Transient Behavior Analysis of Serial Production Lines

Student Thesis

submitted by

Pengfei Zheng
Matriculation Number: 4729268

Institut für Robotik und Prozessinformatik

1. Examiner: Prof. Dr. -Ing. Klaus Dröder
2. Examiner: Prof. Dr. Jochen Steil

Technische Universität Braunschweig

2020.04.30

Eidesstattliche Erklärung

Ich erkläre, dass ich die vorliegende wissenschaftliche Arbeit selbständig, sowie ohne unerlaubte fremde Hilfe verfasst und nicht anderweitig für Prüfungszwecke vorgelegt habe. Alle verwendeten Quellen und Hilfsmittel wurden angegeben, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet.

Braunschweig, den 01.01.2020

Pengfei Zheng

Acknowledgement

I would like to express my special thanks to Prof. Dr. Jochen Steil as well as my supervisor Tianran Wang for professional guidance and valuable support. They helped me in understanding and going deep into my topic in the early stages of the work, and later also gave me many valuable suggestions and inspirations when I was in difficulty. I am really grateful to them.

Abstract

In the past few years, due to the global energy crisis and rapid expansion of production requirement improving productivity and reducing energy consumption have received extensively focus. Therefore, manufacturing system which aims to improve operation control efficiency and results in energy sparing, has been widely studied. Steady state analysis of prodution systems has obtained broadly investigated. On the contrary, the trasient performance remained largely unexplored. This research mainly focus on system modeling, transient performance evaluation, and production line parameter optimization. Indeed, transient behavior of production systems has generally practical and theoretical implications.

The main contribution of this work is to implement the mathematical models in a high-level programming language, python, and make a result analysis. Furthermore, we also designed a experiment to find a optimized buffer size to improve the transient performance of the production lines.

Keywords: Geomatric Machine, Production Lines, mathmaticall model, performance evaluation, transient analysis.

Contents

List of Figures

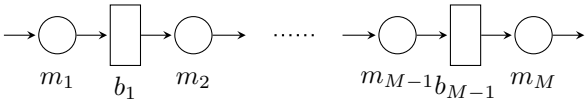
List of Tables

1 Introduction

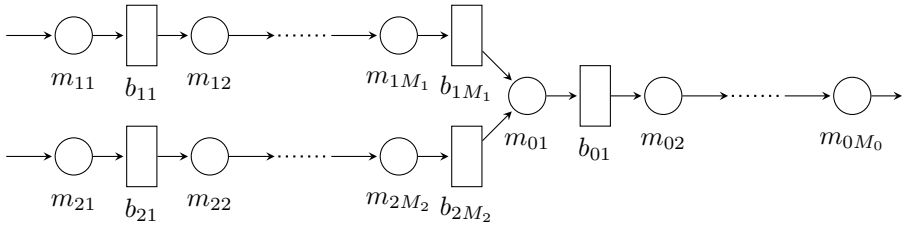
Production system has been studied widely during the last 65 years [1]. A production system is an industrial system that describe a procedure to transform from different resources into useful products. In this process, producing units (human operators, industrial robots, cells, etc.) and resource handling devices (shelves, carts, holders, vehicles, etc.) connected with each others so that desired products can be produced. It is a very important part of manufacturing research and application.

Extensive research has been invested in develpoing for design, modeling, improvement, analysis and control of production systems (for instance, monographs [2–5]). Despite the fact that in practice production systems may take several kinds of physical topologies, serial lines [see Figure 1.1(a)] and assembly systems [see Figure 1.1(b)] are the two most basic structures used in different manufacturing environments. In the literature, meanwhile, while assembly systems have been extensively investigated, serial production lines are been paid much less attention. Early research on assembly systems only considered the cases of multi-sequence-single-server, where different types of parts arrive at a single server in oder to be assembled together [4, 6]. Inspired by these works, few three-server system with limited sequence capacities have been studied [7, 8]. In these papers, dual servers represent component parts production, while the other server describes assembly operations. In addition, assembly systems based on queueing model have been further explored in papers [5, 9, 10] and the references within . The problem of steady-state performance evaluation of assembly system with unreliable machines and limited buffers has been studied in works [11–16] . In particular, the literature [11] developed a deconstruction technique to approximate the steady-state throughput for assembly systems based on machines with geometric models and identical processing times, while the article [12] focuses on assembly systems with geometric machines and a nonidentical processing time through turning the assembly system to a serial line. Furthermore, paper [13] develops the analysis to assembly system by geometric processing times.

The steady state behavior of production systems has been deeply studied in the last few years [15, 16]. Although it is often difficult to declare from the partial view that a production system is in steady state, the steady-state analysis approach is effective and accurate enough for manufacturing systems with large production capacity. The large production capacity allows the system to decay instantaneously to a negligible time compared to the global production run-time, and, therefore, allows it to use steady-state methods. Unfortunately, in addition to large-capacity manufacturing, there are a large number of mid- and small-capacity manufacturing systems in practice, which usually operate in a different method. In some these cases, one production line is usually able to produce several end products but can only produce one type of end product, equipment or special product at a time because of process. This usually lead to small- to medium-size production run-based operation based on the customer's order, where a production run contains only a specific amount of particular type of product. Obviously, when the



(a) Serial line.



(b) Assymbly system.

Figure 1.1: two production system

size of the production run is relatively small, the steady-state approaches cannot provide an ideal and accurate analysis of system. In some industries, a production run somtimes means a batch.

Nevertheless, the transient period of the behavior in production system has received much less research attention because of its complexity and the still large numbers of unsolved problems in steady state production systems research. On the other hand, recent research [17] has proved that transient analysis has become one of the most important fields in production systems research. Indeed, transient behavior of production systems have not been systematically studied and it is considered as one of the most significant directions in production systems research [17]. Specially, transient properties of serial production systems with two machines having the Bernoulli reliability model have been explored in [18–20] based in Markovian analysis. The research was later also extended to the case of multi-machine Bernoulli lines in [21, 22], which set up computationally efficient algorithms with recursive aggregation to approximate the transient performance with high accuracy.

Applications of Bernoulli line transient properties analysis is reported in [22–24]. Specailly, the paper [22] extended the algorithm developed in work [21] to the Bernoulli series production line with time-varying machine parameters. In spite of important results have been obtained regarding the transient behabior of the production system, it should be considered that most of the analysis studies cited above are only applicable to systems with machines based on Bernoulli reliability model, which can only be applied in the situations where the average machine downtime is comparable to its cycle time. Although the paper [25] tried to study the transient properties of serial production lines with machines in the geometric reliability model, the results were only applicable to the case of two-machine lines with an initial buffer occupancy at the beginning. For two-machine production lines with general initial state and longer lines, as far as we know, no analytical methods for have been built up for analysis of their transient performance because of larger dimension of the system state. Thus, the goal of this paper is to set up mathmaticall models in python code with the help of the Bernoulli reliabile geomatric machine serial

production line. Then, compare the results and consider the causes of the differences. In addition, an analysis of the relationship between the parameter improvement and trasient performance is evaluated.

The remainder of the papaer is organized as follows: Section II introduces the assumptions for the system and presents the performance measures of interest, and theory of Markov chain. mathmaticall modeling and behavior performance evaluation of individual machines, two-machine lines. and multi-machine lines are described in Section III. Nevertheless, the optimazition of a serial production line is given in Section IV. The conclusions and future work are given in Section V.

2 Model and Performance Measures

This chapter mainly focus on the mathmaticall model, the first part introduces the theoretical basis of the mathmaticall mode, and the assumptions to build such mathmaticall model, the second part presents the performance measures of transient properties.

2.1 Model

Before the discussion of the mathmaticall mode, the classic conception of Markov chain should be first introduced as the theoretical basis of stochastic models to describe a process of real world. This made the foundation of for general stochastic simulation methods known such as Markov chain Mente Carlo, which are suitable for simulating sampling from comlex probability distributions, and also able to be applied in Bayesian statistics and artificial intelligence. Besides, for serial production lines with geometric machines and finite buffers, in particular, there are more assumptions to describe and derive such a system.

2.1.1 Markov chain

A Markov chain is a stochastic model basically describing a sequence of random events in which the probability of each event is related only to the state attained in the previous event [26]. In time-continuous situation, it is known as Markov process. It is named after the Russian mathematician Andrey Markov.

Here is a diagram (see Figure 2.1) representing a two state Markov process, where the states are labelled as A and B respectively. Every number represents the probability of the Markov process transforming from one state to another state with the arrow that indicates the direction. For instance, if the Markov process is now in state B, then the probability it transforms to state A is 0.6, while the probability it remains in state B is 0.4.

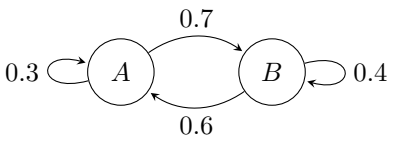


Figure 2.1: A two-state Markov process

The precise definition of the conception "Markov chain" will be given as following. A Markov process is a certain type of stochastic process distiguished by the Markov property [27]. A Markov chain is a Markov process with a calculable (namely, finit or denumerably infinite) number of states. The time parameters can be taken as the set of nonnegative integers or the set of nonnegative real numbers, so we have discrete parameter cases or continuous parameter cases. The adjective "simple" is sometimes used to qualify

the Markov chain, but since we really do not discuss the "multiple" chains we should not treat it differently. In addition, we should only discuss the Markov chains which has a " stationary (or temporally homogeneous) transition probabilities" so that the qualifying expression in quotes will be understood. In the end, our discussion does not distinguish between a finite or a calculable infinite number of states and therefore we does not do any special treatment for the former case.

In this part we handle the case of a discrete parameter. Here the essential foundations can be summarized as follows.

Suppose we have an abstract set $\mathbf{\Omega}$, called the probability space, having the generic element ω, called elementary event; a Borel field $\mathscr{F}$ of subsets of $\mathbf{\Omega}$, called measurable sets or events, where $\mathbf{\Omega}$ is considered as a member; and a probability measure $\boldsymbol{P}$ defined on $\mathscr{F}$. The triple $(\mathbf{\Omega}, \mathscr{F}, \boldsymbol{P})$ is named as a probability triple. A set in $\mathscr{F}$ of probability zero will be named as a null set; "almost all ω (a.a.ω)" or "almost everywhere (a.e.)" expresses "all ω except a null set". The pair $(\mathscr{F}, \boldsymbol{P})$ will be regarded to be complete in the sense that each subset of a null set is attached to $\mathscr{F}$ and is a null set. If and only if a Borel subfield of $\mathscr{F}$ contains all null sets, it is said to be augmented. If a Borel subfield is given, there exists a unique smallest augmented Borel subfield of $\mathscr{F}$ including the given one. Unless exceptions specified all the following Borel fields will be presumed to be augmented. A (real) random variable is regarded as a single-valued function from a set Δ_0 in $\mathscr{F}$ to the closed real line $X = [-\infty, +\infty]$ thus for every real number c the set of ω in Δ_0 for which $x(\omega) \leq c$ is attached to $\mathscr{F}$. Δ_0 is named as the domain of definition of x and the set Δ of ω in Δ_0 for which $|x(\omega)| < \infty$ is named as the domain of finiteness of x. If neither domain is specific it will be figured out that $\boldsymbol{P}(\Delta) = 1$; otherwise stated, the random variable is finite-valued. It comes from the definition that if A is any Borel set in X, then the set of ω for which $x(\omega) \in A$, to be symbolized as $\omega : x(\omega) \in A$ is attached to $\mathscr{F}$. The probability of the set comes to be denoted by

$$\boldsymbol{P}\{x(\omega) \in A\}. \tag{2.1}$$

Symbols alike to these which will come later should be self-explanatory if we mark that commas or semicolons are served as symbolizing intersection of sets; for instance

$$\begin{aligned} \boldsymbol{P}\{x_\nu(\omega) \leq c_\nu, 1 \leq \nu \leq 2\} &= \boldsymbol{P}\{x_1(\omega) \leq c_1; x_2(\omega) \leq c_2\} \\ &= \boldsymbol{P}\left\{\cap_{n=1}^{2}[x_n(\omega) \leq c_n]\right\}. \end{aligned} \tag{2.2}$$

Suppose we hold a set $\{x_s, s \in S\}$ of random variables, the Borel field created by them is the smallest augmented Borel field regarding which all the random variables in the set are able to be measured. Special cases are $\mathscr{F}\{x_s, s \circ t\}$ where $\circ$ is one of the four symbols $<, \leq, >, \geq$.

The function F described by

$$F(u) = \boldsymbol{P}\{x(\omega) \leq u\} \tag{2.3}$$

for all real u is named as the distribution function of x. We get such that

$$\lim_{u \to +\infty} F(u) - \lim_{u \to -\infty} F(u) = \boldsymbol{P}(\Delta). \tag{2.4}$$

Unless otherwise stated, we will use a distribution function or a probability distribution that of a random variable which is limited with probability one to represent. Thus $\lim_{u\to-\infty} F(u) = 0$, $\lim_{u\to+\infty} F(u) = 1$. A random variable x is discrete if and only if there exists a calculable set A thus $P\{x(\omega) \in A\} = 1$. A possible value c of x is just one like $\boldsymbol{P}\{x(\omega) = c\} > 0$. All random variables engaged in the part are discrete. A function x from Δ_0 in $\mathscr{F}$ to a calculable set A can be taken as a random variable if and only if the set $\{\omega : x(\omega) = c\}$ is attached to $\mathscr{F}$ for every c in A. Actually A could be assigned as any abstract calculable set and we could define an abstract-valued random variable like this.

Suppose Λ_1 and Λ_2 are two sets in $\mathscr{F}$, the conditional probability of Λ_2 in relation to Λ_1 is defined By

$$\boldsymbol{P}(\Lambda_2|\Lambda_1) = \frac{\boldsymbol{P}(\Lambda_1\Lambda_2)}{\boldsymbol{P}(\Lambda_1)} \tag{2.5}$$

given that $\boldsymbol{P}(\Lambda_1) > 0$. When $\boldsymbol{P}(\Lambda_1) = 0$, $\boldsymbol{P}(\Lambda_2|\Lambda_1)$ is undefined. For convenience, undefined conditional probabilities will often be seen in following. If one that comes our from them is multiplied by a variable which is equal to 0, the product is considered to be 0. The conditional probability of the set $\{\omega : x_3(\omega) = c_3\}$ in relation to the set $\cap_{n-1}^{2}\{\omega : x_n(\omega) = c_n\}$ for example is symbolized as

$$\boldsymbol{P}\{x_3(\omega) = c_3|x_1(\omega) = c_1, x_2(\omega) = c_2\}. \tag{2.6}$$

For the random variables $\{x_\nu, 1 \le \nu \le n\}$, it is not necessarily finite-valued, and is said to be independent in case

$$\boldsymbol{P}\left\{\bigcap_{\nu=1}^{n}[x_\nu(\omega) \le c_\nu]\right\} = \prod_{\nu=1}^{n}\boldsymbol{P}\{x_\nu(\omega) \le c_\nu\} \tag{2.7}$$

for any real finite c_ν, $1 \le \nu \le n$. It comes out that the same equation remains true if the sets $\{\omega : x_\nu(\omega) \le c_\nu\}$ are changed to the more general sets $\omega : x_\nu(\omega) \in A_\nu$ where the A_ν are Borel sets in X. Suppose we habe a sequence $\{x_n, n \ge 1\}$, it is a sequence of independent random variables only if any finite number of them are independent. The denumerable sets Λ_ν, $1 \le \nu \le n$ or $1 \le \nu < \infty$ are independent only if their indicators are, the indicator of a set rendered as the function which on the set end is equal to one zero elsewhere.

The mathmaticall expectation of a random variable x can be give by the abstract Lebesgue-Stieltjes integral

$$\boldsymbol{E}(x) = \int_{\boldsymbol{\Omega}} x(\omega)\boldsymbol{P}(d\omega). \tag{2.8}$$

In general, we extend this definition to a random variable that supposes that only if the integral existing, one of two values $+\infty$ or $-\infty$ with positive probability is finite or infinite. The conditional expectation of x relative to Λ, provided $\Lambda \in \mathscr{F}$, is defined as

$$\boldsymbol{E}(x|\Lambda) = \int_{\boldsymbol{\Omega}} x(\omega)\boldsymbol{P}(d\omega|\Lambda) = \frac{\int_{\boldsymbol{\Omega}} x(\omega)\boldsymbol{P}(d\omega)}{\boldsymbol{P}(\Lambda)}. \tag{2.9}$$

Specially if x is discrete with all its probable quantities in the measurable set A then

$$\boldsymbol{E}(x|\Lambda) = \frac{1}{\boldsymbol{P}} \sum_{i \in A} i\boldsymbol{P}\{\Lambda; x(\omega) = i\} \tag{2.10}$$

as long as the series is fully converged.

In this part the letters n, m, ν, r, s, t denote non-negative integers unless otherwise stated.

A discrete parameter stochastic process is described as a sequence of random variables $\{x_n, n \geq 0\}$ defined regarding a probability triple $(\boldsymbol{\Omega}, \mathscr{F}, \boldsymbol{P})$. If all random variables are discrete, the union $\boldsymbol{I}$ of all probable quantities of all x_n is a countable set named as the minimum state space of the process and every element of $\boldsymbol{I}$ is a state. Therefore, $i \in \boldsymbol{I}$ if and only if there is an $n \geq O$ so as $\boldsymbol{P}\{x_n(\omega) = i\} > O$. We take from the language of physics where the term "state" indicates that of a material system whose evolution in time is characterized by our stochastic process model.

A discrete parameter Markov chain can be defined by a sequence of discrete random variables $\{x_n, n \geq O\}$ possessing the under property: for arbitrary $n \geq 2, O \geq t_1 < ... < t_n$ and arbitrary $i_1, ..., i_n$ in the state space $\boldsymbol{I}$ we have

$$\begin{cases} \boldsymbol{P}\{x_{t_n}(\omega) = i_n | x_{t_1}(\omega) = i_1, ..., x_{t_{n-1}}(\omega) = i_{n-1}\} \\ \qquad = \boldsymbol{P}\{x_{t_n}(\omega) = i_n | x_{t_{n-1}}(\omega) = i_{n-1}\}. \end{cases} \tag{2.11}$$

whenever the left member is defined. The condition 2.11 will appear as the Markov property. It is identical to the under apparently weaker condition: for each $n \geq 1$,

$$\begin{aligned} &\boldsymbol{P}\{x_n(\omega) = i_n | x_0(\omega) = i_0, ..., x_{n-1}(\omega) = i_{n-1}\} \\ &\qquad = \boldsymbol{P}\{x_n(\omega) = i_n | x_{n-1}(\omega) = i_{n-1}\}. \end{aligned} \tag{2.12}$$

The proof here is neglected. An important consequence comes from condition 2.11 is that for arbitrary $n \geq 0$ and $0 \leq t_1 < ... < t_n ... < t_{n+m}$; and arbitrary $i_1, ..., i_n, ...mi_{n+m}$ in $\boldsymbol{I}$ we get

$$\begin{cases} \boldsymbol{P}\{x_{t_\nu}(\omega) = i_\nu, n \leq \nu \leq n+m | x_{t_\nu}(\omega) = i_\nu, 1 \leq \nu \leq n-1\} \\ = \boldsymbol{P}\{x_{t_\nu}(\omega) = i_\nu, n \leq \nu \leq n+m | x_{t_{n-1}}(\omega) = i_{n-1}\}. \end{cases} \tag{2.13}$$

The proof of 2.13 is carried out by induction on m. For $m = 0$, it decreases to 2.11. Supposing that 2.13 is true for a certain quantity of m, it comes to use the rules of combining conditional probabilities and 2.11,

$$\begin{aligned} &\boldsymbol{P}\{x_{t_\nu}(\omega) = i_\nu, n \leq \nu \leq n+m+1 | x_{t_\nu}(\omega) = i_\nu, 1 \leq \nu \leq n-1\} \\ =&\boldsymbol{P}\{x_{t_\nu}(\omega) = i_\nu, n \leq \nu \leq n+m | x_{t_\nu}(\omega) = i_\nu, 1 \leq \nu \leq n-1\} \times \\ \times&\boldsymbol{P}\{x_{t_{n+m+1}}(\omega) = i_{n+m+1} | x_{t_\nu}(\omega) = i_\nu, 1 \leq \nu \leq n+m\} \\ =&\boldsymbol{P}\{x_{t_\nu}(\omega) = i_\nu, n \leq \nu \leq n+m | x_{t_{n-1}}(\omega) = i_{n-1}\} \times \\ \times&\boldsymbol{P}\{x_{t_{n+m+1}}(\omega) = i_{n+m+1} | x_{t_{n+m}}(\omega) = i_{n+m}\} \\ =&\boldsymbol{P}\{x_{t_\nu}(\omega) = i_\nu, n \leq \nu \leq n+m | x_{t_{n-1}}(\omega) = i_{n-1}\} \times \\ \times&\boldsymbol{P}\{x_{t_{n+m+1}}(\omega) = i_{n+m+1} | x_{t_\nu}(\omega) = i_\nu, n-1 \leq \nu \leq n+m\} \\ =&\boldsymbol{P}\{x_{t_\nu}(\omega) = i_\nu, n \leq \nu \leq n+m+1 | x_{t_{n-1}}(\omega) = i_{n-1}\}. \end{aligned} \tag{2.14}$$

Therefore 2.13 is true when m is replaced by $m+1$ and the induction is complete.

It is well know that a result comes from measure theory asserts that for all m the validity of 2.13 indicates that of the more general result: for each $\boldsymbol{M} \in \mathscr{F}\{x_t, t \geq t_n\}$ we get

$$\boldsymbol{P}\{\boldsymbol{M}|x_{t_\nu}(\omega) = i_\nu, 1 \leq \nu \leq n\} = \boldsymbol{P}\{\boldsymbol{M}|x_{t_n}(\omega) = i_n\}. \tag{2.15}$$

In this expression Markov property can be expressed verbally as positing that "If we know the state of an event at the previous moments, the probability of it is the same as only the last given state". More vaguely, it can be presented that "the past state is completely independent from the future state". Misleading by these description, it is a normal mistake to believe that Equation 2.15 stays true if the conditional events in it are replaced by a more general event, for instance, replacing i_ν with A_ν, where A_ν is a subset of $\boldsymbol{I}$. No doubt the resulting equation will generally be wrong. Nevertheless, the under extension of 2.15 is correct and trivial: if $\Lambda \in \mathscr{F}\{x_t, t \leq t_n\}$

$$\boldsymbol{P}\{\boldsymbol{M}|\Lambda; x_{t_n}(\omega) = i\} = \boldsymbol{P}\{\boldsymbol{M}|x_{t_n}(\omega) = i\}. \tag{2.16}$$

2.1.2 Descriptive Model

Consider a production line shown in Figure 2.2, in which circles illustrate machines and rectangles illustrate buffers. The system will be defined by the under assumptions.

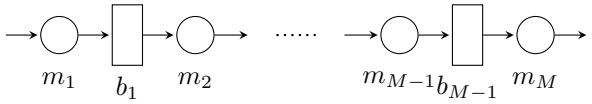


Figure 2.2: Serial production line.

.

1. The system is composed of M machines, arranged in series, and $M-1$ buffers between each consecutive pair of machines.

2. The machines all have the same constant cycle time τ. The time axis is limited in a slot duration τ. Machines start to operate at the beginning of each time slot.

3. The model requires the machine to observe geometric reliability: We use $s_i(n) \in \{0 = down, 1 = up\}$ to symbolize the state of machine m_i during time slot $n, i = 1, ..., M$. Then, we can fomulate the transient probabilities as follow:

$$\begin{aligned} &\text{Prob}[s_i(n+1) = 0|s_i(n) = 1] = P_i \\ &\text{Prob}[s_i(n+1) = 1|s_i(n) = 1] = 1 - P_i \\ &\text{Prob}[s_i(n+1) = 1|s_i(n) = 0] = R_i \\ &\text{Prob}[s_i(n+1) = 0|s_i(n) = 0] = 1 - R_i \end{aligned} \tag{2.17}$$

   where P_i and R_i are used to denote the breakdown and repair probabilities separately. Each machine has the operation independence from one another.

4. Each buffer is described by its capacity (in other words, the maximum number of products the buffer can preserve), $1 \leq N_i < \infty$, $i = 1, ..., M-1$.

5. Machine $m_i, i = 2, ..., M$, is in a state of starved condition in one time slot if it is up (i.e. it works properly) and buffer b_{i-1} is empty at the start of this time slot. Machine m_1 will be never starved for there will always be raw material.
6. Machine $m_i, i = 1, ..., M-1$, is in a blocked state in one time slot if it works properly, buffer b_i has N_i parts at the start of this time slot, and machine m_{i+1} is to take a product in this time slot. Machine m_M will never be blocked.
7. If a machine works well and is neither starved or blocked, it is capable to process one product during a time slot (in other words, takes a product from its upstream buffer at the start of this time slot, processes it within the time slot, and puts it into its downstream buffer at the end of the time slot); otherwise, no processing will occur for the machine in this time slot.
8. The system works in a total of T time slots.

Remark 1: For the assumption 3, the up and downtime of machine m_i are geometric random values. We can calculate its average up- and downtime through $T_{up,i} = 1/P_i$ and $T_{down,i} = 1/R_i$, separately. Moreover, the machine efficiency, in other words, the probability (ratio of time) that m_i works properly in steady state, can be calculated by $e_i = T_{up,i}/(T_{up,i} + T_{down,i}) = R_i/(R_i + P_i)$.

Remark 2: When the average downtime of the machine is much longer than its cycle time (for instance, in machining, painting, wielding operations)the geometric reliability is generally applicable. Steady state performance of the geometric serial lines has been explored in numerous publications in manufacturing systems literatures [1, 2, 28, 29] The geometric model also has a successful application in industrial occasions works (i.e. [30, 31]).

Remark 3: These previous assumptions indicate that the failures of machines are time-dependent (in other words, in a state of starvation or blockage a machine may still break down). Another kind of failure model as an optional choice, operation-dependent failure (in other words, in the state of starvation or blockage, a machine will not break down), is also covered in the literature. The properties and behavior of systems established by both declarations are very approximate (by [32]). In this work we consider the former one, time-dependent failures.

Remark 4: Assumption 6. indicates the blocked-before-service (BBS) regulation, under which, a machine may be in a starvation and blockage state in the same time slot. Its similitude, the blocked-after-service (BAS) regulation, is also extensively used in production systems research (see [17, 33, 34]). The analysis of systems established by both regulations are approximate. In fact, either system models under one regulation can be transformed into the another one with the other regulation by changing the buffer size by one unit. In this work we use BBS regulation, because it results in a simpler statement [35].

2.2 Performance Measures

In order to conduct strict control of production systems and real-time evaluation, transient performance measures must be introduced. In the framework of geometric serial lines given by assumptions 1-8, the performance measures of interest are:

- *Production Rate*, $PR(n)$ = the expected number of finished parts produced by m_M in time slot n;
- *Consumption Rate*, $CR(n)$ = the expected number of raw parts consumed by m_1 in time slot n;
- *Work-in-process*, $WIP_i(n)$ = the expected number of parts in buffer b_i at the start of time slot n, $i = 1, ..., M-1$;
- *Machine Starvation*, $ST_i(n) = \text{Prob}[m_i \text{ is starved by } b_{i-1} \text{ in time slot } n]$, $i = 2, ..., M$;
- *Machine Blockage*,$BL_i(n) = \text{Prob}[m_i \text{ is blocked by } b_i \text{ in time slot } n]$, $i = 1, ..., M-1$.

In this paper, we take average of numours simulation results to get these performance measures of interest. It should be noted that, in this paper, a simulation procedure uses least 10000 the production lines to get the mathematical expectations.

3 Transient Performance of Various Geomatric Machines

In this chapter we will use the programming language to implement these mathmaticall models from the simple individual machine to multi-machine and attempt to get the transient performance. In the second section, we will try to analysis the difference between this work and another one.

We choose to use python as the implementation language because python is a interpreted, general-purpose and object-oriented programming language. With its extensive mathmatics library and other third-party library, it is very popular to be used as a scientific scripting language to aid in a numerical data processing and manipulation problem like this.

3.1 Individual Geomatric Machine

Although transient analysis of individual geometric machines with constant parameters has been studied in [25], as the basis of all the following models, we still take it as the primary work. Since the performance evaluation method which we try to derive is the foundation of the study in the this paper, we briefly introduce it below.

3.1.1 Mathematical Derivation of Individual Machine

Since derivation of the performance evaluation method is the foundamental work of the study in this work, we first view it as following.

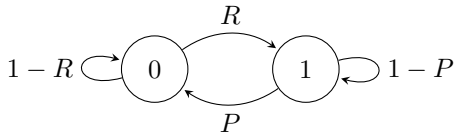


Figure 3.1: State transition diagram of one geometric machine

The state transition diagram for an individual geometric machine is illustrated in Figure 3.1. We use $x_i(n), i \in \{0 = down, 1 = up\}$ to indicate the probability that the machine is in state i during time slot n, that is $x_i(n) = Prob[s(n) = i]$. Apparently, the system is described by a two-state ergodic Markov chain and the transformation of state vector $x(n) = [x_0(n) \quad x_1(n)]^T$ can be presented by

$$x(n+1) = A_1 x(n), x_0(n) + x_1(n) = 1 \tag{3.1}$$

where

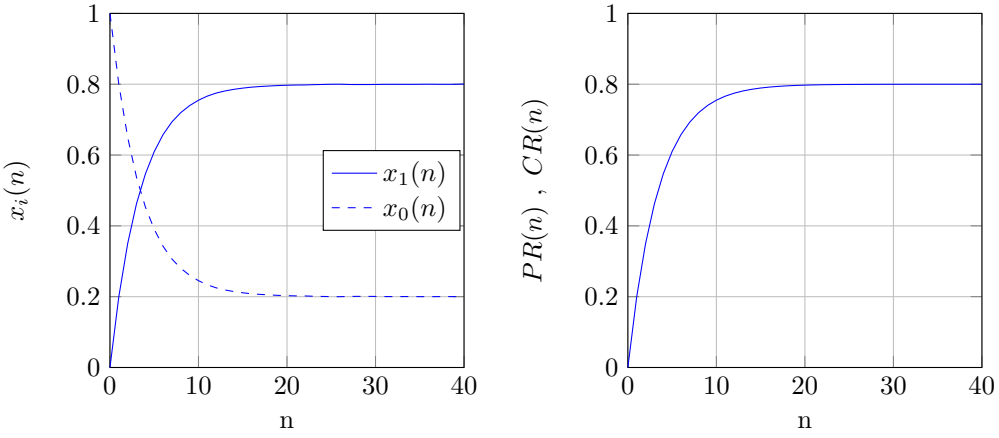
$$A_1 = \begin{bmatrix} 1-R & P \\ R & 1-P \end{bmatrix} \tag{3.2}$$

The production rate and consumption rate of an individual machine with the original state both down(0) and up(1) can be evaluated as follow

$$PR(n) = CR(n) = x_1(n) = \begin{bmatrix} 0 & 1 \end{bmatrix} x(n) = \begin{bmatrix} 0 & 1 \end{bmatrix} A_1^n x(0) \tag{3.3}$$

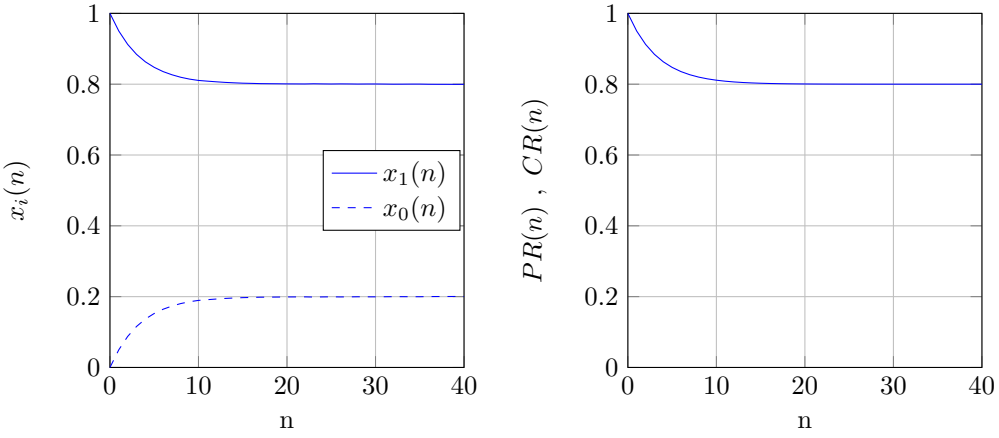
which is separately linear in machine state $x(n)$.

As figured out in the diagram, suppose a geometric machine given with breakdown probability $P = 0.05$ and repair probability $R = 0.2$. The performance measures and the transformation of the system state can be illustrated in Figure 3.2 and 3.3, assuming the machine is in a beginning state of down and up, separately. As all implied, the initial condition of a machine strongly influence on system state transients, which may lead to production loss (see Figure 3.2) or production gain (see Figure 3.3).



(a) Result of simulation (b) Result of calculation

Figure 3.2: Transients of an individual geomatric machine when it is initially down



(a) Result of simulation (b) Result of calculation

Figure 3.3: Transients of an individual geomatric machine when it is initially up

3.1.2 Implementation of Individual Machine

In the python, we use the object-oriented features to help build the model. In order to illustrate the structure, we use a xml diagram (see Figure 3.4). We separate the codes into two parts. The first part is a class file called Individual. A **`class Individual`** represents a geomatirc machine that runs in a two-state Markov chain. It holds the parameters, which are transformed from another file, and calculates once a time slot till the end of the time control parameter **`n`** changes to zero.

Another file, which is used to call the "simu1.py", are also attached in Appendix. The main purpose of this file is to calculate the the average values of all the evaluation performance in order to get the mathmaticall expectation. The final daten are collected in the file called **`result.txt`**.

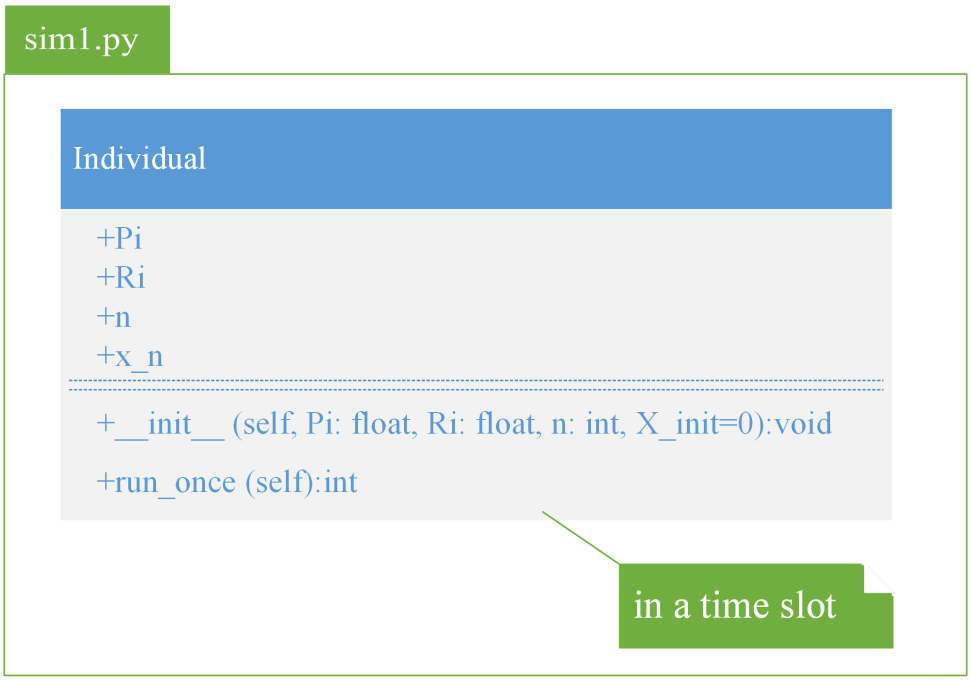


Figure 3.4: UML diagram of an individual machine model

We also use a flow chart to describe the procedure of an individual machine model (see Figure 3.5). The fuction **`run_once()`** detect whether the machine is up or down, and according its situation give another judgement if it should be repaired/broken or not. Furthermore, it will ended up with this time slot and start another one.

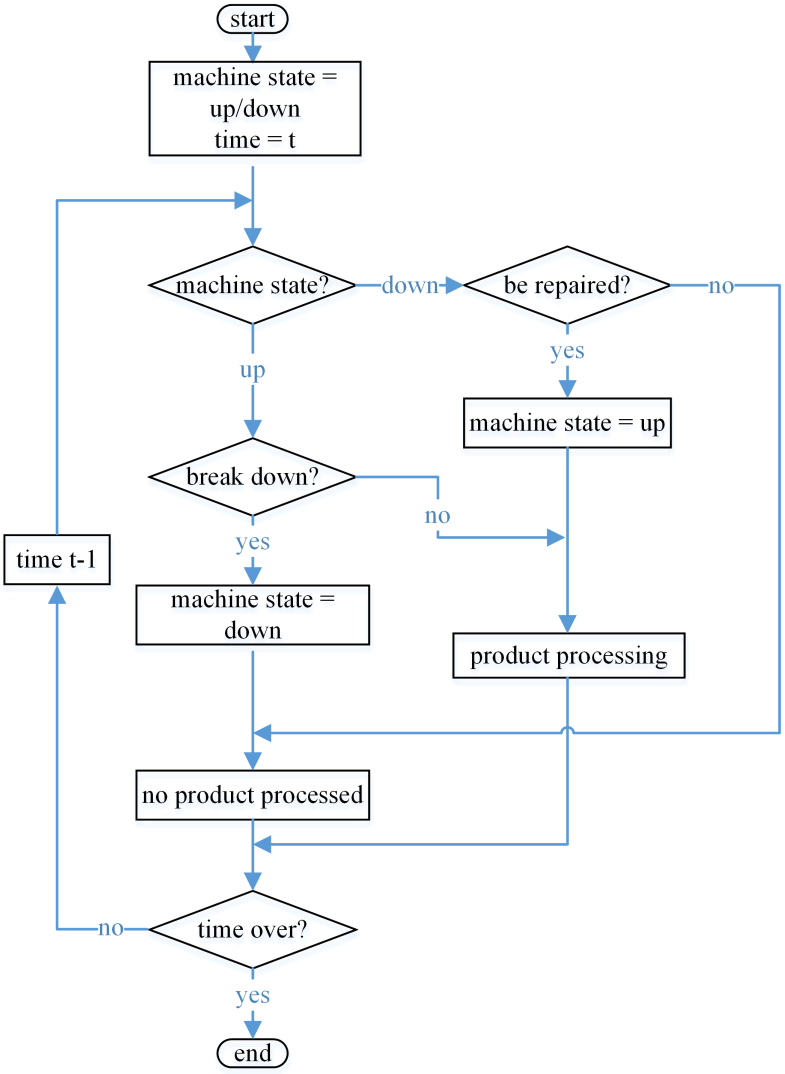


Figure 3.5: Flow chart of an individual machine model

3.2 Transient Performance of Two-Machine Geometric Lines

In this Section we also use two part to introduce the two-machine model. It is the most significant difference between two-machine model and individual machine model that there is a buffer between two mahines. This will lead to a performance improvement and other complicated situation.

3.2.1 Mathematical Dirivation of two-machine Model

Suppose that we have a two-machine geometric line restrained by assumptions 1-8. shown in Figure 3.6. From the figure we can easily tell that the system obersve an ergodic Markov chain. Moreover, to machine state $s_i(n)$, we use $j(n)$ to imply the number of products in the buffer at the start of time slot n. Then, the state of the Markov chain can be described

by a triple $(j(n), s_1(n), s_2(n))$, where $j(n) \in 0, 1, ..., N$ and $s_1(n), s_2(n) \in 0, 1$. Apparently, the system will in total have states of $4(N+1)$.In order derive the transition probabilities from these states, we organize the states in the following form: we use $r(j, s1, s2)$ indicate the state number of the Markov chain state (j, s_1, s_2), $h \in 0, 1, ..., N, s_1, s_2 \in 0, 1$. Given

$$r(j, s_1, s_2) = 4j + 2s_1 + s_2 + 1. \tag{3.4}$$

Then, the sequence of the $4(N+1)$ system states can be described in Table 3.1. Moreover,

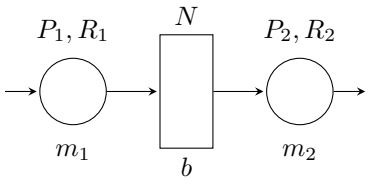


Figure 3.6: State transition diagram of two geometric machine with buffer

we use a unique number from 1 to $4(N+1)$ to represent each system state. For instance, State 1 represents that both machines are broken down and the buffer is empty, while State $4(N+1)$ represents that both machines are working properly and the buffer is full. Furthermore, suppose a state number r, its related buffer and machines states can be described as follows:

$$\begin{aligned} j^{(r)} &= \left\lfloor \frac{r-1}{4} \right\rfloor, s_1^{(r)} = \left\lfloor \frac{r-1-4j^{(r)}}{2} \right\rfloor \\ s_2^{(r)} &= \left\lfloor \frac{r-1-4j^{(r)-2s_1^{(r)}}}{1} \right\rfloor \end{aligned} \tag{3.5}$$

where $\lfloor a \rfloor$ refers to the largest integer not greater than a.

Table 3.1: Arrangement of the System States $k = 0, 1, ..., N$

State number(r)	$4k+1$	$4k+2$	$4k+3$	$4k+4$
j	k	k	k	k
s_1	0	0	1	1
s_2	0	1	0	1

Notice that the transition of $j(n)$ is deterministic defined by $s_1(n)$ and $s_2(n)$. For a given serial production lines, the equations that characterize the dynamics of $j(n)$ have been formulated in [21]

$$j(n+1) = j'(n) + s_1(n) min\{N - j'(n), 1\} \tag{3.6}$$

where

$$j'(n) = j(n) - s_2(n) min\{j(n), 1\} \tag{3.7}$$

In the above eqution, $j'(n)$ refers to the occupancy of the buffer as long as machine m_2 takes a part from the buffer at the start of time slot n.

The transitions of $s_i(n)$'s, meanwhile, are probabilistic based on 2.17. Thus, we can view each of the $4(N+1)$ states, then, coming from 3.6 and 3.7, distinguish all possible aimed states after one time slot by enumerating all four combining ways of $s_1(n)$ and $s_2(n)$, and, in the end, measure the corresponding transition probabilities using 2.17. Suppose that we use A_2 to characterize the transition probability matrix derived and use $x_i(n)$, $i \in 1, 2, ..., 4(N+1)$, describe the probability that the system, in other words, the Markov chain, is in state i in the period of the time slot n. Furthermore, the evolution of the system state, $x(n) = [x_1(n)x_2(n)...x_4(N+1)^{(n)}]^T$, is derived by

$$x(n+1) = A_2 x(n), \sum_{i=1}^{4(N+1)} x_i(n) = 1. \tag{3.8}$$

According the state arrangement 3.4, we have

$$\begin{aligned} x_{4j+1}(n) =& \text{Prob}[m_1 \text{ down, } m_2 \text{ down, buffer } b \text{ has } j \text{ parts at time } n] \\ x_{4j+2}(n) =& \text{Prob}[m_1 \text{ down, } m_2 \text{ up, buffer } b \text{ has } j \text{ parts at time } n] \\ x_{4j+3}(n) =& \text{Prob}[m_1 \text{ up, } m_2 \text{ down, buffer } b \text{ has } j \text{ parts at time } n] \\ x_{4j+4}(n) =& \text{Prob}[m_1 \text{ up, } m_2 \text{ up, buffer } b \text{ has } j \text{ parts at time } n]. \end{aligned} \tag{3.9}$$

Thus, according to the definitions derived in 2.2, we can compute the performance measures of the two-machine geometric line system in the following way:

$$\begin{aligned} PR(n) &= \text{Prob}[m_2 \text{ up,buffer } b \text{ not empty during time } n] \\ &= C_1 x(n) = [C_{1,0}\ C_{1,1}\ ...\ C_{1,N}]x(n) \\ CR(n) &= \text{Prob[m1 up and not blocked during time n]} \\ &= C_2 x(n) = [C_{2,0}\ C_{2,1}\ ...\ C_{2,N}]x(n) \\ WIP(n) &= \sum_{i=1}^{N} i \cdot \text{Prob}[\text{buffer } b \text{ has } i \text{ parts at time } n] \\ &= C_3 x(n) = [C_{3,0}\ C_{3,1}\ ...\ C_{3,N}]x(n) \\ ST_2 &= \text{Prob}[m_2 \text{ up and buffer } b \text{ empty at time } n] \\ &= C_4 x(n) = [C_{4,0}\ C_{4,1}\ ...\ C_{4,N}]x(n) \\ BL_1 &= \text{Prob}[m_1 \text{ up, } m_2 \text{ down,buffer } b \text{ is full at time } n] \\ &= C_5 x(n) = [C_{5,0}\ C_{5,1}\ ...\ C_{5,N}]x(n) \end{aligned} \tag{3.10}$$

where

$$\begin{aligned} C_{1,0} &= [0000], \quad C_{1,i} = [0101], \quad i = 1, ..., N \\ C_{2,N} &= [0001], \quad C_{2,i} = [0011], \quad i = 0, .., N-1 \\ C_{3,i} &= [iiii], \qquad i = 0, ..., N \\ C_{4,0} &= [0101], \quad C_{4,i} = [0000], \quad i = 1, ..., N \\ C_{5,N} &= [0010], \quad C_{5,i} = [0000], \quad i = 0, ..., N-1 \end{aligned} \tag{3.11}$$

In consequence, all these performance measures can be described as linear related in system state $x(n)$.

As an example, suppose a two-machine geometric line given by ssumptions 1-8. with machine and buffer parameters

$$P_1 = 0.03, R_1 = 0.18, P_2 = 0.06, R_2 = 0.21, N = 10.$$

Presume that both machines are initially broken and the buffer in the middle is initially empty. The Figure 3.7 presents transients of the performance measures of this system. For both machines are initially down and the buffer is empty, $PR(n)$,$CR(n)$ and $WIP(n)$ all rise from 0. Moreover, $PR(n)$ and $CR(n)$ have significant difference during the transient process, while with n growing larger, both converge to the same value because of the conservation of flow in steady state. In addition, for the buffer is empty in the beginning, $BL_1(n)$ stays zero (in other words, machine m_1 is not in blockage state) until $n > N$.

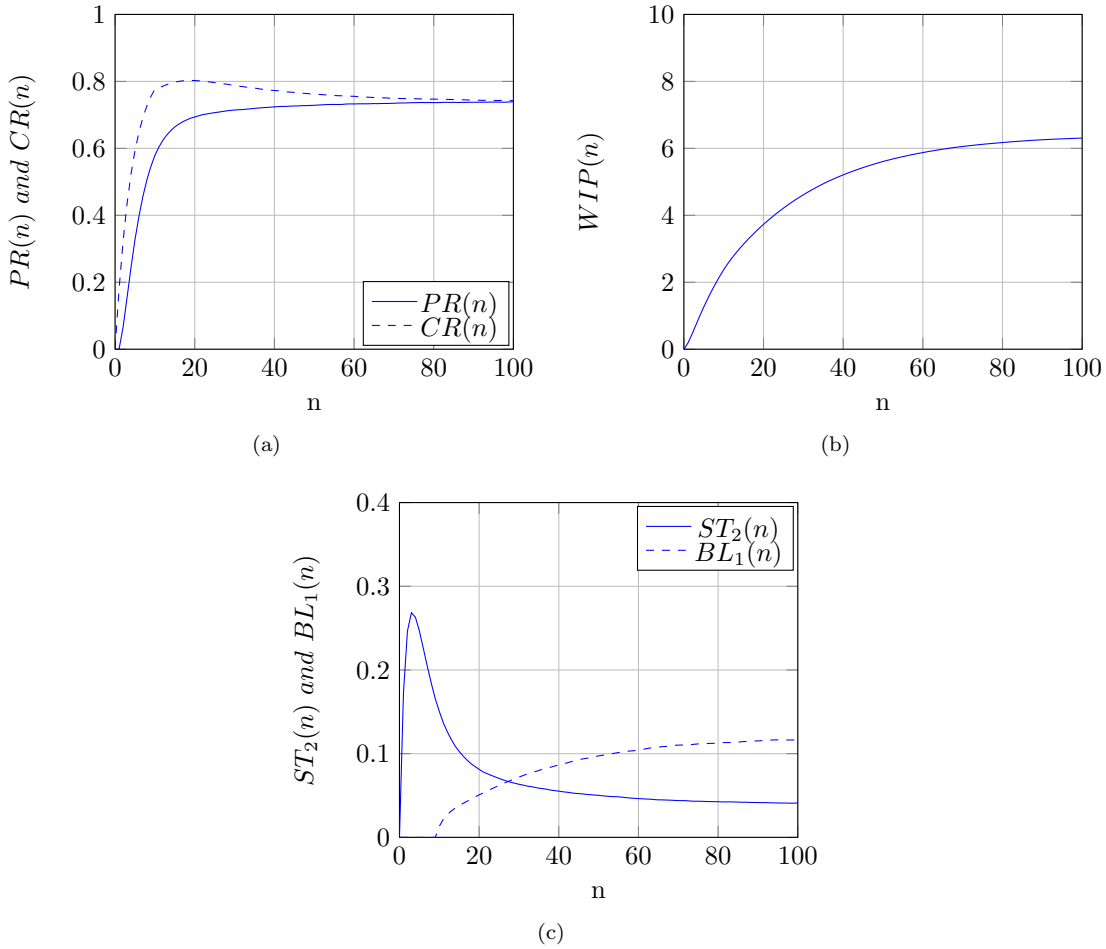


Figure 3.7: Transients of a two-machine geometric line. (a) $PR(n)$ *and* $CR(n)$;(b) $WIP(n)$; (c) $ST_2(n)$ *and* $BL_2(n)$

3.2.2 Implementation of two-machine model

In the python platform, we still keep the **class Individual** and have a slight adjust to simulate a machine instance. The main modification is to add two instance to represent the buffer upstream and downstream respectively. Moreover, it also hold a flag to judge whether the machine is blocked or starved according to the state of buffer in up- and downstream so that it cannot process a part in a time slot. Meanwhile, we add another **class Buffer** to represent the buffer in the middle of two machines. The buffer have its own storage to denote the numbers of products it hold and the maximal capacity. In addition, it also implements the products receiving and taking away function by two method **add_one()** and **take_one()**.

We use an UML diagram (Figure 3.8) to describe the relationship between three class. The program procedure is as follow: first, in the script "sim2.py" create we a certain amount of production lines, and each contains an instance from **class TwoMachine**. Meanwhile, the **class TwoMachine** consists of two instances from **class Individual** which represent the first and second machine respectively, and a buffer instance from **class Buffer**. In addition, the function **one_slot()** simulate a process in one time slot, and then give the critical transient performance measures back to script "sim2.py" in order that the data can be record in the file.

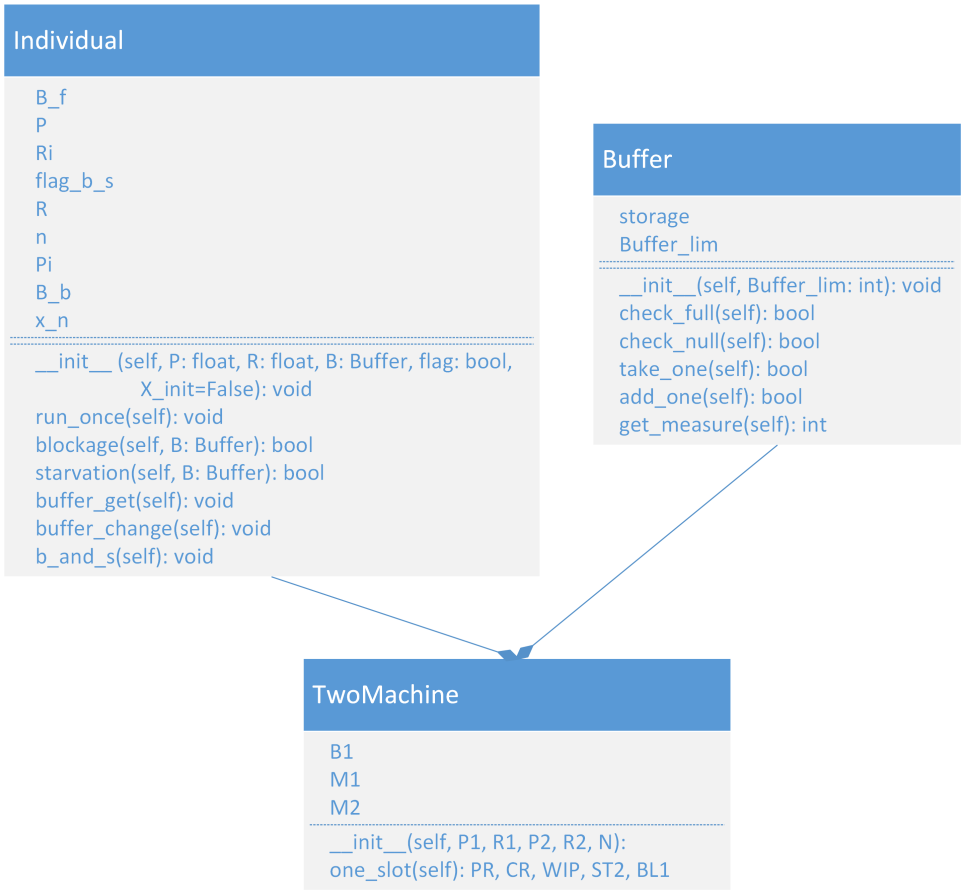


Figure 3.8: UML diagram of a two-machine model

3.3 Transient Performance of Multi Machine Lines

In this section we extend the two-machine model to multi-machine model. The model now can be used with arbitrary numbers of machines and buffers composition. The importanet problem that we need to solve, is to arrange the operations that in one time slot the machines and buffers should do.

3.3.1 Mathematical Dirivation of multi-machine Model

Suppose we have an M-machine geometric line observing assumptions 1-8. Because the geometric distribution obey the memoryless attribute, the system can still be described by a Markov chain. Use $j_i(n)$ to indicate the number of parts in buffer b_i at the start of time slot n. Moreover, we can describe the state of Markov chain with vector $(j_1(n), ..., j_{M-1}(n), s_1(n), ..., s_M(n))$, where $j_i(n) \in 0, 1, ..., N_i, i = 1, ..., M-1$, and $s_i(n) \in 0, 1, i = 1, ..., M$. Apparently, the system has totally states of $S = 2^M \sum_{i=1}^{M-1}(N_i + 1)$. To derive the transition probabilities among these states, the same approach described in Section 3.2.1 is applied to linearize the state space. Particularly, the states from State 1 to S are arranged in the following manner: $r(j_1, ..., j_{M-1}, s_1, ..., s_M)$ refer to the state number of the Markov chain state $(j_1, ..., j_{M-1}, s_1, ..., s_M)$. Define

$$r(j_1, ..., j_{M-1}, s_1, ..., s_m) = 1 + \sum_{i=1}^{M-1} j_i \alpha_i + \sum_{i=1}^{M} s_i \beta_i \tag{3.12}$$

where

$$\alpha_i = \begin{cases} 2^M \prod_{h=i+1}^{M-1} (N_h + 1), & i = 1, ..., M-2 \\ 2^M , & i = M-1 \end{cases}$$
$$\beta_i = 2^{M-1}, i = 1, ..., M$$

In this regulation, each state can be arranged a individual number between 1 and S. On the contrary, consider the state number r of a system state, $r \in 1, .., S$, the corresponding machine state $s_i^{(r)}, i = 1, ..., M$, and buffer state, $j_i^{(r)}, i = 1, ..., M-1$, can be derived as follows:

$$j_i^{(r)} = \begin{cases} \left\lfloor \dfrac{r-1}{\alpha_1} \right\rfloor, & i = 1 \\ \left\lfloor \dfrac{r-1-\sum_{h-1}^{i-1} j_h^{(r)} \alpha_h}{\alpha_i} \right\rfloor, & i = 2, ..., M-1 \end{cases} \tag{3.13}$$

$$s_i^{(r)} = \begin{cases} \left\lfloor \dfrac{r-1-\sum_{h-1}^{M-1} j_h^{(r)} \alpha_h}{\beta_1} \right\rfloor, & i = 1 \\ \left\lfloor \dfrac{r-1-\sum_{h-1}^{M-1} j_h^{(r)} \alpha_h - \sum_{h-1}^{i-1} s_h^{(r)} \beta_h}{\beta_i} \right\rfloor, & i = 2, ..., M \end{cases} \tag{3.14}$$

As is shown above, compared to the two-machine production lines model, the state arrangement of both system are very similar, but in this term with enlarged state it can accomodate more buffers and machines.

Alike with the two-machine situation, the transitions of $j_i(n)$'s are deterministic with machine state $s_1(n), ..., s_M(n)$, and can be computed according to the following equations:

$$\begin{aligned} j_i(n+1) &= j_i'(n) + s_i(n) min\{j_{i-1}(n), N_i - j_i'(n), 1\} \\ i &= 2, ..., M-1 \\ j_1(n+1) &= j_1'(n) + s_1(n) min\{N_1 - j_1'(n), 1\} \end{aligned} \tag{3.15}$$

where

$$\begin{aligned} j_{M-1}'(n) &= j_{M-1}(n) - s_M(n) min\{j_{M-1}(n), 1\} \\ j_i'(n) &= j_i(n) - s_{i+1}(n) min\{j_i(n), Ni + 1 - j'i + 1(n), 1\} \\ i &= 1, ..., M-2. \end{aligned} \tag{3.16}$$

In the above equtions, $j_i'(n)$ indicates the occupancy of buffer b_i as loog as machine m_{i+1} takes a products from b_i at the start of time slot n.

Subsequently, according to the state number derived in 3.12 all S system states from 1 to S and describe $x_i(n) = \text{Prob[System in state } i \text{ in time slot } n]$. Afterwards, the procedure of the state transformation of the Markov chain, $x(n) = [x_1(n)x_2(n)...x_S(n)]^T$, can be derived by

$$x(n+1) = A_M x(n), \sum_{i=1}^{S} x_i(n) = 1. \tag{3.17}$$

The transient perfomrmace of the system can be given by

$$\begin{aligned} PR(n) &= \text{Prob}[m_M \text{up and} b_{M-1} \text{not empty at time slot} n] \\ &= \text{Prob}[s_M(n) = 1 \text{and} j_{M-1}(n) > 0] \\ &= D_1 x(n) = [d_{1,1} d_{1,2} ... d_{1,S}] x(n) \\ CR(n) &= \text{Prob}[m_1 \text{is up and not blocked at time slot} n] \\ &= D_2 x(n) = [d_{2,1} d_{2,2} ... d_{2.S}] x(n) \\ WIP_i(n) &= \sum_{k=1}^{N_i} k \cdot \text{Prob}[j_i(n) = k] \\ &= D_{3,i} x(n) = [d_{3,1} d_{3,2} ... d_{3.S}] x(n), i = 1, ..., M-1 \\ ST_i(n) &= \text{Prob}[m_i \text{up and} b_{i-1} \text{empty at time slot} n] \\ &= \text{Prob}[s_i(n) = 1 \text{and} j_{i-1}(n) = 0] \\ &= D_{4,i} x(n) = [d_{4,1} d_{4,2} ... d_{4.S}] x(n), i = 2, ..., M \\ BL_i(n) &= \text{Prob}[m_i \text{up} b_i \text{full, and} m_{i+1} \text{down or blocked at time slot} n] \\ &= D_{5,i} x(n) = [d_{5,1} d_{5,2} ... d_{5.S}] x(n), i = 1, ..., M-1 \end{aligned} \tag{3.18}$$

where

$$d_{1,r} = \begin{cases} 1, \text{if} s_M^{(r)} = 1 \text{and} j_{M-1}^{(r)} > 0 \\ 0, otherwise \end{cases}$$

$$d_{2,r} = 1 - d_{5,1,r}$$

$$d_{3,i,r} = j_i^{(r)}, i = 1, ..., M-1, r = 1, ..., S$$

$$d_{4,i,r} = \begin{cases} 1, \text{if} s_i^{(r)} = 1 \text{and} j_{i-1}^{(r)} = 0 \\ 0, otherwise \end{cases}$$

$$d_{5,i,r} = \begin{cases} d_{5,i+1,r}, & (if) s_i^{(r)} = 1, s_{i+1}^{(r)} = 1, \text{and} j_i^{(r)} = N_i \\ 1, & (if) s_i^{(r)} = 1, s_{i+1}^{(r)} = 0, \text{and} j_i^{(r)} = N_i \\ 0, & otherwise \end{cases}$$

in other words, $d_{1,r}, d_{2,r}, d_{3,i,r}, d_{4,i,r}$ and $d_{5,i,r}$ represent the rth element in vectors D_1, D_2, $D_{3,i}, D_{4,i}$ and $D_{5,i}$, separately. Apparently, in system state $x(n)$ all the performance measures are linear.

As an instance, suppose a four-machine geometric line obeying the assumptions 1-8. with machines and buffer parameters

$$P_i = 0.05, R_i - 0.2, i = 1, ...4; N_i = 5, i = 1, ...3.$$

Presume that all machines are broken down in the beginning and buffers are all empty either. The transient performance of this system are shown in Figure 3.9.

3.3.2 Implementation of multi-machine model

In the python platform, it is also very similar to two-machine model in section 3.9. The **class Individual** and **class Buffer** are also keeped to represent one machine instance and one buffer instance, while they are reorganized in new **class MultiMachine**. In addition, in each **MultiMachine** instance it contains two arrays named as `Machine` and `BufferArray` respectively. The formor array contains a number of `MachineNumber` machines while the latter array holds one less buffer compared to the formor array.

We use a UML diagram to illustrate the relationship between these class, see Figure 3.10. In script "sim4.py" we initialize all the given parameters and create enough instances of **class MultiMachine**. Therefore, in each time slot the critical transient performance measures are collected from function **one_slot()**, and in the end write in the file.

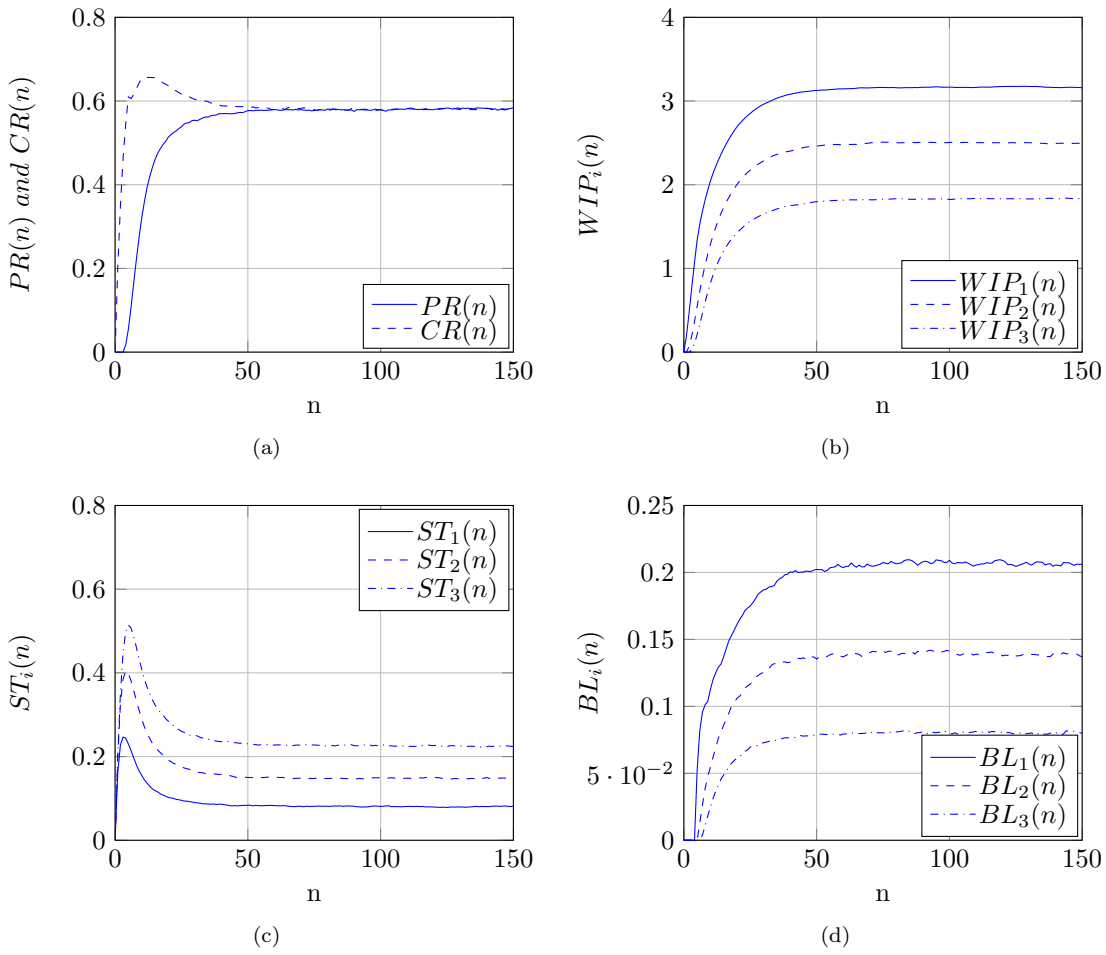


Figure 3.9: Transients of a four-machine geometric line. (a) $PR(n)$ *and* $CR(n)$;(b) $WIP(n)$; (c) $ST_i(n)$;(d) $BL_i(n)$.

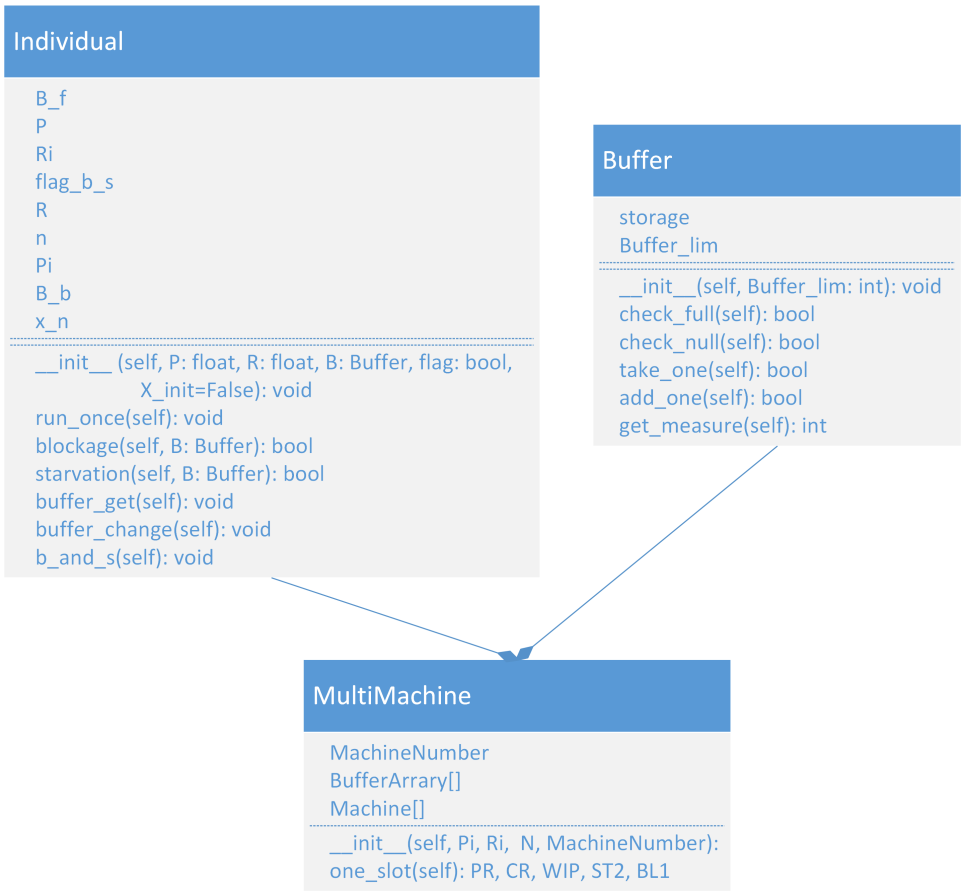


Figure 3.10: UML diagram of a multi-machine model

3.4 Result Analysis

There's no apparently difference between the work of this paper and [36] in one machine mode, but it shows difference in two-machine lines especially in $ST_2(n)$ and BL_1 these two parameters. From the Figure 3.11 can we tell the differece. And in the four-machine lines model there are more difference shown in Figure 3.12.

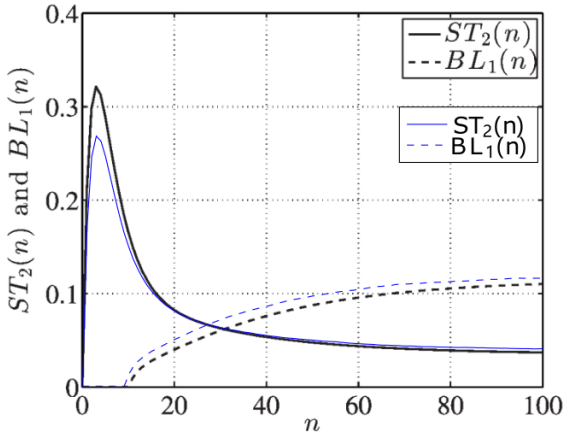


Figure 3.11: Performance contrast of a two-machine geometric line $ST_2(n)$ *and* $BL_2(n)$

For the reasons of these difference, we suppose a few points: First, we doubt that whether the lack of enough quantity of samples cause the problem. But when we enlarege the quantity of simulation time from 10000 to 100000, we found no apparent difference from the result. Second, due to that we don't know what kind of programming language the reseachers coding with, a probable cause may be we take a different platform to programm with, and that may cause some accuracy problem. Third, we also doubt that the researchers may do some kind of curve smoothing in oder to present a perfect result.

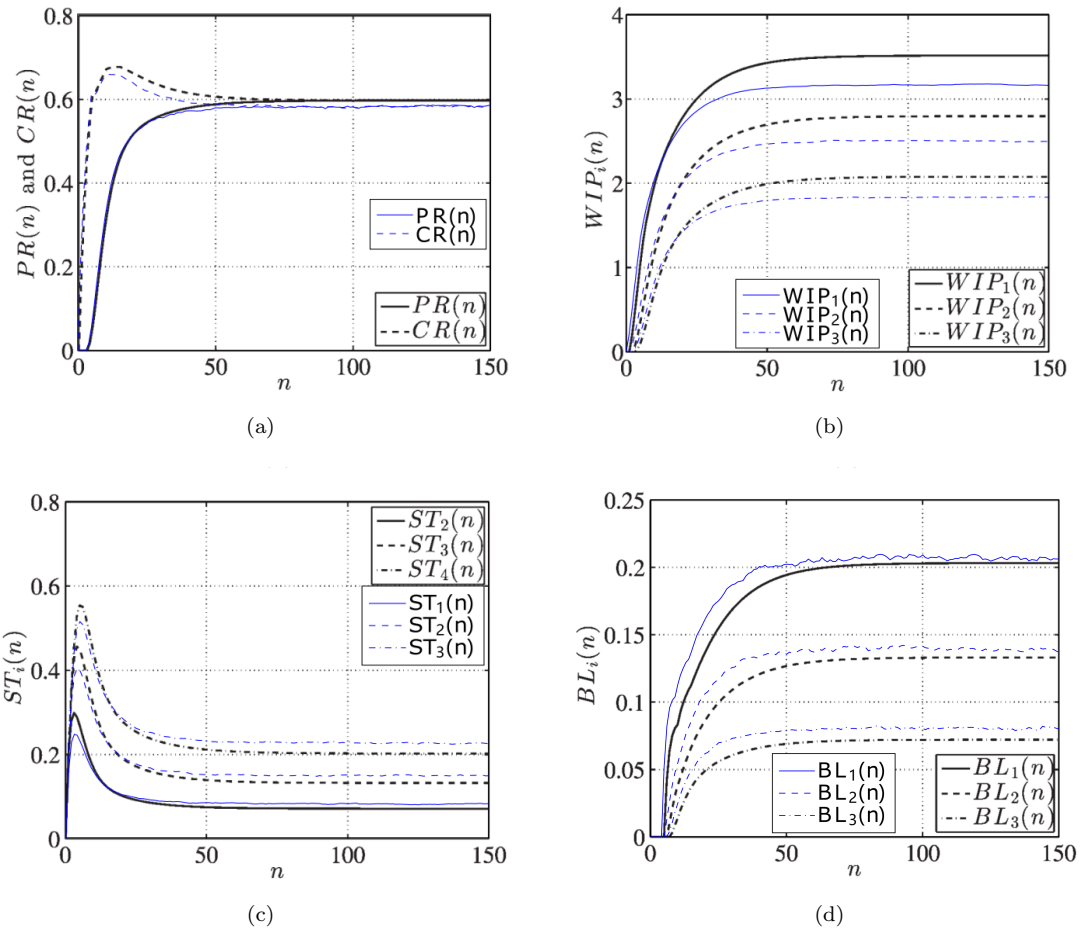


Figure 3.12: Transients of a four-machine geometric line. (a) $PR(n)$ *and* $CR(n)$;(b) $WIP(n)$; (c) $ST_i(n)$;(d) $BL_i(n)$.

4 Impact Analysis in Variation of the Buffer Size

In this chapter we attepmt to explore the relationship between the paramters and the transient performance of a multi-machine line with five machines.

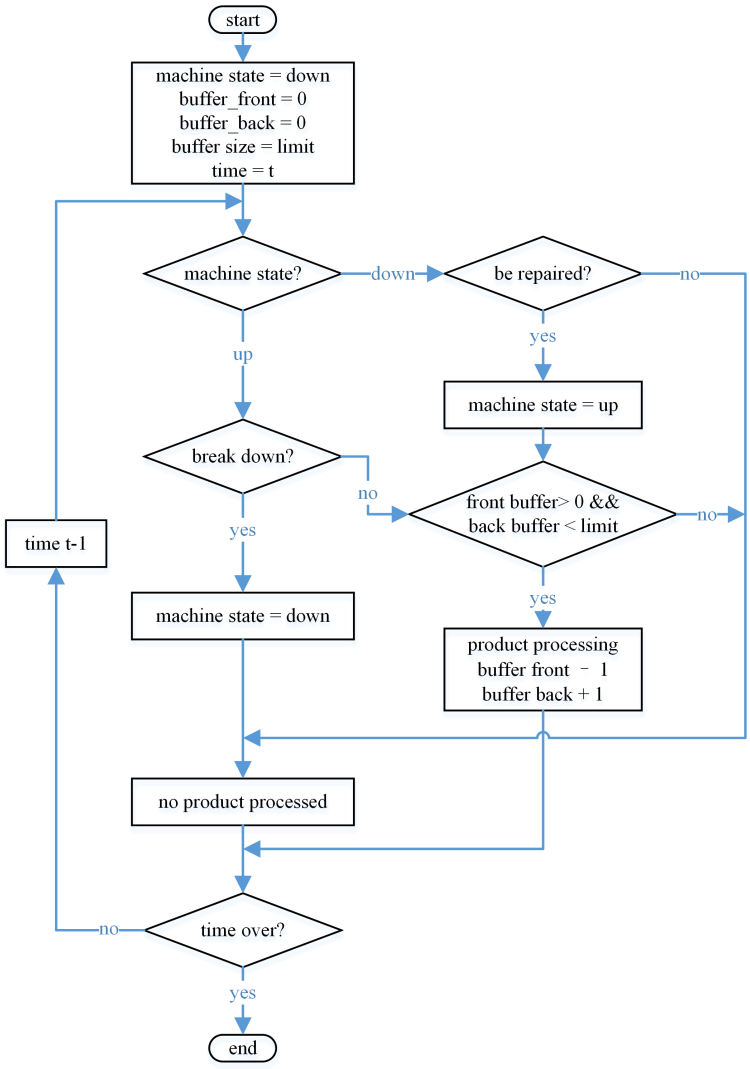


Figure 4.1: Flow chart of Procedure.

4.1 Procedure Description

We suppose that a suitable buffer size can help to improve the transient performance of the production line. On the other hand, if the buffer size is designed . Therefore, we design

a experiment on a five-machine production-line model. We use the following parameters:

$$P = 0.05, R = 0.2, N = 1, ...20, \text{All Buffer storage} = \text{epmty}$$

In order to describe the procedure, we use a flow chart to illustrate the critical part of the program logic. At the beginning, we initialize all the parameters given in the file "simulation.py". Next, the parameters are passed to the file called "multi_machine.py", and a certain amount of this object are created from the **class MultiMachine** to calculate the average of the cirtical values. Furthermore, the related machine objects and buffer objects will be created and simulated according to the procedure shown in flow chart.

When the programm creates a simulated geomatric machine line, it first initiallize states of all machines and storages of all buffers. After that, in each time slot two conditional function will be called sequentially in order to make sure whether the machine break down , and if it can to be repaired or may break down according to its present state espectively. If the machine is in good condition (i.e. it can work), thereupon another conditional function will be called to test if the buffer in front of the machine has at least one product (except the first machine) or the buffer at the end of the machine has enough space to receive one new product (except the last machine). If all these conditions filling, then the machine produce a product, and at the same time take one product from the front buffer with put a product in the back buffer.

4.2 Results and Discussion

After the simulation, we collected the data of evaluation of transient performance of the geomatric machine lines. The Figure 4.2(a) shows with the buffer size enlarging the trend of the Production Rate. Figure 4.2(b) shows that this trend of Consumption Rate.

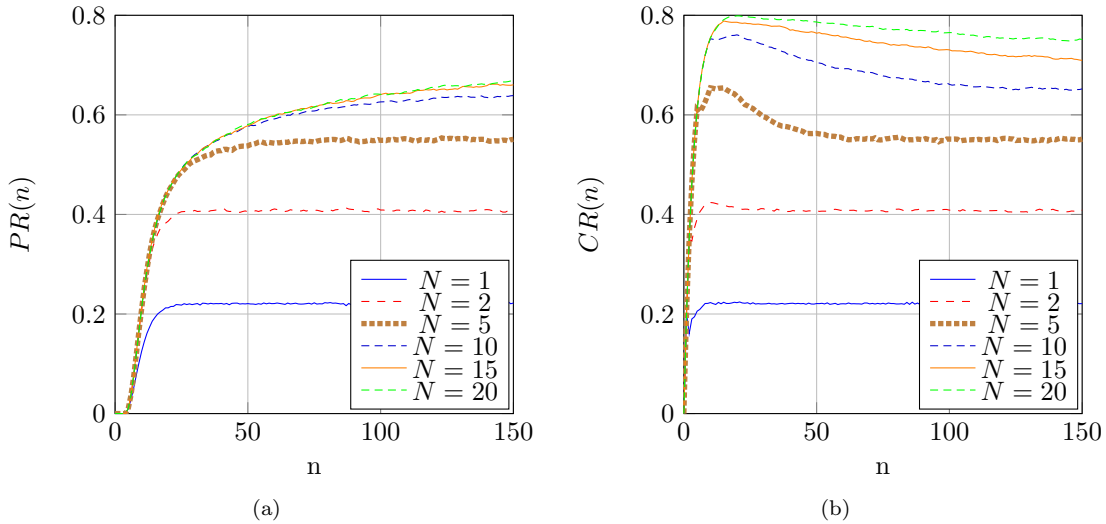


Figure 4.2: Variation of Transient Parameters. (a) Production Rate; (b) Consumption Rate.

From the diagram we figure out that with the expanding of the buffer size, the steady state of production rate will rise rapidly at the beginning, however, when the buffer size exceeds over 10, it will be extremly small in the variation of the steady state of the production rate, and almost is undetectable. Nevertheless, with the growth of the buffer size, it takes apparently more time to reach the steady state. In a similar way, this rule works also in the variation of consumption rate.

In the end, we can come to a conclusion, that in a multi-machine geomatric line model with the parameter of five machines, broken probability Pi of 0.02, repair probablity Ri of 0.5 and initial buffer all empty can get a good balance of performance in Production Rate and Consumption Rate when buffer sized are settled in 10.

5 Conclusion

Nowadays, improving productivity and quality has becoming a core concern in manufacturing research. With global energy crisis and environmental problems, reducing energy costs and waste gas emission has been a significant issue for the manufacturing industry. In order to solve these problems, researchers investigate a lot of effort in the study of production systems.

Obviously, in comparison to the enrmous studies on steady state performance of production lines [11, 12], very few results have been made in literature on the transient behavior of production systems. The goal of this research is to use the python to implement the simulations of the geometric machine model, and optimize the buffer size of a five-machine production lines.

In this paper, we studied the transient performance evaluation of serial production lines with machines with geometric reliability model and finite buffers. The contribution are givne in this work as follows:

- First part we review the research works of production systems, then make a summary, i.e. the past works in this field most focused on the steady performance of the production lines, while the transient evaluation still need more effort to be devoted. The detailed derivations of the model are given in Chapter 2.
- Next, we make a brief introduction of the theory basis. In addition, the assumptions of the model of serial produciton line systems are presented and the performance measures are defined. The implementation of the system models are give in next chapter.
- In Chapter 3, we use the python to implement three models, the single-machine production lines, two-machine with a buffer, and multi-machine with buffers. And we separately give the performance measures according to each kind of production lines. On the other hand, we make a comparison of the results with another work, and attempt to analyse the difference of the results.
- Moreover, in the next chapter we also make a study about the impact of buffer sizes on transient performace. In order to find the most suitable buffer size, we designed a procedure to analyse the critical transient performance. Eventual, it come to a conclusion that for a five-machine serial production lines 10 buffer is a more balanced size for each buffer.

It is plausible that still some limitations may exist in the results obtained. Firstly, the computational costs are still very large especially when the parameter changes and more production line devoted to calculate the mathematical expectation. In addition, the bottleneck of the prodution lines may still be dedicated efforts to investigate , and continuous improvement can be studied. Furthermore, the the structure of the model

may be still simplified. Assembly systems and other production with complex structures remains to be explore.

A Source Code of all models

The followings are the python source code according to different models
The Individual machine model
File "individual.py"

```
import random


class Individual:

    def __init__(self, Pi: float, Ri: float, n: int, X_init=0):
        assert (0 <= Pi and Pi <= 1) is True
        assert (0 <= Ri and Ri <= 1) is True
        assert (0 == X_init or X_init == 1) is True
        self.Pi = Pi  # breakdown probability
        self.Ri = Ri  # repair probability
        self.n = n  # number of slot
        self.x_n = X_init  # state of x

    def run_once(self) -> int:  # simulation of a time slot
        if self.x_n == 0:
            if random.random() <= self.Ri:
                self.x_n = 1
        else:
            if random.random() <= self.Pi:
                self.x_n = 0
        return self.x_n
```

File "simu1.py"

```
from individual import Individual
P = 0.05  # break down probability
R = 0.2  # repair probability
n = 40  # numbers of slot
X_init = 1  # initial state of machine
max_range = 1000000  # maxmal quantity of the production lines
sum = []
indi = []

for i in range(max_range):  # initial state of 1
    indi.append(Individual(P, R, n, X_init))
```

```
for i in range(n):
    sum.append(0)
    for j in range(max_range):
        sum[i] += indi[j].run_once()
    sum[i] = (sum[i]/max_range)

str_sum = 'initial 1\n'
i = 1
str_sum += 'x=1\n'
for e in sum:
    str_sum += '(' + str(i) + ',' + str(e) + ')\n'
    i += 1

i = 1
str_sum += 'x=0\n'
for e in sum:
    str_sum += '(' + str(i) + ',' + str(1-e) + ')\n'
    i += 1

# force initial
indi = []
sum = []

for i in range(max_range):  # initial state of 0
    indi.append(Individual(P, R, n))

for i in range(n):
    sum.append(0)
    for j in range(max_range):
        sum[i] += indi[j].run_once()
    sum[i] = (sum[i]/max_range)

str_sum += 'initial 0\n'
i = 1
str_sum += 'x=1\n'
for e in sum:
    str_sum += '(' + str(i) + ',' + str(e) + ')\n'
    i += 1

i = 1
str_sum += 'x=0\n'
for e in sum:
    str_sum += '(' + str(i) + ',' + str(1-e) + ')\n'
    i += 1
```

```
file = open("result.txt", 'w')
file.truncate()
file.write(str_sum)
file.close()
```

The Two-machine model
File "individual.py"

```
import random
from buffer import Buffer


class Individual:

    def __init__(self, P: float, R: float,
                 B: Buffer, flag: bool, X_init=False):
        self.P = P  # breakdown probability
        self.R = R  # repair probability
        self.x_n = X_init  # state of x false represent down
        self.flag_b_s = False
        # True represents the machine is blockage or stravation
        if flag:  # flag is a special condition control
            self.B_b = B  # Buffer backward
            self.B_f = None
        else:
            self.B_b = None
            self.B_f = B  # Buffer forward

    # simulation of a time slot
    def run_once(self):  # -> int bool

        ONE_PRODUCT = 1
        NO_PRODUCT = 0

        if self.x_n is False:
            if random.random() < self.R:
                self.x_n = True
        else:
            if random.random() < self.P:
                self.x_n = False

        if (self.x_n and (not self.flag_b_s)):
            self.buffer_change()
            return ONE_PRODUCT, self.x_n
        else:
```

```
            return NO_PRODUCT, self.x_n

    def blockage(self, B: Buffer) -> bool:
        return B.check_full()

    def starvation(self, B: Buffer) -> bool:
        return B.check_null()

    def b_and_s(self):
        if self.B_f is None:
            self.flag_b_s = self.blockage(self.B_b)
        elif self.B_b is None:
            self.flag_b_s = self.starvation(self.B_f)
        else:
            self.flag_b_s = self.starvation(self.B_f) \
                or self.blockage(self.B_b)

    def buffer_change(self):
        if self.B_f is None:
            self.B_b.add_one()
        elif self.B_b is None:
            self.B_f.take_one()
        else:
            self.B_b.add_one()
            self.B_f.take_one()

    def buffer_get(self):
        if self.B_f is None:
            return self.B_b.get_measure()
        elif self.B_b is None:
            return self.B_f.get_measure()
```

File "buffer.py"

```
class Buffer:

    def __init__(self, Buffer_lim: int):
        self.Buffer_lim = Buffer_lim
        self.storage = 0

    def check_null(self) -> bool:
        if self.storage == 0:
            return True
        else:
            return False
```

```
    def check_full(self) -> bool:
        if self.storage == self.Buffer_lim:
            return True
        else:
            return False

    def add_one(self) -> bool:
        if self.storage + 1 > self.Buffer_lim:
            return False
        else:
            self.storage += 1
            return True

    def take_one(self) -> bool:
        if self.storage == 0:
            return False
        else:
            self.storage -= 1
            return True

    def get_measure(self) -> int:
        return self.storage
```

File "twomachine.py"

```
from individual import Individual
from buffer import Buffer


class TwoMachine:

    def __init__(self, P1, R1, P2, R2, N):
        self.B1 = Buffer(N)
        self.M1 = Individual(P1, R1, self.B1, True)
        self.M2 = Individual(P2, R2, self.B1, False)

    def one_slot(self):
        # step 1: judge if runable
        self.M1.b_and_s()
        self.M2.b_and_s()
        # step 2: begin run
        CR, M1_s = self.M1.run_once()
        PR, M2_s = self.M2.run_once()
        WIP = self.B1.get_measure()
        if M2_s and self.B1.check_null():
            ST2 = 1
```

```
        else:
            ST2 = 0
        if M1_s and (not M2_s) and self.B1.check_full():
            BL1 = 1
        else:
            BL1 = 0
        return PR, CR, WIP, ST2, BL1
```

File "sim2.py"

```
from two_machine import TwoMachine

P1 = 0.03  # break down probability machine 1
R1 = 0.18  # repair probability machine 1
P2 = 0.06  # break down probability machine 2
R2 = 0.21  # repair probability machine 2
N = 10  # Buffer space
n = 100  # slot of time
max_range = 1000000
indi = []
PR = []
CR = []
WIP = []
ST2 = []
BL1 = []

for i in range(max_range):  # initial
    indi.append(TwoMachine(P1, R1, P2, R2, N))

for i in range(n):
    PR.append(0.0)
    CR.append(0.0)
    WIP.append(0.0)
    ST2.append(0.0)
    BL1.append(0.0)
    for j in range(max_range):
        pr_t, cr_t, wip_t, st2_t, bl1_t = indi[j].one_slot()
        PR[i] += pr_t
        CR[i] += cr_t
        WIP[i] += wip_t
        ST2[i] += st2_t
        BL1[i] += bl1_t
    PR[i] = PR[i]/max_range
    CR[i] = CR[i]/max_range
    WIP[i] = WIP[i]/max_range
    ST2[i] = ST2[i]/max_range
```

```
    BL1[i] = BL1[i]/max_range

str_out = ''
i = 1
str_out += 'PR\n'
for e in PR:
    str_out += '(' + str(i) + ',' + str(e) + ')\n'
    i += 1

i = 1
str_out += 'CR\n'
for e in CR:
    str_out += '(' + str(i) + ',' + str(e) + ')\n'
    i += 1

i = 1
str_out += 'WIP\n'
for e in WIP:
    str_out += '(' + str(i) + ',' + str(e) + ')\n'
    i += 1

i = 1
str_out += 'ST2\n'
for e in ST2:
    str_out += '(' + str(i) + ',' + str(e) + ')\n'
    i += 1

i = 1
str_out += 'BL1\n'
for e in BL1:
    str_out += '(' + str(i) + ',' + str(e) + ')\n'
    i += 1

file = open("result_sim.txt", 'w')
file.truncate()
file.write(str_out)
file.close()
```

The Multi-machine model
File "individual.py"

```
import random
from buffer import Buffer


class Individual:
```

```
def __init__(self, Pi: float, Ri: float,
             B_f: Buffer,  B_b: Buffer, X_init=False):
    assert (0 <= Pi and Pi <= 1) is True
    assert (0 <= Ri and Ri <= 1) is True
    self.Pi = Pi  # breakdown probability
    self.Ri = Ri  # repair probability
    self.x_n = X_init  # state of x false represent down
    self.flag_b_s = False
    # True represents the machine is blockage or stravation
    if B_f is  None:  # flag is a special condition control
        self.B_b = B_b  # Buffer backward
        self.B_f = None
    elif B_b is None:
        self.B_b = None
        self.B_f = B_f  # Buffer forward
    else:
        self.B_f = B_f  # Buffer forward
        self.B_b = B_b  # Buffer backward

# simulation of a time slot
def run_once(self):  # -> int bool

    ONE_PRODUCT = 1
    NO_PRODUCT = 0

    if self.x_n is False:
        if random.random() < self.Ri:
            self.x_n = True
    else:
        if random.random() < self.Pi:
            self.x_n = False

    if (self.x_n and (not self.flag_b_s)):
        self.buffer_change()
        # print("A",self.flag_b_s ,self.x_n)
        return ONE_PRODUCT, self.x_n
    else:
        # print("B",self.flag_b_s ,self.x_n)
        return NO_PRODUCT, self.x_n

def blockage(self, B: Buffer) -> bool:
    return B.check_full()
```

```
    def starvation(self, B: Buffer) -> bool:
        return B.check_null()

    def b_and_s(self):
        if self.B_f is None:
            # print("blo")
            self.flag_b_s = self.blockage(self.B_b)
        elif self.B_b is None:
            # print("fail")
            self.flag_b_s = self.starvation(self.B_f)
        else:
            self.flag_b_s = self.starvation(self.B_f) \
                or self.blockage(self.B_b)

    def buffer_change(self):
        if self.B_f is None:
            self.B_b.add_one()
        elif self.B_b is None:
            self.B_f.take_one()
        else:
            self.B_b.add_one()
            self.B_f.take_one()
```

File "buffer.py"

```
class Buffer:

    def __init__(self, Buffer_lim: int):
        self.Buffer_lim = Buffer_lim
        self.storage = 0

    def check_null(self) -> bool:
    #     print("check n")
        if self.storage == 0:
            return True
        else:
            return False

    def check_full(self) -> bool:
        if self.storage == self.Buffer_lim:
            return True
        else:
            return False

    def add_one(self) -> bool:
        if self.storage + 1 > self.Buffer_lim:
```

```
            return False
        else:
            self.storage += 1
            return True

    def take_one(self) -> bool:
        if self.storage == 0:
            return False
        else:
            self.storage -= 1
            return True

    def get_measure(self) -> int:
        # print(self.storage)
        return self.storage
```

File "multimachine.py"

```
from individual import Individual
from buffer import Buffer

class MultiMachine:

    MachineNumber = 4

    def __init__(self, Pi, Ri, N, MachineNumber):
        self.MachineNumber = MachineNumber
        self.BufferArray = []
        for i in range(self.MachineNumber-1): \
            self.BufferArray.append(Buffer(N))

        self.Machine = []
        self.Machine.append(Individual(Pi, Ri, None, \
            self.BufferArray[0]))
        for i in range(self.MachineNumber-2):
            self.Machine.append(Individual(Pi, Ri, \
                self.BufferArray[i],self.BufferArray[i+1]))
        self.Machine.append(Individual(Pi, Ri, \
            self.BufferArray[self.MachineNumber-2], None))

    def one_slot(self):
        # step 1: judge if runable
        for i in range(self.MachineNumber): \
            self.Machine[i].b_and_s()
        # step 2: begin run
        M = [0 for i in range(self.MachineNumber)]
```

```
        CR, M[0] = self.Machine[0].run_once()
        for i in range(self.MachineNumber-2):
            M[i+1] = self.Machine[i+1].run_once()[1]
        PR, M[self.MachineNumber-1] = \
            self.Machine[self.MachineNumber-1].run_once()
        WIP = []
        ST = []
        BL = []
        for i in range(self.MachineNumber-1):
            WIP.append(self.BufferArray[i].get_measure())


        for i in range(self.MachineNumber-1):
            if M[i+1] and self.BufferArray[i].check_null():
                ST.append(1)
            else:
                ST.append(0)

        for i in range(self.MachineNumber-2):
            if M[i] and self.BufferArray[i].check_full() \
                and (not M[i+1] or \
                self.BufferArray[i+1].check_full()):
                BL.append(1)
            else:
                BL.append(0)
        if M[self.MachineNumber-2] and \
            self.BufferArray[self.MachineNumber-2].check_full() \
                and (not M[self.MachineNumber-1] ):
            BL.append(1)
        else:
            BL.append(0)

        return PR, CR, WIP, ST, BL
```

File "sim4.py"

```
from multi_machine import MultiMachine

Pi = 0.05  # break down probability
Ri = 0.2  # repair probability
N = 5  # Buffer space
n = 150  # slot of time
max_range = 1000000
indi = []
PR = [0 for i in range(n)]
CR = [0 for i in range(n)]
```

```
WIP = []
ST = []
BL = []

for i in range(max_range):  # initial
    indi.append(MultiMachine(Pi, Ri, N))

for i in range(n):
    WIP.append([0 for j in \
        range(MultiMachine.MachineNumber-1)])
    ST.append([0 for j in range(MultiMachine.MachineNumber-1)])
    BL.append([0 for j in range(MultiMachine.MachineNumber-1)])
    for j in range(max_range):
        pr_t, cr_t, wip_t, st_t, bl_t = indi[j].one_slot()
        PR[i] += pr_t
        CR[i] += cr_t
        for k in range(MultiMachine.MachineNumber-1):
            WIP[i][k] += wip_t[k]
            ST[i][k] += st_t[k]
            BL[i][k] += bl_t[k]

    PR[i] = PR[i]/max_range
    CR[i] = CR[i]/max_range
    for k in range(MultiMachine.MachineNumber-1):
            WIP[i][k] = WIP[i][k]/max_range
            ST[i][k] = ST[i][k]/max_range
            BL[i][k] = BL[i][k]/max_range

str_out = ''
i = 1
str_out += 'PR\n'
for e in PR:
    str_out += '(' + str(i) + ',' + str(e) + ')\n'
    i += 1

i = 1
str_out += 'CR\n'
for e in CR:
    str_out += '(' + str(i) + ',' + str(e) + ')\n'
    i += 1

for j in range(3):
    i = 1
    str_out += str(j+1) + 'WIP\n'
```

```
    for e in WIP:
        str_out += '(' + str(i) + ',' + str(e[j]) + ')\n'
        i += 1

for j in range(3):
    i = 1
    str_out += str(j+1) + 'ST\n'
    for e in ST:
        str_out += '(' + str(i) + ',' + str(e[j]) + ')\n'
        i += 1

for j in range(3):
    i = 1
    str_out += str(j+1) + 'BL\n'
    for e in BL:
        str_out += '(' + str(i) + ',' + str(e[j]) + ')\n'
        i += 1

file = open("result_sim.txt", 'w')
file.truncate()
file.write(str_out)
file.close()
```

The study in Chapter 4
File "individual.py" is same as file "individual.py" in Multi-machine model
File "buffer.py" is same as file "buffer.py" in Multi-machine model
File "multimachine.py" is same as file "multimachine.py" in Multi-machine model
File "simulation.py"

```
from multiprocessing import Process
import numpy as np
from multi_machine import MultiMachine

class procedure1:


    def __init__(self, MachineNumber = 5):
        self.MachineNumber = MachineNumber
        n = 150  # slot of time
        simulation_num = 100000
        process_all = 20
        for i in range(process_all):
            Process(target=self.calculation, \
                args=( i, n, simulation_num)).start()

    def calculation(self, process_num, n, simulation_num):
```

```
Pi = 0.05  # break down probability
Ri = 0.2  # repair probability
N = process_num +1  # Buffer space

indi = []
PR = []
CR = []
WIP = []
ST = []
BL = []

for i in range(simulation_num):  # initial
    indi.append(MultiMachine(Pi, Ri, N, \
        self.MachineNumber))

for i in range(n):
    PR.append(0)
    CR.append(0)
    WIP.append([0 for j in \
        range(self.MachineNumber-1)])
    ST.append([0 for j in range(self.MachineNumber-1)])
    BL.append([0 for j in range(self.MachineNumber-1)])


    for j in range(simulation_num):

        pr_t, cr_t, wip_t, st_t, bl_t = \
            indi[j].one_slot()
        PR[i] += pr_t
        CR[i] += cr_t
        for k in range(self.MachineNumber-1):
            WIP[i][k] += wip_t[k]
            ST[i][k] += st_t[k]
            BL[i][k] += bl_t[k]

    PR[i] = PR[i]/simulation_num
    CR[i] = CR[i]/simulation_num
    for k in range(self.MachineNumber-1):
            WIP[i][k] = WIP[i][k]/simulation_num
            ST[i][k] = ST[i][k]/simulation_num
            BL[i][k] = BL[i][k]/simulation_num


str_out = ''
```

```
        i = 1
        str_out += 'PR\n'
        for e in PR:
            str_out += '(' + str(i) + ',' + str(e) + ')\n'
            i += 1

        i = 1
        str_out += 'CR\n'
        for e in CR:
            str_out += '(' + str(i) + ',' + str(e) + ')\n'
            i += 1

        for j in range(self.MachineNumber-1):
            i = 1
            str_out += str(j+1) + 'WIP\n'
            for e in WIP:
                str_out += '(' + str(i) + ',' \
                    + str(e[j]) + ')\n'
                i += 1

        for j in range(self.MachineNumber-1):
            i = 1
            str_out += str(j+2) + 'ST\n'
            for e in ST:
                str_out += '(' + str(i) + ',' \
                    + str(e[j]) + ')\n'
                i += 1

        for j in range(self.MachineNumber-1):
            i = 1
            str_out += str(j+1) + 'BL\n'
            for e in BL:
                str_out += '(' + str(i) + ',' \
                    + str(e[j]) + ')\n'
                i += 1

        string_name = 'process' + str(process_num) \
            + 'result_sim' + str(self.MachineNumber)
        file = open('{}.txt'.format(string_name), 'w')
        file.truncate()
        file.write(str_out)
        file.close()

if __name__ == "__main__":
```

```
procedure1()
```

List of References

[1] Papadopolous, HT ; Heavey, Cathal ; Browne, Jimmie: *Queueing theory in manufacturing systems analysis and design*. Springer Science & Business Media, 1993

[2] Buzacott, John A. ; Shanthikumar, J G.: *Stochastic models of manufacturing systems*. Bd. 4. Prentice Hall Englewood Cliffs, NJ, 1993

[3] Askin, Ronald G. ; Standridge, Charles R.: *Modeling and analysis of manufacturing systems*. John Wiley & Sons Inc, 1993

[4] Bonomi, Flavio: An approximate analysis for a class of assembly-like queues. In: *Queueing Systems* 1 (1987), Nr. 3, S. 289–309

[5] RAO, P C. ; Suri, Rajan: Performance analysis of an assembly station with input from multiple fabrication lines. In: *Production and Operations Management* 9 (2000), Nr. 3, S. 283–302

[6] Harrison, J M.: Assembly-like queues. In: *Journal of Applied Probability* 10 (1973), Nr. 2, S. 354–367

[7] Kuo, C-T ; Lim, J-T ; Meerkov, SM ; Park, E: Improvability theory for assembly systems: Two component—One assembly machine case. In: *Mathematical Problems in Engineering* 3 (1996), Nr. 2, S. 95–171

[8] Lipper, EH ; Sengupta, Bhaskar: Assembly-like queues with finite capacity: bounds, asymptotics and approximations. In: *Queueing Systems* 1 (1986), Nr. 1, S. 67–83

[9] Manitz, Michael: Queueing-model based analysis of assembly lines with finite buffers and general service times. In: *Computers & Operations Research* 35 (2008), Nr. 8, S. 2520–2536

[10] Rao, P C. ; Suri, Rajan: Approximate queueing network models for closed fabrication/assembly systems. part I: Single level systems. In: *Production and Operations Management* 3 (1994), Nr. 4, S. 244–275

[11] Gershwin, Stanley B.: Assembly/disassembly systems: An efficient decomposition algorithm for tree-structured networks. In: *IIE TRANSACTIONS* 23 (1991), Nr. 4, S. 302–314

[12] Liu, Xiao-Gao ; Buzacott, John A.: Approximate models of assembly systems with finite inventory banks. In: *European Journal of Operational Research* 45 (1990), Nr. 2-3, S. 143–154

[13] Helber, Stefan: Decomposition of unreliable assembly/disassembly networks with limited buffer capacity and random processing times. In: *European Journal of Operational Research* 109 (1998), Nr. 1, S. 24–42

[14] MASCOLO, MARIA D. ; David, Rene ; Dallery, Yves: Modeling and analysis of assembly systems with unreliable machines and finite buffers. In: *IIE transactions* 23 (1991), Nr. 4, S. 315–330

[15] Chiang, S-Y ; Kuo, C-T ; Lim, J-T ; Meerkov, SM: Improvability of assembly systems I: Problem formulation and performance evaluation. In: *Mathematical Problems in Engineering* 6 (1999), S. 321–357

[16] Chiang, S-Y ; Kuo, C-T ; Lim, J-T ; Meerkov, SM: Improvability of assembly systems II: Improvability indicators and case study. In: *Mathematical Problems in Engineering* 6 (1999), S. 359–393

[17] Li, Jingshan ; E. Blumenfeld, Dennis ; Huang, Ningjian ; M. Alden, Jeffrey: Throughput analysis of production systems: recent advances and future topics. In: *International Journal of Production Research* 47 (2009), Nr. 14, S. 3823–3851

[18] Meerkov, Semyon M. ; Zhang, Liang: Transient behavior of serial production lines with Bernoulli machines. In: *IIE Transactions* 40 (2008), Nr. 3, S. 297–312

[19] Meerkov, SM ; Zhang, L: Transients in production lines with two non-identical Bernoulli machines. In: *Proceedings of 7th International Conference on the Analysis of Manufacturing Systems*, 2009, S. 212–221

[20] Meerkov, Semyon M. ; Zhang, Liang: Unbalanced production systems with floats: Analysis and lean design. In: *International journal of manufacturing technology and management* 23 (2011), Nr. 1-2, S. 4–15

[21] Zhang, Liang ; Wang, Chuanfeng ; Arinez, Jorge ; Biller, Stephan: Transient analysis of Bernoulli serial lines: Performance evaluation and system-theoretic properties. In: *IIE Transactions* 45 (2013), Nr. 5, S. 528–543

[22] Chen, Guorong ; Zhang, Liang ; Arinez, Jorge ; Biller, Stephan: Energy-efficient production systems through schedule-based operations. In: *IEEE Transactions on Automation Science and Engineering* 10 (2012), Nr. 1, S. 27–37

[23] Wang, Junwen ; Hu, Yao ; Li, Jingshan: Transient analysis to design buffer capacity in dairy filling and packing production lines. In: *Journal of Food Engineering* 98 (2010), Nr. 1, S. 1–12

[24] Chen, Guorong ; Zhang, Liang ; Arinez, Jorge ; Biller, Stephan: Feedback control of machine startup for energy-efficient manufacturing in Bernoulli serial lines. In: *2011 IEEE International Conference on Automation Science and Engineering* IEEE, 2011, S. 666–671

[25] Meerkov, Semyon M. ; Shimkin, Nahum ; Zhang, Liang: Transient behavior of two-machine geometric production lines. In: *IEEE Transactions on Automatic Control* 55 (2010), Nr. 2, S. 453–458

[26] Gagniuc, Paul A.: *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017

[27] Chung, Kai L.: Markov chains. In: *Springer-Verlag, New York* (1967)

[28] Gershwin, Stanley B. ; Gershwin, SB: Manufacturing systems engineering. (1994)

[29] Altiok, Tayfur: *Performance analysis of manufacturing systems*. Springer Science & Business Media, 1997

[30] Li, Jingshan ; Meerkov, Semyon M.: Due-time performance of production systems with Markovian machines. In: *Analysis and modeling of manufacturing systems*. Springer, 2003, S. 221–253

[31] Liberopoulos, George ; Kozanidis, George ; Tsarouhas, Panagiotis: Performance evaluation of an automatic transfer line with WIP scrapping during long failures. In: *Manufacturing & Service Operations Management* 9 (2007), Nr. 1, S. 62–83

[32] Li, Jingshan ; Blumenfeld, Dennis E. ; Alden, Jeffrey M.: Comparisons of two-machine line models in throughput analysis. In: *International journal of production research* 44 (2006), Nr. 7, S. 1375–1398

[33] Dallery, Yves ; Gershwin, Stanley B.: Manufacturing flow line systems: a review of models and analytical results. In: *Queueing systems* 12 (1992), Nr. 1-2, S. 3–94

[34] Papadopoulos, HT ; Heavey, Cathal: Queueing theory in manufacturing systems analysis and design: A classification of models for production and transfer lines. In: *European journal of operational Research* 92 (1996), Nr. 1, S. 1–27

[35] Li, Jingshan ; Meerkov, Semyon M.: *Production systems engineering*. Springer Science & Business Media, 2008

[36] Chen, Guorong ; Wang, Chuanfeng ; Zhang, Liang ; Arinez, Jorge ; Xiao, Guoxian: Transient performance analysis of serial production lines with geometric machines. In: *IEEE Transactions on Automatic Control* 61 (2015), Nr. 4, S. 877–891