |>, 2, msetup, t)
            ; [ a = 0
                -> skip
              | a = 1
                -> x:= < i, (len(lotbuf.i)) * tp.i >
              ]
         | x.1 < (len(lotbuf.i)) * tp.i
           -> x:= < i, (len(lotbuf.i)) * tp.i >
         ]
       ; i:= i + 1
    ]
 ; ret x.0
]|


// ------------- CLSA function
// Calculates which of the non-empty buffers contains the largest work
// and returns that buffer.
func CLSA (lotbuf: (real*)^2, tp, ta: real^2
          , nbuf: nat, msetup, t: real) -> nat =
|[ i: nat, x:  nat # real, age: real, w: real
 | i:= 0; x:= < 0, 0.0>
 ; *[ i <  nbuf
      -> age:= msetup^2 / (2* ta.i) +  msetup * len(lotbuf.i) + len(lotbuf.i)*t - fold (lotbuf.i, sum, 0.0)
       ; w:= 1 / (msetup * (1 - tp.i / ta.i))     // Weight factor
       ; [  x.1 >= age * w or len(lotbuf.i) <= 0  // Buffer is empty or has smaller scaled age
            -> skip
         |  x.1 < age * w and len(lotbuf.i) > 0   // Buffer is  not empty and has larger scaled age
            -> x:= < i, age * w >
         ]
       ; i:= i + 1
    ]
 ; ret x.0
]|

// ------------- sum function
// Sums to reals
func sum(x , y: real) -> real =
|[ ret x + y ]|

// ------------- bufchan function
// The function returns to the controller to which buffer a part
// should be send when it arrives at the workstation. It investigates
// if the part type and the visit number correspond with the information
// stored in bufinfo. When there is a match, the buffer number is returned.
func bufchan (lot: job, nbuf: nat, bufinfo: (nat^2)^maxbuf ) -> nat =
|[ i: nat
 | i:= 0
 ; *[ i < nbuf and (lot.0 /= bufinfo.i.0 or hd(lot.3).1 /= bufinfo.i.1)
      -> i:= i+1
    ]
 ; ret i
]|
```

CLSAvar.chi

Most of the CLSAvar.chi file is similar to the CLWvar.chi file. Code that differs from SAVvar.chi is denoted below. Only the controller process and the `CLSA` function differ slightly from CLWvar.chi. Therefore, for the controller process only the part that differs is shown.

```
proc C(....) =
|[ ...
 | ...
 ; ...
 ; ....
    | c: nat <-0..maxbuf: len(lotbuf.c) > 0; im?
      -> [ len(lotbuf.buffer) > 0
           -> skip
         | len(lotbuf.buffer) <= 0
           -> buffer:= CLSA(lotbuf, tp, ta, nbuf, msetup, time)
         ]
 ; ....
]|

// -------------------------------------------------------------
// Functions
// -------------------------------------------------------------
// ------------- CLSA function
// Calculates which of the non-empty buffers contains the largest work
// and returns that buffer.
func CLSA (lotbuf: (real*)^maxbuf, tp, ta: real^maxbuf
          , nbuf: nat, msetup, t: real) -> nat =
|[ i: nat, x:  nat # real, age: real, w: real
 | i:= 0; x:= < 0, 0.0>
 ; *[ i <  nbuf
      -> age:= msetup^2 / (2* ta.i) +  msetup * len(lotbuf.i) + len(lotbuf.i)*t - fold (lotbuf.i, sum, 0.0)
       ; w:= 1 / (msetup * (1 - tp.i / ta.i))     // Weight factor
       ; [  x.1 >= age * w or len(lotbuf.i) <= 0  // Buffer is empty or has smaller scaled age
            -> skip
         |  x.1 < age * w and len(lotbuf.i) > 0   // Buffer is  not empty and has larger scaled age
            -> x:= < i, age * w >
         ]
       ; i:= i + 1
   ]
 ; ret x.0
]|
```

No variability

A simulation for a system without variability can be performed with the three χ-files depict above. Only some small modifications are needed. These modification are already shown in the code, but they are commented. Uncommenting them and commenting there stochastic counterparts results in simulation for a system without variability. However, for SAVvar.chi some other small modifications need to be made in the controller proces. The following part can be deleted.

```
           ; [ sut <  msetup
               -> sut:=msetup
             | sut >= msetup
               -> skip
             ]
```

This part is not necessary anymore, because due to deterministic behavior the computed setup time can not become smaller than the minimal setup time. Furthermore, errors can arris due to truncation. Therefore, the following needs to be added.

```
proc C(....) =
|[ ...
 | ...
 ; ...
 ; ....
    | true; im?
      -> [ switch
           -> ts:=time; te:= ts + pt.buffer; switch:= false
         | not switch
           -> skip
         ]
       ; [ te > time and lotbuf.buffer > 0        // Checks if it is time to setup
           -> [ te-time < 1e-6 -> !time, "\t Error: ", te-time,"\n"
                               ; terminate
              | te-time >= 1e-6 -> skip
              ]
         | te <= time ...
 ; ...
]|
```

Note that due to the lack of variability the following can happen. Assume that the machine is processing parts from buffer 1 and that at $t = 30$ the machine sends a request signal to the controller of workstation 1 that it is ready to process a part. Furthermore, a part arrives at buffer 1, which the controller needs to register. Thus, at $t = 30$ two events take place which the controller needs the process, namely:

- the controller receives a request signal form the machine.
- the controller registers that a parts arrives at buffer 1.

Which event takes place "first" at $t = 30$ is determined non deterministically by χ. This, can cause problems if buffer 1 is empty at $t = 30$. If the request of the machine is processed first, the controller will start a setup, because the buffer is empty. But if the parts arrival is registered first, the machine will not perform a setup.

Initial buffer levels

During the experiments only the initial levels of the first buffer that a part visits has been changed. Thus, only the $x_{p,1}(0)$ needs the be controlled. Therefore, the following constants need to be defined.

```
    , lotsini   : nat^1 = <| 367 |>       //  The initial parts that need to be generated.
    , bufini    : (nat^2)^2 = <| <|367,0|>,<|0,0|> |>
    // <| ini. buf. lev. of buffers at a workstation, ini. buf. lev. of buffers at a workstation |>
    // The initial buffer levels, only introduce initial buffer levels for the buffers
    // that are visited first for a part, thus only b_{p,1}.
```

The size of the tuples depend on the number of parts types, the maximum number of buffers found at a workstation and the number of workstations. However, the type definition of a constant can not cope with an earlier defined constant. Therefore the following is not possible.

```
    , lotsini   : nat^numpart = ...
    , bufini    : (nat^maxbuf)^numworkst = ...
```

The following needs to be added to the generator process of all the policies.

```
proc g(...
|[ ...
 | ...
 ; u:= triangle(ta.0, ta.1, ta.2)                // The triangular distribution
 ; *[ id <= lotsini.part                         // Initial buffer level
      -> o!< part, id, time, data.dpath.part >
       ; id:= id + 1 ]
 ; *[ true ; delta ...
 ; ...
]|
```

Furthermore, for CLW or CLSA the following needs to be added to the controller part.

```
proc C(....) =
|[ ...
 | ...
 ; ...
 ; ....
 ; om!info                                       // With this info. the machine can be initialized
                                                 // It contains info about from which buf. should
                                                 // be processed, the process time and the time
                                                 // of a possible setup
 ; *[ c: nat <-0..maxbuf: len(lotbuf.c) < bufini.workstation.c; i?lots
      -> j:= bufchan(hd(lots), nbuf, bufinfo); ob1.j!lots
       ; *[ len (lots) > 0
            -> lotbuf.j:= lotbuf.j ++ [time]; lots:= tl(lots) ]
    ]
    // This ensures that the buffers have the desired buffer levels.
// ------- End of initialization
 ; *[ true; i?...
 ; ...
]|
```

For SAV the following has to be ad.

```
proc C(....) =
|[ ...
 | ...
 ; ...
 ; ....
 ;  ts:=time; te:= ts + pt.buffer                 // The end time of production is computed and
                                                 // stored, also the start time is stored.
 ; *[ c: nat <-0..maxbuf: lotbuf.c < bufini.workstation.c; i?lots
      -> j:= bufchan(hd(lots), bufseq, bufinfo); ob1.j!lots
       ; lotbuf.j:= lotbuf.j + len(lots)
    ]
    // This ensures that the buffers have the desired buffer levels.
// ------- End of initialization
 ; *[ true; i?...
 ; ...
]|
```

Regulators

The buffer process, control process and the workstation cluster are modified in order to introduce regulators. Below the modifications for CLWvar.chi are shown.

```
// ------------------------------------------------------------
// Workstation cluster
// ------------------------------------------------------------
// ------------- Controller
// The controller has a double function:
// 1) It makes sure that the parts arrive in the proper buffer
```

```
// 2) It controls which buffer is used by the machine.

proc C (i: ?job*, im: ?void, ib1: (?void)^maxbuf, ib2: (?void)^maxbuf
       , ob1: (!job*)^maxbuf , ob2: (!real)^maxbuf
       , om: !infor, workstation: nat
       , data: datatype) =
// ------------- Type definition
|[ included: bool, ...
 | ...
 ; ...
 ; ...
                  ; ta.buffer:= data.dta.j.1
                  ; bufinfo.buffer:= <| j, visit |>    // it is stored which part visits what buffer
                  ; q:=fileini(workstation, buffer, j) // Info. on the buffers send to Python
                  ; ob2.buffer!ta.buffer               // Arrival time info send to the buffer
                  ; buffer:= buffer + 1
 ; ...
// ------- End of initialization
 ; *[ true; i?lots
      -> j:= bufchan(hd(lots), nbuf, bufinfo); ob1.j!lots
       // Incoming parts are send to the proper buffer by checking there part
       // type and there #th visit.
    | b:nat <-0..maxbuf: true; (ib1).b?
      -> lotbuf.b:= lotbuf.b ++ [time]
       //  The controller registers the increase of the regulated buffer
       // lots in the buffer
    | d:nat <-0..maxbuf: true; (ib2).d?
      -> lotbuf.d:= tl(lotbuf.d)
      // The controller registers when a part leaves a regulated buffer
    | c: nat <-0..maxbuf:...
 ; ...
 ]|

// ------------- Buffer
// The buffer processes and receives parts from the transport proces, via
// the workstation controller. If the parts were batched during transport,
// these batches are separated. Information about the buffer is written to a file.

proc B (ic1: ?job*, ic2: ?real, om: !job, oc1, oc2: !void, buffer: nat, workstation: nat)=
|[ LOTS1, LOTS2, LOTStot, lots: job*, x: nat
 , info1, info2: bool, tend, ta : real
 | LOTS1:= []; LOTS2:= []; LOTStot:= []; info1:= false; info2:= false
 ; ic2?ta; tend:=0.0
 ; *[ true; ic1?lots                                        // Receives parts
      -> *[ len (lots) > 0                                  // Stores parts in the
            -> LOTS1:= LOTS1 ++ [hd(lots)]                  // regulator buffer
             ; LOTStot:= LOTStot ++ [hd(lots)]              // total buffer
             ; lots:= tl(lots)
         ]
       ; x:=outfilebuffer(time,workstation,buffer,len(LOTStot))   // Writes info. to file
    | not info1 and len(LOTS1) > 0 ; delta (max(tend - time,0.0)) // Parts move from the regulator
      -> LOTS2:= LOTS2 ++ [hd(LOTS1)]; LOTS1:= tl(LOTS1)          // to the regulated buffer
       ; info1:= true
    | info1; oc1!                                               // The controller is informed
      -> info1:= false; tend:= time + ta
    | not info2 and len(LOTS2) > 0; om!hd(LOTS2)                // Sends parts
      -> LOTS2:= tl(LOTS2); LOTStot:= tl(LOTStot)               // Updates buf. info.
//       ; x:=outfilebuffer(time,workstation,buffer,len(LOTStot))
       ; info2:=true
    | info2; oc2!                                               // The controller is informed
      -> info2:= false
    ]
]|

proc M(...
|[ ...
```

```
 | ...
 ; ...
]|

// ------------- Workstation cluster
clus W (i: ?job*, o: !job, workstation: nat, data: datatype) =
|[ cb1: (-job*)^maxbuf, bm: (-job)^maxbuf, cb2:(-real)^maxbuf, cm:-infor
 , mc:-void, bc1,bc2: (-void)^maxbuf
 | j:nat <- 0..maxbuf: B (cb1.j, cb2.j, bm.j, bc1.j, bc2.j, j, workstation)
|| M (bm, cm, o, mc, workstation)
|| C (i, mc, bc1, bc2, cb1, cb2,cm, workstation, data)
]|
```

For CLSAvar.chi the same modification need to be made.

Appendix E

Optimization model

In this appendix the file is presented used to solve the optimization problem of Section 6.2.

optimization.m

```
clear all
close all
clc

% ------------------------------------------------------------
%                   Input of parameters
% ------------------------------------------------------------

LAMBDA=[1 2];                   % The arrival rates
MU=[3 6 ];                      % The process rates
theta=2;                        % The setup time
epsilon= 1e-3;                  % Epsilon
X0=[10 10];                     % The initial buffer levels


% ------------------------------------------------------------
%               Calculate the necessary parameters
% ------------------------------------------------------------
%
j=length(LAMBDA);               % The number of buffers
T=j*theta/(1-sum(LAMBDA./MU));  % The scheduling period
D=LAMBDA*T;                     % Desired production levels

% ------------------------------------------------------------
%                   Ideal Savkin Trajectory
% ------------------------------------------------------------
    % Suppose that the buffer sequence of Savkin is X(1)->X(2)->...->X(end)->X(1)
    % and the machine is ready to start with production from X(1)
    % then the ideal initial buffer levels are:
for i=1:j
    Xideal(1,i)=LAMBDA(i)*(T-(i-1)*theta-sum(D(1:i)./MU(1:i)));
end

    % Calculating the ideal Savkin trajectory:
Tideal=[0];
for i=1:j
    dt=D(i)/MU(i);                                      % The production time from buffer i
    Tideal=[Tideal; Tideal(end)+dt];                    % Store the time
    Xideal=[Xideal; Xideal(end,:)+LAMBDA*dt];           % Arrival of work in that period
    Xideal(end,i)=0;                                    % The buffer is completely emptied
    % A switch to the next buffer, a setup takes place:
```

```
    Tideal=[Tideal; Tideal(end)+theta];                 % Store the time
    Xideal=[Xideal; Xideal(end,:)+LAMBDA*theta];        % The work that arrives during the setup
end
Xideal0=Xideal(1,:);                                    % The 'initial buffer level of the ideal sequence

% ------------------------------------------------------------
%                   cyclic clearing policy
% ------------------------------------------------------------

Tclear=[0];                                             % Store the time
Xclear=[X0];                                            % Store the buffer levels
    % The distance covered:
Diffclear=[norm(Xclear(1,:)-Xideal0)-norm(Xclear(end,:)-Xideal0)];
    % The distance between the start of the ideal sequence and the start of
    % cyclic clearing policy sequence:
Distclear=[norm(Xclear(end,:)-Xideal0)];
    % The ratio beteeen the distance covered and the time:
Ratioclear=[Diffclear(end)/Tclear(end)];
tclear=[];
clearingcycles=0;                                        % The number of cycles
    % Start a new cycle if the ideal Savkin trajectory has not been
    % sufficienly approached yet:
while norm(Xclear(end,:)-Xideal0) > epsilon
    clearingcycles=clearingcycles+1;                     % Increase the number of cycles
    fprintf('Starting clearing cycle: %g\n',clearingcycles)
    for i=1:j
        dt=Xclear(end,i)/(MU(i)-LAMBDA(i));              % The production time of buffer i
        tclear=[tclear; dt];                             % Store the production time
        Tclear=[Tclear; Tclear(end)+dt];                 % Store time
        Xclear=[Xclear; Xclear(end,:)+LAMBDA*dt];        % Arrival of work in that period
        Xclear(end,i)=0;                                 % The buffer is completely emptied
    % A switch to the next buffer, a setup takes place:
        Tclear=[Tclear; Tclear(end)+theta];              % Store time
        Xclear=[Xclear; Xclear(end,:)+LAMBDA*theta];     % The work that arrives during the setup
    end
    % Update Distclear, Diffclear and Ratioclear:
    Distclear=[Distclear; norm(Xclear(end,:)-Xideal0)];
    Diffclear=[Diffclear; norm(Xclear(1,:)-Xideal0)-norm(Xclear(end,:)-Xideal0)];
    Ratioclear=[Ratioclear; Diffclear(end)/Tclear(end)];
end
fprintf('The cyclic clearing policy reached the ideal Savkin trajectory in: %g',...
    ((length(Tclear)-1)/4))
fprintf(' cycles.\n')

% ------------------------------------------------------------
%                         Optimization
% ------------------------------------------------------------
ALPHA=LAMBDA./(MU-LAMBDA);                               % Calculating alpha
A=diag(ones(j,1));
A2=zeros(j,j);
s=X0'./(MU-LAMBDA)';                                     % Calculating s
    % Using the solution from the cyclic clearing policy as initial guess:
topt0=tclear(1:j,1);
for z=2:j
    A=A+diag(-ALPHA(z:end),-z+1);                        % Calculating A
    A2=A2+diag(-ALPHA(1:j-(z-1)),z-1);
    s(z)=s(z)+(z-1)*theta*ALPHA(z);                      % Calculating s
end
A2=A+A2;                                                 % Calculating A2
i=1;                                                     % Number of cycles
exitflag=0;
options=optimset;
options=optimset(options,'Diagnostics','off','Display','off',...
    'largescale','off','TolCon',1e-6,'TolFun',1e-6);
    % Start a new cycle if the ideal Savkin trajectory has not been
    % sufficiently approached yet:
```

```
while exitflag<=0
    fprintf('Running optimization algorithm for %g',i)
    fprintf(' cycles\n')
    if i>1                                              % Update A,en s
        A=[A zeros((i-1)*j,j); A2 A(end-(j-1):end,:)];
        s=[s; s(end-(j-1):end)+j*theta*ones(j,1).*ALPHA'];
        topt0=tclear(1:j*i,1);                          % Update gues
    end
    % Run the optimization algorithm:
    [topt, fvalopt, exitflag]= fmincon(@shortesttime,topt0,A,s,[],[],0,[],...
        @nonlconstraint,options,j,i,theta,MU,LAMBDA,X0,Xideal0,epsilon);
    i=i+1;                                              %  Update the number of cycles
end

    %Calculate the position that now has been reached:
numofcycle=i-1;
Xopt=X0;
Topt=[0];
    % The distance covered:
Distopt=[norm(Xopt(end,:)-Xideal0)];
    % The distance between the start of the ideal sequence and the start of
    % cyclic clearing policy sequence:
Diffopt=[norm(Xopt(1,:)-Xideal0)-norm(Xopt(end,:)-Xideal0)];
    % The ratio beteeen the distance covered and the time:
Ratioopt=[Diffopt(end)/Topt(end)];
for i=1:numofcycle
    for z=1:j;
        dt=topt((i-1)*j+z);                             % The production time of buffer i
        Xopt=[Xopt; Xopt(end,:)+LAMBDA*dt];             % Arrival of work in that period
        Xopt(end,z)=Xopt(end,z)-MU(z)*dt;               % Amount of work removed in that period
        Topt=[Topt; Topt(end)+dt];                      % Store time
    % A switch to the next buffer, a setup takes place:
        Xopt=[Xopt; Xopt(end,:)+LAMBDA*theta];          % The work that arrives during the setup.
        Topt=[Topt; Topt(end)+theta];                   % Store time
    end
    % Update Distopt, Diffopt and Ratioopt:
    Distopt=[Distopt; norm(Xopt(end,:)-Xideal0)];
    Diffopt=[Diffopt; norm(Xopt(1,:)-Xideal0)-norm(Xopt(end,:)-Xideal0)];
    Ratioopt=[Ratioopt; Diffopt(end)/Topt(end)];
end

% ------------------------------------------------------------
%                   Comparing
% ------------------------------------------------------------

fprintf('\n The optimization algorithm is %g',(Tclear(end)-Topt(end))/Tclear(end)*100)
fprintf(' procent faster than clearing. \n')
fprintf(' The optimization algorithm lays %g',(Distclear(end)-Distopt(end))/Distclear(end)*100)
fprintf(' procent closer to the ideal Savkin trajectory than clearing. \n')
fprintf(' The optimization algorithm has a %g',(Ratioopt(end)-Ratioclear(end))/Ratioclear(end)*100)
fprintf(' procent better ratio improvement in time than clearing. \n')

% ------------------------------------------------------------
%                   Figures
% ------------------------------------------------------------
figure(1)
plot(Xideal(:,1),Xideal(:,2),'r',Xideal0(:,1),Xideal0(:,2),'r*',...
    Xclear(:,1),Xclear(:,2),'b',...
    Xopt(:,1),Xopt(:,2),'k')
grid
xlabel('X1')
ylabel('X2')
legend('Ideal Savkin trajectory','startpos. ideal Savkin trajectory','cyclic clearing policy','')
title('The trajectories of the buffers')

figure(2)
```

```
plot(Tclear(1:4:end),Distclear,'b.',...
     Topt(1:4:end),Distopt,'kp',...
     [0 Topt(end)],[epsilon epsilon],'y')
grid
xlabel('time')
ylabel('Distance to startpos. ideal Savkin trajectory')
title('Distance to startpos. ideal Savkin trajectory in time')
legend('cyclic clearing policy','Optimization')

figure(3)
plot(1:length(Distclear),Distclear,'b.',...
     1:length(Distopt),Distopt,'kp',...
     [0 length(Distclear)],[epsilon epsilon],'y')
grid
xlabel('cycle')
ylabel('Distance to startpos. ideal Savkin trajectory')
title('Distance to startpos. ideal Savkin trajectory per cycle')
legend('cyclic clearing policy','Optimization')
```

shortesttime.m

```
% function f = shortesttime(t,j,i,theta,MU,LAMBDA,X0,Xideal0,epsilon)
% computes the objective function. The objective function should be
% minimized.

function f = shortesttime(t,j,i,theta,MU,LAMBDA,X0,Xideal0,epsilon)
f = norm(t,1)+j*i*theta;    %Computes the total time
```

nonlconstraint.m

```
% function [c,ceq] = nonlconstraint(t,j,i,theta,MU,LAMBDA,X0,Xideal0,epsilon)
% computes the non-linear constraint. If c is smaller than zero, then
% the reached position is sufficiently close to the ideal Savkin
% trajectory.

function [c,ceq] = nonlconstraint(t,j,i,theta,MU,LAMBDA,X0,Xideal0,epsilon)
C= repmat(eye(j),i,1);
X=X0+LAMBDA*(norm(t,1)+i*j*theta)-t'*C.*MU;     % Computes the position
c = norm(Xideal0-X)-epsilon;                    % This should c <= 0.
ceq = 0;
```