ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Προσομοίωση και Ανάλυση Ευαισθησίας της Απόδοσης Πολιτικών Προγραμματισμού Παρτίδων Παραγωγής

Θεόδωρος Α. Θωμάς

Επιβλέπων: Βασίλειος Σ. Κουϊκόγλου, Καθηγητής

Μέλη Τριμελούς Επιτροπής:

Βασίλειος Σ. Κουϊκόγλου

Ευστράτιος Ιωαννίδης

Γεώργιος Ι. Τσιναράκης

Χανιά, Σεπτέμβριος 2025

Technical University of Crete

School of Production Engineering and Management

Diploma Thesis

Simulation and Performance Sensitivity Analysis of Lot Scheduling Policies

Theodoros A. Thomas

Supervisor: Vassilis S. Kouikoglou, Professor

Thesis Committee Members:

Vassilis S. Kouikoglou

Stratos Ioannidis

George J. Tsinarakis

Chania, September 2025

ΕΥΧΑΡΙΣΤΙΕΣ / ACKNOWLEDGEMENTS

Με την περάτωση της παρούσας διπλωματικής εργασίας θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου, κύριο Βασίλη Κουϊκόγλου για την αμέριστη και άμεση βοήθεια, την αδιάκοπη καθοδήγηση, την υπομονή και την κατανόηση από την πρώτη στιγμή της συνεργασίας μας. Η συμβολή του υπήρξε καθοριστική για την ολοκλήρωση αυτής της προσπάθειας.

Επιπλέον, θα ήθελα να ευχαριστήσω τους γονείς μου για την ευκαιρία που μου έδωσαν να σπουδάσω στο Πολυτεχνείο Κρήτης, για τη συνεχή στήριξη των επιλογών μου και για τη βοήθεια τους στην επίτευξη των στόχων μου.

Ακόμη, θα ήθελα να ευχαριστήσω την κυρία Τατιάνα, την κυρία Καλλιόπη και τον κύριο Λάμπρο για το ενδιαφέρον και τη διαρκή υποστήριξη τους όλο αυτό το διάστημα.

Τέλος, ευχαριστώ τους συμφοιτητές, Παππά Αναστασία και Γκούβερη Κωνσταντίνο, με τους οποίους υπήρξε συνεργασία κατά τη διάρκεια των σπουδών.

ΠΕΡΙΛΗΨΗ

Θεωρείται ένα σύστημα παραγωγής διαφορετικών προϊόντων κατά παρτίδες ώστε να αποφεύγονται συχνές προετοιμασίες για αλλαγές προϊόντων οι οποίες επιφέρουν κόστος και καθυστερήσεις. Το σύστημα ικανοποιεί παραγγελίες με τυχαίους χρόνους άφιξης και τυχαία μεγέθη και ποικιλίες προϊόντων. Μια πολιτική παραγωγής είναι ένας κανόνας απόφασης που καθορίζει τους χρόνους έναρξης και περάτωσης της παραγωγής κάθε προϊόντος όταν όλα παράγονται από μια μηχανή. Με την προσομοίωση μπορεί κάποιος να δημιουργήσει ένα σενάριο λειτουργίας του συστήματος για οποιαδήποτε πολιτική παραγωγής και να εκτιμήσει μέσω δειγματοληψίας οικονομικούς δείκτες όπως κόστος παραγωγής, κόστος συντήρησης αποθεμάτων και καθυστερήσεις στη διεκπεραίωση παραγγελιών. Η εύρεση της βέλτιστης πολιτικής βασίζεται στην ανάλυση ευαισθησίας που υπολογίζει πώς ορισμένες μικρές αλλαγές στην πολιτική (όπως μια μικρή καθυστέρηση στην έναρξη ή στην περάτωση της παραγωγής ενός προϊόντος) θα επηρέαζαν το συνολικό κόστος. Η πληροφορία αυτή απαιτεί συνήθως πρόσθετες προσομοιώσεις για κάθε δυνατή αλλαγή των παραμέτρων ελέγχου. Στην εργασία αυτή αναπτύχθηκε ένα μοντέλο Markov για την περιγραφή της λειτουργίας του συστήματος και εφαρμόσθηκε η λεγόμενη τεχνική του ρολογιού αναφοράς (standard clock), η οποία επέτρεψε την ανάλυση ευαισθησίας με μια μόνο προσομοίωση και μικρό πρόσθετο υπολογιστικό κόστος. Με αριθμητικά αποτελέσματα αποτιμήθηκε η ακρίβεια και η ταχύτητα της προτεινόμενης μεθόδου έναντι της ανάλυσης ευαισθησίας που χρησιμοποιεί πολλές προσομοιώσεις.

ABSTRACT

A production system making different types of products in batches to avoid frequent product changes and set ups, which incur cost and delays, is considered. The system produces orders having random arrival times, random volumes and varieties of products. A production policy is a decision rule which sets the batch production start and completion times of every product when all of them are produced by the same machine. Using simulation, a system's production operational scenario can be generated under any given production policy and statistics of important economic indicators such as production cost, inventory holding cost and delays in filling customer orders can be collected. Finding an optimal production policy entails sensitivity analysis which calculates how specific small changes in a policy (e.g imposing a short delay in the start or completion time of some product type) would affect the total cost. Such information usually requires additional simulations to be carried out using all possible control parameter changes. This thesis developed a Markov model to describe the operation of the system and simultaneously applies the so-called, standard clock technique, which enabled a sensitivity analysis to be performed during a single simulation involving only the original policy at a small additional computational cost. A number of numerical experiments were carried out to investigate the accuracy of the proposed method and its speed against a standard sensitivity analysis that requires several simulation runs.

CONTENTS

1 INTRODUCTION

1.1 Aim and Scope of the Thesis

A production system is considered that has a machine which manufactures n different types of products in batches. The items produced either fill orders that have arrived and are still pending or they are stored in the warehouse to satisfy future demand. Each arriving order may contain different types of products j in varying quantities $Q(j)$, depending on the needs of the corresponding customer. Furthermore, we assume that the following quantities are random variables:

- customer order interarrival times
- quantity of each product type in an order
- machine processing times
- and setup times required for the machine to switch to a different product type

The system operation is controlled by a production policy which determines which product type is to be produced next by the machine or whether the machine should be kept idle. The type of control that is applied to multi-product batch production systems is known as lot scheduling.

The objective of this thesis is: a) to apply a commonly used lot scheduling policy characterized by a set of parameters and use sensitivity analysis to estimate the effect of a change in each parameter on important performance metrics such as production cost, net inventory of n product types, and order fulfillment delay b) to use the Standard Clock Technique for performance analysis via simulation of all production policies resulting from every possible parameter change by carrying out a single simulation run.

Computer simulation generates a scenario of events, such as arrivals and departures of orders, machine setup completions, and batch productions over a given operation period and collects statistical data on performance indices. The stochasticity of order arrival times, the quantities contained in each order, and the machine processing and setup times are modeled through random numbers generated by the computer.

Sensitivity analysis provides information regarding the incremental costs or savings associated with a change in a control parameter and can be used for optimal parameter selection. A standard clock and common random numbers are used for all parameter variations to calculate the average performance indices for each combination of control parameters.

1.2 Description of the Production System

1.2.1 Production Systems and (s, S) Parameters

A scheduling policy is considered which uses two parameters, the safety stock $s(j)$ and the maximum stock $S(j)$, for product types $j = 1, \dots, n$. The safety stock $s(j)$ represents the threshold of the lowest acceptable net inventory for product type j. If the net inventory falls below the safety stock $s(j)$, a production order will be issued. The maximum stock $S(j)$ represents the maximum inventory of a product type j that can be produced and simultaneously the production target. If the net inventory is less than the safety stock $s(j)$, a quantity must be produced such that the inventory after production equals the maximum stock $S(j)$. In each simulation, one of the two parameters $s(j)$, $S(j)$, is increased by 1 unit for a particular product type j, while for all other products $k \neq j$ the policy parameters $s(k), S(k)$ remain at their previous values. This change causes the system to operate in a different manner from the original system. The calculation of performance improvement or performance degradation resulting from this change, known as sensitivity analysis, is performed by simulating simultaneously all the systems resulting from the unit increment of each control parameter. The use of simultaneous simulation saves computations and ensures a fair comparison among systems.

Sensitivity analysis is combined with a search algorithm to find the best combination of parameters. The primary step of sensitivity analysis is the simulation of the base system with initial values $s(j)$, $S(j)$ and the initialization of a procedure that changes the parameter values by 1 at a time and runs another simulation to evaluate the performance of the new parameter combinations. Each such change signifies the existence of a different system from the base one. The values $s(j), S(j)$ are common and serve as the starting point for all product types $j = 1, \dots n$ and are provided initially by the program user. The different systems process the same product types $j = 1, \dots, n$.

The systems that are simulated are:

A) The base system (1)

B) n systems in which only the parameter $S(j)$ of product *j*, one at a time, is increased by one

C) n systems in which only the parameter $s(j)$ of product *j*, one at a time, is increased by one.

In total, $1 + n + n = 2n + 1$ systems are simulated simultaneously. The table below is an example of the above description with the number of different product types $n = 2$. Therefore, there are $2n + 1 = 5$ different systems.

Table 1.1:Simulation systems parameter settings (s, S) for n = 2 products

System sys	Parameter change	Value of $s(1)$	Value of $S(1)$	Value of $s(2)$	Value of $S(2)$
1	Base System	$s(1)$	$S(1)$	$s(2)$	$S(2)$
2	Increase $S(1)$ by 1	$s(1)$	$S(1) + 1$	$s(2)$	$S(2)$
3	Increase $S(2)$ by 1	$s(1)$	$S(1)$	$s(2)$	$S(2) + 1$
4	Increase $s(1)$ by 1	$s(1) + 1$	$S(1)$	$s(2)$	$S(2)$
5	Increase $s(2)$ by 1	$s(1)$	$S(1)$	$s(2) + 1$	$S(2)$

A computational code was developed that encodes each parameter combination into a system $sys = 1, \dots, 2n + 1$. The base system $sys = 1$ has the nominal parameter values $s(j)$ and $S(j)$ for all *j* = 1, …, *n*; each of the next n systems correspond to an increase of +1 in a parameter $S(j)$. The last n systems correspond to increases of +1 in a parameter $s(j)$ for each product type $j = 1, \dots, n$. This means that the changes $S(j) + 1$ correspond to systems $sys = 2, \dots, n + 1$, while the changes $s(j) + 1$ correspond to systems $sys = n + 2, \dots, 2n + 1$.

1.2.2 Cost and Rate Parameters with Fixed Initial Values

The production system, besides the starting values $s(j), S(j)$ which define the initial parameters of the lot scheduling policy, also has several other fixed parameters defined by the user. These are stored in a data file named StClock_LotSize_in.txt, which is read by the program at the beginning of the simulation process. These parameters are as follows:

- n : Total number of distinct product types
- $nevents$: Total number of events that are simulated
- $lamda\ or\ (\lambda)$: Mean arrival rate of customer orders (mean number of arriving orders per time unit)
- $v(j)$: Mean setup rate for product type j (mean number of setups for *j* that can be completed per time unit, provided that the machine is set up)
- $mu(j)$: Mean production rate for product type j (mean number of type *j* items that can be produced per time unit, provided that the machine is producing this type)
- $A(j)$: Unit production cost associated with product type *j*
- $B(j)$: Fixed setup cost incurred before the machine starts producing type j products
- $h(j)$: Unit inventory holding cost per unit of time and per unit of product type j
- $g(j)$: Unit backordering cost per unit of time and per unit of product type j
- $NQ(j)$: Number of packaging variants of product j (for example, a product for which the possible demand can be either 1, 5, or 10 and nothing else between or larger than these quantities has $NQ(j) = 3$)
- $QM(j, q)$: Quantity corresponding to packaging variant $q = 1, 2, \dots, NQ(j)$ of product type j (for the previous example we have given $QM(j, 1) = 1$, $QM(j, 2) = 5, QM(j, 3)$)
- $P(j, q)$: Probability that an order requests quantity variant q of product type j that is the customer requests exactly $QM(j, q)$ items.

Additionally, there are average times which are indirectly determined by the initial values of the rates that characterize them and are not affected by time.

- 1/mu(*j*) : Mean production time of a unit of product type j
- 1/*v*(*j*): Mean setup time for a unit of product type j
- $1/\lambda$: Mean inter-arrival time between successive orders

1.2.3 Variables (Time-Dependent Parameters)

At the beginning of the simulation, it is assumed that no events have occurred, the simulation clock (time) is set to 0. Other important variables used in simulation change dynamically over time. At time 0 these variables are initialized to 0 as well:

- $t = 0$: Time of the simulation. Simulation clock starts from time 0
- $ievent = 0$:
- $E(sys) = 0$: Number of pending orders. Initially = 0
- $totE(sys) = 0$: Total number of orders not served immediately
- $totOrders(sys) = 0$: Total number of orders placed
- $setup\_c(sys) = 0$: Total setup cost
- $prod\_c(sys) = 0$: Total production cost
- $inv\_c(sys) = 0$: Total inventory cost
- $delay\_c(sys) = 0$: Total delay cost
- $product\_in\_m(sys) = 0$: Product type on the machine. 0 means idle
- $npMustMake(sys) = 0$: Number of products with low inventory that must be produced
- $X(j)$: Net inventory level of product type j

The parameter $Q(j)$ is the random quantity of type j products requested in each new order. This is a random number, generated by a suitable random number generator and differs for each product j and for each new order arrival. When $Q(j)$ items are ordered, the system net inventory level $X(j)$ and shortage $RQ(j)$ for type j items increase

- $Q(j)$: Requested quantity of product type j
- $RQ(j)$ Quantity shortage for product type j in the corresponding order

The variable $RQ(j)$, through the program, specifies the shortage of product type j in a specific system sys and in a specific pending order $E(sys)$: $RQ(sys, E(sys), j)$. The net inventory $X(j)$ refers to the remaining available stock of product type j, from which the quantity $Q(j)$, required by all pending orders has been subtracted. If, at the time an order arrives, the warehouse has sufficient available stock for each requested item j, i.e. $X(j) \geq Q(j)$, the order will be fulfilled immediately. Otherwise, it will remain pending until all required quantities have been produced. In this case, the following relation holds: $RQ(j) = Q(j) - X(j) > 0$, meaning that the shortage of product j is positive. Each pending order served with priority over those that arrive after it. These variables are initialized with a value of 0 at the beginning of the simulation. The quantities $Q(j)$ requested by each customer are random variables and have probabilities $P(j, q)$.

$$P(j, q) = P(an\ order\ requests\ q\ units\ of\ product\ j) \tag{1.1}$$

for different values of $q = 0, 1, \dots, \mathrm{QM(j)}$, where $QM(j)$ is the maximum quantity of product j that can be requested. These probabilities and the maximum quantity $QM(j)$ are known and derived from statistical estimates.

The unit of measurement for all these times is the same. For mathematical simplicity, it is assumed that the production times of units, setup times, and the times between successive order arrivals are independent random variables with exponential distribution.

2 MODELING AND SIMULATION

2.1 Description of the Markov Model

2.1.1 Probability Theory

Probability theory deals with random experiments. Every random experiment has a set Ω of all possible outcomes ω which is called the certain event or sample space. For example, for the random experiment of tossing a fair die we have that $\Omega = \{1, 2, 3, 4, 5, 6\}$. A subset A of Ω is called an event. For example, in the die tossing experiment we may have *A* = even outcomes = {2, 4, 6}.

A random experiment has a set $\mathcal{F}$ of events *A* (subsets of Ω) for which probabilities *P*(*A*) must be defined. $\mathcal{F}$ is a set of sets. $\mathcal{F}$ must at least include the certain event Ω and the empty set $\emptyset$ which represents the impossible event. Also, if it contains an event *A* it must also contain its complement. For example, if the probability of event {2, 4, 6} is defined then the probability of the odd outcome {1, 3, 5} must also be defined. Finally, if the events $A_1, A_2, \ldots$ belong to $\mathcal{F}$ then $\mathcal{F}$ must also have their union $A_1 \cup A_2 \cup \ldots$ as an additional element.

The set $\mathcal{F}$ satisfying the above requirements is a σ-field (sigma-field).

The probability of any event *A* is a number *P*(*A*) in [0, 1]. The description of a random experiment is completed by requiring that the probabilities *P*(*A*) satisfy three fundamental properties which define the fundamental rules that ensure that probabilities are logical, mathematically correct, and practically applicable. The three fundamental properties, known as the axioms of probability, are the following:

A) Axiom 1 – non-negativity

For every event $A \in \mathcal{F}$

$$P(A) \geq 0 \tag{2.2}$$

The probability of an event is always a non-negative number.

B) Axiom 2 – The certain event Ω is assigned a probability value of 1:

$$P(\Omega) = 1 \tag{3.2}$$

The probability of certain events, that is, that some events occur, is always equal to 1.

C) Axiom 3 – Union of mutually exclusive events

If $A_1, A_2, A_3, \ldots$ is a countable sequence of mutually exclusive events such that $A_i \cap A_j = \emptyset$ for every $i \neq j$, and all belong to $\mathcal{F}$, then the probability of their union equals to the sum of their individual probabilities:

$$P(A_1 \cup A_2 \cup \dots) = P(A_1) + P(A_2) + \cdots \tag{2.4}$$

2.1.2 Stochastic Processes

A random variable X is defined as a function that assigns a number $X(\omega)$ to each outcome $\omega, \omega \in \Omega$, of a random experiment, provided it satisfies the following properties:

$$P[X(\omega) = -\infty] \triangleq P[\{\omega: X(\omega) = -\infty\}] = 0$$

That is, the probability that the random variable X takes the value $-\infty$, is defined as the probability of the set of outcomes ω for $X(\omega) = -\infty$, and this is 0.

$$P[X(\omega) = \infty] \triangleq P[\{\omega: X(\omega) = \infty\}] = 0$$

Similarly, the probability that the random variable X takes the value ∞, is defined as the probability of the set of outcomes ω for which $X(\omega) = \infty$, and this is 0. $X(\omega)$ is a measurable function, meaning the set $\{\omega: X(\omega) \leq x\}$ is an event for every real number x.

A stochastic process $\{X_t(\omega), t \in T\}$ is a family of random variables defined on a common probability space, with the parameter t being a real variable that denotes time. For each outcome $\omega \in \Omega$ of the random experiment, a function $X_t(\omega)$ is defined. If the set T is the real number line, then the process is called a continuous-time process. If the set T is a set of integers, then the process is called a discrete-time process [1].

2.1.3 Markov Property

The term discrete time means that the system remains in any given state for one time cycle and then moves to another state or remains in the same state for another cycle. The state X_i of the chain may evolve in continuous or discrete time and takes values from a measurable set of states, with probabilities that may depend on the current state, but are independent of previous states [1], [2].

Conditional Probability

The occurrence of event A, given that another event B takes place, is denoted by $A|B$. The corresponding conditional probability is defined as:

$$P(A|B) = \frac{P(AB)}{P(B)} \tag{2.5}$$

provided that $P(B) > 0$. The conditional probability satisfies the axioms of the conditional probability and therefore, equations (2.1)– (2.3) remain valid if we adjoin all probabilities with the same suffix "$|B$".

Two events *A* and *B* are independent if $P(A|B) = P(A)$; this also implies that $P(B|A) = P(B)$. An equivalent condition is:

$$P(AB) = P(A)P(B) \tag{2.6}$$

The probability of event A occurring given multiple events B, C is calculated from the following formulas:

$$P(A|BC\ldots) = \frac{P(ABC\ldots)}{P(BC\ldots)} \frac{\frac{P(ABC\ldots)}{P(C\ldots)}}{\frac{P(BC\ldots)}{P(C\ldots)}} = \frac{P(AB|C\ldots)}{P(B|C\ldots)} \tag{2.7}$$

Markov Property

To predict the future state in which the system will be at some future $t + \delta$, only need the most recent information $X(t)$. Knowledge of past states at previous times $\{X(u),$ where $\tau \leq u < t\}$ is of no use. That is, the future state of the system is independent of past states, if the current state is known.

$$P[X(t+\delta)|X(u) = known,\ \tau \leq u \leq t] = P[X(t+\delta)|X(t) = \text{known}] \tag{2.8}$$

If the state is known, say $X(t) = j$ at time t, the desired probability is the probability of moving from state j to state i in the interval $(t, t+\delta)$.

$$P[X(t+\delta) = i|X(t) = j] \tag{2.9}$$

A consequence of the Markov property is the following theorem:

The Markov chain $X(t)$ satisfies the Chapman Kolmogorov (C-K) linear equation: for every state $i = 1, 2, \ldots$ and every initial state $k = 1, 2, \ldots$ [1]

$$P[X(t+\delta) = i|X(0) = k] = \sum_j P[X(t+\delta) = i|X(t) = j]P[X(t) = j|X(0) = k] \tag{2.10}$$

2.2 Markov Chains and Introduction to Simulation

In this present thesis, a continuous-time stochastic system is analyzed. Since arrivals are random and each order includes various types of products in different quantities, there is no analytical solution. For this reason, the use of simulation is necessary. During the simulation, the operation of the system and the evolution of its internal processes over time are mimicked using a computer.

2.2.1 Analysis and Simulation of Discrete-Time Markov Chains

In discrete-time Markov chains, time steps are discrete, $t = 0, 1, 2, \ldots$, there is a time step from one to the next (e.g., $\delta = 1$), states (e.g., $1, 2, \ldots, n$), and known and unchanging transition probabilities.

$$a_{j,i} = P[X(t+1) = i | X(t) = j] \tag{2.11}$$

Substituting into the C-K equation (2.9) with $\delta = 1$, we obtain:

$$P[X(t+\delta) = i | X(0) = k] = \sum_j a_{j,i} P[X(t) = j | X(0) = k] \tag{2.12}$$

The C-K equations are transformed into:

$$P_i(t+1) = \sum_j a_{j,i} P_j(t)\,, t = 1, 2, \ldots \tag{2.13}$$

For simplicity, the initial state k will be omitted from the notation for each state i and time t:

$$P_i(t) = P[X(t) = i | X(0) = k] \tag{2.14}$$

It is assumed that the system is in steady state, where $t \to \infty$, and the probabilities $P_i = P_i(\infty) = P_i(\infty + 1)$ are equal and independent of time. For discrete-time chains, we have C-K equations, which form a system of linear equation for $= 1, 2, \ldots$.

$$P_i = \sum_i a_{j,i} P_i \tag{2.15}$$

The system has infinitely many solutions if a steady state exists, that is, if a solution exists. For a unique solution to exist, the probabilities must satisfy the three axioms of probability theory A, B, C.

The normalization equation is derived:

$$\sum_i P_i = 1 \;\; (P_i \geq 0) \tag{2.16}$$

In the steady state, the Markov chain transitions between states, even though their probabilities do not change as $t \to \infty$. We assume the corresponding limit exists and define the steady-state probabilities:

$$P_i = \lim_{t \to \infty} \frac{S_i(t)}{t} \tag{2.17}$$

- $S_i(t)$: total time the system has remained in state i during the interval $[0, t]$. It changes in a finite way as $t \to \infty$, provided there is entry into or exit from state i. If this change is divided by $t \to \infty$, the change in P_i will be zero.

Therefore, in the steady state, the states change but the probabilities remain.

Discrete-Time Markov Chain Simulation Algorithm

1. Reading Data

$i0$ = initial state, n = total number of states, $KSIM$ = simulation time horizon and total number of transitions to be simulated.

For $i = 1, \dots, n$

For $j = 1, \dots, n$

Read transition probability $a_{i,j}$

Next j

Next i

2. Initial Values

For $i = 1, \dots, n$

Initializing total time in state i is $S_i = 0$

Next i

Current state: $i = i0$

Current time: $t = 0$

3. Simulate KSIM transitions

$t = 1, \dots, KSIM$

Update total time spent in state i: $S_i = S_i + 1$

Next transition (k −th) transition: $t = t + 1$

At time t, a transition occurs from state i to a new state j.

Step 3a. Discrete Random Variable Generator j:

Generate a random number $U \sim U(0,1)$

Probability accumulator $x = 0$, initial $j = 0$

While $< U$:

$j = j + 1$

$x = x + a_{i,j}$

Repeat (while…)

At this point, the first j such that $x \geq U$ has been found.

Set new state: $i = j$

Next t

4. End of simulation at time t,after KSIM transitions

For $i = 1, \dots, n$

Estimate the probability $P_i = S_i/t$ and print the result and print the result

Next i [3]

2.2.2 Analysis and Simulation of Continuous-Time Markov Chains with Application to Lot Scheduling

In a continuous-time Markov chain the time t is a continuous variable on $[0, \infty]$. In this section we shall outline the corresponding equations and simulation algorithm for the case of batch production rather than a general Markov chain. The system is stochastic due to the randomness in machine processing and setup times, order arrivals, as well as the product types and quantities demanded in each customer order.

The state k of the system at some point is described by the variables: $\alpha_k . \beta_k, \gamma_k . \delta_k, \varepsilon_k$, where:

- $\alpha_k =$ the machine state (either idle or working to produce some product $j = 1, \dots, n$ or to prepare to start production of j)
- $\beta_k = (\beta_{1k}, \dots, \beta_{jk}, \dots, \beta_{nk}) =$ physical inventory of each product $j = 1, \dots, n$
- $\gamma_k = (\gamma_{1k}, \dots, \gamma_{jk}, \dots, \gamma_{nk}) =$ quantities of each product j that have been ordered but not yet delivered
- $\delta_k = (\delta_{1k}, \dots, \delta_{jk}, \dots, \delta_{nk}) =$ net inventory of each product j, $\delta_{jk} = \beta_{jk} - \gamma_{jk} =$ total shortage (if negative) or surplus (if positive), when the system is in state k
- $\varepsilon_k =$ set of detailed information for each pending order with the quantities of all products included in that order (ε_k contains more detailed information than γ_k)

Therefore, state k is a composite variable. The system is in one state, and when a new order arrives or when the work on the machine is completed, if the machine is not idle, it then moves to a new state. To find the next state we first observe that in state k the machine either operates producing a specific product, or prepares for a specific product, or is idle. Therefore, j is known when k is determined. We define λ_k as the average machine rate in state k.

$$\lambda_k = \begin{cases} 0 & if\ in\ state\ k\ the\ machine\ is\ idle \\ mu(j) & if\ in\ state\ k\ the\ machine\ produces\ j \\ v(j) & if\ in\ state\ k\ the\ machine\ prepares\ for\ j \end{cases}$$

A continuous-time Markov chain has the following properties:

- a set of states $i = 1, 2, \dots, n$
- average transition rates $\lambda_{k,l}$ from state k to $l \neq k$
- average exit rate $(\lambda + \lambda_k)$ from state k, $(\lambda + \lambda_k) = \sum_{l \neq k} \lambda_{k,l}$
- it is assumed that the interarrival times of orders, and the setup times are independent random variables with exponential distributions.

If the system is in state k, there exists the transition rate from k to l, λ_{kl}, such that:

$$P[X(t+\delta) = l | X(t) = k, \{X(\tau), \tau < t\}] = \lambda_{kl}\delta + o(\delta) \text{ , for } k \neq l \tag{18}$$

for all time intervals $[t, t + \delta]$.

It follows from the Markov property that such transitions depend only on the fact that $X(t) = k$ and are independent of the system history at times $\tau < t$. Furthermore, we assume for simplicity that $[X(t+\delta) = l | X(t) = k]$, depend on $\delta, l,$ k but are independent of time t. The

state transitions do not change over time, they remain stationary. Now two properties which permit the development of simple algorithms for simulating continuous-time Markov chains are presented.

For the sake of brevity, we only formulate these properties in the context of the particular Markov chain corresponding to the lot scheduling policy.

Property M1

A) If it is known at time t that $X(t) = \mathrm{k}$, the remaining time of stay in state l has an exponential distribution with mean $1/(\lambda + \lambda_k)$. The exit rate from state l is defined as:

$$(\lambda + \lambda_k) = \sum_{k \neq l} \lambda_{k,l} \tag{2.19}$$

B) Given that the system is in state k, its remaining time before it jumps to another state is independent of the time it has already spent in state k.

Property M2

A) When the chain leaves a state k, the probability of transitioning to l is $a_{k,l} = \lambda_{k,l}/(\lambda + \lambda_k)$
B) This probability is independent of the time the chain had remained in state k.

To estimate the parameters, one must first compute the mean holding times for all states, equate them with $1/(\lambda + \lambda_k)$, and estimate the rates $(\lambda + \lambda_k)$. Then, for each state k, the number of transitions to state l is counted and divided by the total number of transitions out of state k. The quotient is an estimate of the transition $a_{k,l}$. The simulation is based on the observation of transitions from one state to another. If the total time spent in each state is divided by the simulation (observation) time of the system, the state probability is obtained. By applying properties M1, M2 the new state of the system is selected, and the holding time in each state is computed. Assuming a continuous-time Markov chain with a finite number of states 1, 2, …, n, the system is simulated by applying properties $M1$ and $M2$ and the following:

Continuous-Time Markov Chain Lot Scheduling Simulation Algorithm

Read the data $\lambda, mu(j), v(j), j = 1, \dots, n$, and the production policy. Let $k0$ be the initial state. Initially, time $t = 0$, state $k = k0$, performance measures $= 0$. Repeat for some number of steps the following:

(i) Compute the sojourn time W_k in state k: sample W_k from an exponential distribution with mean $1/(\lambda + \lambda_k)$. (Property M1)

(ii) Make the necessary state changes by applying the production policy.

(iii) Compute sample estimates of the performance measures due to staying in k for time W_k

(iv) Advance time: $t = t + W_k$

(v) Determine the new state l to which the system will move using a random-state generator with known probabilities:

$$probability = \frac{(\lambda_{k,l})}{\lambda + \lambda_k}$$

for each possible transition $k \to l$. (Property M2)

(vi) Update k: $k = l$

Repeat.

The simulation has been completed. The total cost is computed by individual performance measures.

Let $P_k(t)$ be the probability that at time t the system is in state k. If these probabilities are known, various performance measures of the system can be calculated for each time interval $[t, t + dt]$. For example, the average inventory of j in $[t, t + dt]$ is:

$$\sum_{\text{for all states } k} \beta_{jk} P_k(t) \quad dt$$

β_{jk} is the physical inventory of items j when the state is k, and the summation considers all possible states k in which the system can be at time t, such as:

$$\sum_{for\ all\ states\ k} P_k(t) = 1 \tag{20}$$

This equation is the normalization equation. The transition rates from l to k are also computed from $\lambda, mu(j), v(j)$, where:

$\lambda =$ mean order arrival rate, the same for all states $=$ (1/mean interarrival time between successive orders)

$mu(j) =$ mean completion rate of machine work, given that in state k the machine produces $j =$ (1/mean production time of one item j)

$v(j) =$ mean completion rate of machine work, given that in state k the machine prepares to start production of $j =$ (1/mean setup time of the machine to start a production cycle of products j)

From now on, $P_k(t)$, will be written as P_k. The Chapman-Kolmogorov (C-K) equation becomes:

$$\frac{dP_k}{dt} = -(\lambda + \lambda_\kappa)P_k + \sum_{l \neq \mathrm{k}} \lambda_{l,\mathrm{k}} P_l \tag{21}$$

From the solution of the C-K and normalization equations, the unknown probabilities P_k are obtained. Instead of analytically determining the P_k, the simulation uses certain properties of continuous-time Markov chains, and the algorithm of the present thesis is developed.

Markov Chain Continuous-Time Simulation Algorithm

1. Reading Data – 2. Initial Values

Read $i0$ = initial state, n = total number of states, $KSIM$= number of transitions to simulate

For $i = 1, \dots, n$

Initially, the exit rate from state i is set to zero: Λ_i=0, and the probability estimates are also set to zero: $P_i = 0$

For $j = 1, \dots, n$

If $j \neq i$ then

Read $\lambda_{i,j}$

$\Lambda_i = \Lambda_i + \lambda_{i,j}$

Else

Set $\lambda_{i,j} = 0$

End If

Next j

Next i

For $i = 1, \dots, n$

For $j = 1, \dots, n$

Transition probability: $a_{i,j} = \lambda_{i,j}/\Lambda_i$

Next j

Next i

Current state $i = i0$

Current time $t = 0$

3. Execute KSIM transitions

For $k = 1, \dots, KSIM$

Generate a uniform random number U

The exponential holding time in state i is obtained from the generator:

$$W = -(lnU)/\Lambda_i$$

Update total holding time in state i: $S_i = S_i + W$

Time of the next $(k - th)$ transition: $t = t + W$

At time t, a transition from i to a new state j will occur. j is found executing Step 3^{α} of a discrete-time Markov chain algorithm.

The new state is set: $i = j$

Next k

4. End of simulation at time t, after $KSIM$ transitions

For $i = 1, \ldots, n$
Print probability $P_i = S_i/t$
Output the results
Next i [3]

2.3 Application of Little's Law

The use of Little's Law is important in production systems because it can help evaluate the efficiency of the system and improve the production policy. Little's Law can be applied, under certain conditions, to any production system as well as to individual parts of a larger system. Little's formula is:

$$N = \lambda T \tag{2.22}$$

N : The average number (of orders, times) in the system

λ $(lambda)$: The average arrival rate of customers on the system, the average departure rate from the servers (machines), or the average production rate

T : The average time spent on the system

Little's formula is used to find either *N* or *T* when the other two variable in (2.21) are known. The conditions for the validity of Little's law are:

Condition L1: The quantities λ, N, T are calculated in the steady state, that is, they are averages over infinite observation time and infinite customer arrivals and are finite numbers.

Condition L2: The following hold with probability 1 :

A) In finite time, we have a finite (i.e., $< \infty$) number of arrivals

B) The time between successive arrivals is finite

C) The duration of each service is finite and, finally

D) The queuing system is stable, i.e., at every moment t, the inventory $n(t)$ of the system is finite [3].

3 PRODUCTION POLICIES

A production policy in a stochastic production system is defined as the set of rules determining when and how much of each product should be produced, aiming to efficiently meet demand under uncertainty. Production decisions are based on the current state of the system, such as inventory levels, backlogged orders, or machine availability, and are designed to maintain sufficient stock levels, minimize inventory holding costs, and avoid shortages. Some researchers who extended the (s, S) policy to multiple products include Paul H. Zipkin who developed models for the design and control of stochastic, multi-product batch production systems, combining inventory and queueing analysis to estimate operating costs and support design decisions. Similarly, Herbert Scarf analyzed the dynamic inventory problem with linear holding and shortage costs and demonstrated that the optimal policy in each period is of the (s, S) type. These studies provide a theoretical foundation for the application of the (s, S) policy in systems with multiple products, where the production occurs in batches and inventory and order management is stochastic [4], [5].

3.1 The (s, S) Policy

The production policy applied in this specific production system is based on the (s, S) inventory policy, which aims to maintain a safe inventory level for each different product type j. According to this policy, when the net inventory of a product falls below the lower threshold s, known as the safety stock, a production order is issued to replenish the inventory up to the upper limit S, which represents the maximum inventory. The production policy is applied separately for each product type j. The n different product types j generate the number of different production systems (different production policies) according to the formula $2n + 1$. These systems differ from each other due to small variations in the s and S parameters. These parameters, starting from the base system, are increased by $+1$ each time. Each production system is considered to have its own machine, operates independently, and manages demand differently compared to the others. The main events that determine the operation of the policy are order arrivals and machine events. A machine event can be either the production of a product type or the preparation for its production.

Order arrivals are uniform across all systems. Then, each system handles them differently. Initially, it is checked whether the existing inventory fully or partially covers the demand for the order, and the corresponding reduction is applied. If a shortage arises that did not exist before, a production order is issued for the product types that need to be produced, and the product type is added to the production list. If, during this process, the type is not added to the production list, it means that the order does not require that specific product. Next, it is checked whether the order can be fully fulfilled or if it needs to be placed on the backlog list.

In the final step, it is checked whether production should start on the machine, which had been idle until that moment.

During the machine production event, it is assumed that a batch of the product has been produced, and the system's inventory is increased. Then, the inventory is reduced again to meet the demand generated by the pending orders. It is checked whether the orders have been fully satisfied and removed from the backlog list. Finally, it is verified whether the inventory has reached the target S, ensuring compliance with the (s, S) policy.

3.2 Batch Production Approach in Multi-Product Production Systems

Each production system, through its machine, manages order arrivals, batch production, and preparation to produce a batch of product type j. Batch production aims to reduce the delay associated with the setup required to switch to another product type. Orders that cannot be fulfilled immediately are placed on a backlog list and are served according to priority. The oldest order on the list has the highest priority.

At the same time, when the production of a batch of product j is completed, the backlog list is checked for orders requiring that specific product type. If the inventory is sufficient to satisfy not only the highest-priority order but also the next one, and if the latter does not require another product type to be fully fulfilled, then this order will be completed earlier than the one that was originally first on the backlog list. In this way, the system effectively maintains priority while simultaneously utilizing inventory to minimize delays and maximize order fulfillment. In other words, there is flexible management.

In the case that a production order is issued while the machine is busy with another product type, its execution waits for its turn. It is necessary to complete the production of the entire batch before moving on to the next type. When a product type change is required, preparation for the new type is necessary; if production continues with the same product type, no preparation is needed. In this way, the multi-product system ensures effective production management, maintains inventory levels, and maximizes order fulfillment.

4 SENSITIVITY ANALYSIS

4.1 Concept and Role of Sensitivity Analysis

Simulation is used in the optimization of both queueing systems and Markov chains. The goal of optimization is to determine the values of certain design or control parameters of the system that maximizes a specific performance or utility measure. Such problems are always subject to constraints. To solve these problems using simulation, iterative methods are employed. Starting from arbitrary initial values for the design or control parameters of the system, the optimal values are iterative approached. In each iteration, the system parameters are fixed, and the system's performance measure is evaluated. The next step is to find which parameters must be changed to improve performance further. This information arises from sensitivity analysis, which, for differentiable functions, is equivalent to estimating the partial derivative of the performance measure with respect to each parameter. Sensitivity analysis indicates how a change in the value of each parameter affects system performance. If the change improves the performance, then the parameter is changed in the direction (increase/decrease) that points to the improvement. Each subsequent iteration further improves the solution until the performance reaches its maximum possible value, at which point iterations terminate [3].

4.2 Use of Common Random Numbers

To simulate stochastic systems, we use random numbers. To compare the performance of different control policies we use common random numbers and generate

(i) the same random orders, order arrival times, and product quantities requested and

(ii) the same random processing and setup times for all policies.

This ensures that all policies are fairly compared. In this work we satisfy requirement (i) above and partly requirement (ii).

The generation of a single random number requires considerable computational time. The use of common random numbers for different policies saves computations because one number is used in several simulations. The only problem is that different policies require the same random number at different times during a simulation. For example, the time when the tenth product j is produced differs among different policies and therefore the corresponding random number must be generated at different times rather than synchronously. One way to avoid repeating the generation of random numbers in each simulation is to store the common random numbers and reuse them. However, the computer RAM memory can only store a few million random numbers whereas using the hard disk is time consuming. The only possibility is to find a method that permits simulating all systems synchronously. For Markov chains, this is possible by applying the Standard Clock technique and an additional Markov property which is discussed next.

4.3 Sensitivity Analysis Using the Standard Clock Technique

The Standard Clock technique (SC technique), originally developed by P. Vakili in 1991, is an efficient simulation method for stochastic systems, particularly useful for discrete event systems and continuous-time Markov chains (CTMC) in multiprocessor or parallel computing environments. This method is applicable to all continuous-time Markov chains. The standard clock technique is a simpler method of sensitivity analysis. During the simulation of the continuous-time Markov chain, the most computational demanding arithmetic operations are those involving the random number generators. Sensitivity analysis may require executing multiple simulations with different sets of states. The standard clock technique uses another property of Markov chains, that allows the simultaneous simulation of the original system and any other system, for example different $(s(j), S(j))$ policies, using the same result from the residence time generator, even though the current state of the original system differs from the states of the other systems being simulated simultaneously. In this way, repeated computations for the generation of random numbers are avoided, but the simulation of virtual departures is required. Besides properties $M1$ and $M2$, this result is a consequence of a continuous-time Markov property described below [6].

Property M3

Every Markov chain with transition rates $(\lambda + \lambda_k)$ is independent to another chain with equal or greater rates $(\lambda + M)$ for any $M > \lambda_k$. We write

$$\lambda + M = \lambda + \lambda_k + (M - \lambda_\kappa) \tag{23}$$

For each state k, the difference $(M - \lambda_k)$ correspond to a virtual departure from k and immediate return to k while λ and λ_k correspond to real transitions to other states $l \neq k$ due to either an order arrival (λ) or a machine completion task (λ_k). Consider a continuous-time Markov chain with n states and average exit rate from state k, $\lambda + \lambda_\kappa$. The evolution of the chain depends on the state probabilities P_k, which satisfy the Chapman-Kolmogorov (C-K) equations. This equation gives the rate of change of the probability of state k as the difference between the rate of decrease of the probability due to departures from k and the rate of increase due to transitions into k.

The equation (2.20)

$$\frac{dP_k}{dt} = -(\lambda + \lambda_\kappa)P_k + \sum_{l \neq \mathrm{k}} \lambda_{l,\mathrm{k}} P_l$$

changes by adding and subtracting MP_k, where M is the maximum of all $mu(j), v(j)$: $M = \max(mu(1), v(1), \dots, mu(n), v(n))$. A new equation, equivalent to the original, results

$$\frac{dP_k}{dt} = -(\lambda + \lambda_\kappa)P_k - MP_k + MP_k + \sum_{l \neq k} \lambda_{l,k} P_l = -(\lambda + M - M + \lambda_\kappa)P_k + \sum_{l \neq k} \lambda_{l,k} P_l \Rightarrow$$

$$\frac{dP_k}{dt} = -(\lambda + M)P_k + (M - \lambda_k)P_k + \sum_{all\ states\ l} \lambda_{l,k} P_k \tag{24}$$

In other words, it is assumed that from state k a departure occurs at the higher rate $\lambda + M$ instead of $\lambda + \lambda_k$, but the departure can be virtual since with rate $M - \lambda_\kappa$ a return to the same state occurs. This virtual departure and return to k may initially seem unnecessary, but it allows the simultaneous simulation of many systems, each of which may be in different state, because the term $\lambda + M$ is independent of k. The solutions of the equations (4.2), (2.20) are the same. The second corresponds to a new Markov chain with the same distribution (equivalent or stochastically equal) as the original chain. If the rates $mu(j), v(j)$ have similar values and the machine is rarely idle (i.e., nearly fully utilized), which is typically the case in practice, then the terms $M - \lambda_k$ are small, virtual transitions occur rarely, and the method outperforms in speed the one that performs many simulations with repeated random number generations (but without virtual events).

5 MATLAB CODE DESCRIPTION AND ANALYSIS

5.1 Structure and Execution Flow

The algorithm is written and executed in the Matlab programming language. It is based on the theory presented in the previous chapters, given some initial parameters, to calculate: the number of systems being simulated (2*n+1), the total number of orders placed (TotOrders), those were not completed due to the termination of the simulation after the predefined number of events (Backorders), average setup cost rate (SetCstRt), average inventory cost rate (InvCstRt), average delay cost rate (DelCstRt), average production cost rate (PrdCstRt), average total cost rate (TotCstRt). Taking the above into account, the objective of the simulation is achieved, which is to identify the optimal policy. For its operation an input file input.txt is required, the name of it is StClock_LotSize_in.txt, in which the basic operating parameters of a production system are recorded. Additionally, the results are displayed in the command window (CW), but are also saved in a results.txt, SC_LotSize_out_10test_.txt file. The basic stages that occur during the simulation are:

- data input from the input.txt file and variable initialization
- finding the time of the next event through the generation of a random number
- determining the type of the next event
- the execution algorithm of the next event
- computation of performance metrics and optimal policy determination

5.1.1 Reading Data and Elements from the File input.txt

The algorithm starts by reading file

StClock_LotSize_in.txt

which contains the system parameters:

- 10test_: The end of the name of the output file SC_LotSize_out_10test_.txt
- n : Total number of distinct product types
- $seed0$: seed which generates random numbers
- $nevents$: number of events that will be simulated
- $lambda$ (or just $\boldsymbol{\lambda}$) : Mean arrival rate of customer orders per unit of time
- $startprint$: from that point and after, the basic variables will be visible
- $sensitivity$: 0, for simulation of the basic system, 1, for simulation of all systems
- $s(j)$: safety stock
- $S(j)$: maximum inventory level
- $v(j)$: Mean setup rate for product type j
- $mu(j)$: Mean production rate for product type j
- $A(j)$: Unit production cost associated with product type j
- $B(j)$: Fixed setup cost incurred for product type j
- $h(j)$: Unit inventory holding cost per unit of tie and per unit of product type j

- $g(j)$: Unit backordering cost per unit of time and per unit of product type j
- $NQ(j)$: Number of packaging variants of product j (for example, a product for which the possible demand can be either 1, or 5, or 10 and nothing else between or larger than these quantities have $NQ(j) = 3$)
- $QM(j,q)$: Quantity corresponding to packaging variant $q = 1, 2, \ldots, NQ(j)$ of product type j (for the previous example we have given $QM(j,1) = 1$, $QM(j,2) = 5, QM(j,3)$)
- $P(j,q)$: Probability that an order requests quantity q of product type j

The data file for $n = 10$, has the following format:

Table 5.1.: StClock_LotSize_in.txt (input file)

```
 # Example with 10 products (Simulation parameters of a multi-item lot sizing production system operating under (sj, Sj) policies with random demand). Any line starting with # is a comment and must remain, even if it's just #: <space>.

# (this is the second line of the file) Output file name suffix (it will be SC_LotSize_out_10test_.txt).

10test_

#n=Number of products   seed0=Seed of random generator (>=0)  nevents=Number of events to simulate: arrivals + productions + setups + pseudo-events (suggested >1000000) lambda=Average order arrival rate (per time unit)   start_print=From which event onward will the simulation model's key variables be printed on screen (For debugging use < nevents, for speed and final results only use > nevents).  sensitivity=0 for simulating only the base system, =1 to also simulate systems where some sj or Sj is +1 compared to the base one

10 1 100000000 0.03 1000000000 1

#For j=1,2,...,n provide base values:  s(j) (can be <0)   S(j) (must be >=0 and >=s(j))  v(j)=average setup rate for j (inverse of average setup time)   mu(j)=average production rate for j , A(j)=unit production cost of j, B(j)=setup cost for j, h(j)=unit holding cost (per time unit and per unit of product j), g(j)=unit backorder cost (per time unit and per unit of product j, because if an order has many products the backorder cost will be large)

-1 15 1 10 1 10 0.1 0.05

-1 15 1 10 1 10 0.1 0.05

-1 15 1 10 1 10 0.1 0.05

-1 15 1 10 1 10 0.1 0.05

-1 15 1 10 1 10 0.1 0.05

-1 15 1 10 1 10 0.1 0.05

-1 15 1 10 1 10 0.1 0.05

-1 15 1 10 1 10 0.1 0.05

-1 15 1 10 1 10 0.1 0.05

-1 15 1 10 1 10 0.1 0.05

#For j=1,2,...,n provide the following:

# (j=1) NQ(j)=number of possible quantities that may be ordered (number of packaging variants of product j)

 2
```

```
# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5

# (j=2) NQ(j)=number of possible quantities that may be ordered (number of packaging
variants of product j)

 2

# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5

# (j=3) NQ(j)=number of possible quantities that may be ordered (number of packaging
variants of product j)

 2

# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5

# (j=4) NQ(j)=number of possible quantities that may be ordered (number of packaging
variants of product j)

 2

# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5

# (j=5) NQ(j)=number of possible quantities that may be ordered (number of packaging
variants of product j)

 2

# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5

# (j=6) NQ(j)=number of possible quantities that may be ordered (number of packaging
variants of product j)

 2

# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
```

```
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5

# (j=7) NQ(j)=number of possible quantities that may be ordered (number of packaging
variants of product j)

 2

# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5

# (j=8) NQ(j)=number of possible quantities that may be ordered (number of packaging
variants of product j)

 2

# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5

# (j=9) NQ(j)=number of possible quantities that may be ordered (number of packaging
variants of product j)

 2

# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5

# (j=10) NQ(j)=number of possible quantities that may be ordered (number of packaging
variants of product j)

 2

# For q=1,..., NQ(j) provide QM(j,q)=packaging quantity, and the corresponding
probabilities P(j,q)=P[an order requires product j in quantity QM(j,q)], such that
P(j,1)+P(j,2)+...+ P(j,NQ(j)) <= 1. Use < 1 if an order may NOT include product j at
all

  1 0.3

  5 0.5
```

5.1.2 Implementation of (s,S) Production Policy

Next, the system dynamic parameters are initialized. An internal clock is invoked to start the CPU timing, using the command tic, ($t = 0$), the system state is initialized $k = k_0$ and the following variables are set to 0:

- $E(sys) = 0$: Number of pending orders (currently). Initially = 0
- $totE(sys) = 0$: Total number of orders not immediately served
- $totOrders(sys) = 0$: Total number of orders placed
- $setup\_c(sys) = 0$: Total setup cost
- $prod\_c(sys) = 0$: Total production cost
- $inv\_c(sys) = 0$: Total inventory cost
- $delay\_c(sys) = 0$: Total cost of delivery delays
- $status(sys) = 0$
- $product\_in\_m(sys) = 0$: Product type currently on the machine. 0 means idle
- $np\_MustMake(sys) = 0$: Number of Low-inventory products that must be produced

At the start of the algorithm, the condition for initiating production is defined based on the production policy, along with the combination of variables that will indicate the events of order arrival, production, and machine setup for the start of production.

$if\ X(sys, j) < s(sys, j)$

$np\_Must\_Make(sys) = np\_Must\_Make(sys) + 1;$

$PL\big(sys, np\_Must\_Make(sys)\big) = j;$

$MustMake(sys, j) = 1;$

$if\ product\_in\_m(sys) = 0$

$\%\ product\_in\_m > 0\ and\ status = 0\ together,$

$mean\ machine\ setup\ to\ start\ production\ of\ product\_in\_m$

$\%\ product\_in\_m > 0\ and\ status = 1\ together,$

$mean\ production\ of\ product\_in\_m$

$\%product\_in\_m = 0\ and\ any\ status\ mean\ full\ machine\ idle$

$product\_in\_m(sys) = j; status(sys) = 0;$

$setupc(sys) = setupc(sys) + B(j);$

end

end

5.2 Simulation of Continuous-Time Markov Chain (CTMC)

Repeat the following:

5.2.1 Calculation of the Time Spent in State k

Section 3. Find and execute next event in each system, sys, the time step for the next possible event is found and is added to the total simulation time t. This time step for state k, W_k, represents the frequency of the standard clock ticks and coincides with the sojourn time in

state k. According to the standard clock technique, although each system has a different k, the W_k is the same for all states and is denoted by W, because the determining factor in its calculation is the same departure rate from state k, $(\lambda + M)$. This specific step of simulation is common to all systems being simulated. In the code, this term corresponds to:

$$(lambda + \max[mu(product\_in\_m(sys)), v(product\_in\_m(sys))]) = (lambda + mean\_rate\_all\_evts) \tag{25}$$

Next:

$$mean\_rate\_all\_evts = mean\_rate\_all\_evts + lambda \tag{26}$$

Finally,

$$W = 1/mean\_rate\_all\_evts \tag{27}$$

From the data file, specific values have been defined for the order arrival rate, the production rate, and the machine setup rate for production, respectively:

$$lambda = 0.03, \qquad mu(j) = 10, \qquad v(j) = 1 \tag{28}$$

Therefore, the sojourn time is the same for all states k and is equal to:

$$W = \frac{1}{meanrateallevts} = \frac{1}{(0.03+\max[10,1])} = \frac{1}{0.03+10} = \frac{1}{10.03} \approx 0.0997 \text{ time units} \tag{29.5}$$

5.2.2 Necessary State Transitions under the Production Policy

This step and the following ones are repeated until the end of the simulation for each system separately within the same loop. The two possible states of the production system are: arrival of a new order, $arrival == 1$, and machine event, $arrival == 0$. In the case of a machine event, the possible outcomes are production of a new item, preparation for production, and the virtual event. In the case of an arrival, the application of the production policy is examined for each system separately, after the common arrival of the new order.

A) It is assumed that the order is valid and is added to the list of pending orders:

$$valid == 1 \ ; \ E(sys) = E(sys) + 1 \tag{30}$$

B) The case of immediate service is examined:

$$immediate\_service = 1 \tag{31}$$

- If the order contains product type j, there will be a production cost:

$$if\ Q(j) > 0\ ;\ prod\_c(sys) = prod\_c(sys) + A(j) * Q(j) \tag{32}$$

- If the inventory for product j is sufficient, the shortage is zero. If it is positive but not enough, there will be a shortage. If there is no inventory, the shortage equals the quantity requested by the order. Accordingly:

$$if\ X(sys, j) \geq Q(j)\ ; RQ(sys, E(sys), j) = 0 \tag{33}$$

$$if\ X(sys, j) > 0\ ; RQ(sys, E(sys), j) = Q(j) - X(sys, j) \tag{34}$$

$$RQ(sys, E(sys), j) = Q(j) \tag{35}$$

- If there is still a shortage, the order is not served immediately:

$$if\ RQ(sys, E(sys), j) > 0\ ; immediate\_service = 0 \tag{36}$$

- The net inventory of product type j is reduced:

$$\mathrm{X}(sys, j) = X(sys, j) - Q(j) \tag{37}$$

C) Check whether a production order should be issued when none existed before:

$$if\ X(sys, j) < s(sys, j)\ \&\&\ MustMake(sys, j) == 0 \tag{38}$$

- A production order is issued, and the product type is stored in the production list. Otherwise, product j is not included in the order:

$$MustMake(sys, j) = 1\ ;\ PL\big(sys, np\_MustMake(sys)\big) = j \tag{39}$$

$$RQ(sys, E(sys), j) = 0; \tag{40}$$

D) Check whether the order is served without delay and removed from the list of pending orders:

$$if\ immediate\_service == 1\ ; E(sys) = E(sys) - 1 \tag{41}$$

- Computation of inventory cost for the units that are dispatched now:

$$inv\_c(sys) = inv\_c(sys) + h(j) * Q(j) * t \tag{42}$$

- If the order cannot be served immediately, it is added to the list of pending orders, and the delay cost is computed:

$$totE(sys) = totE(sys) + 1\ ;\ delay\_c(sys) = delay\_c(sys) - g(j)Q(j) * t \tag{43}$$

E) Check whether the machine was idle but must start working:

$$if\ product\_in\_m(sys) == 0\ \&\&\ np\_MustMake(sys) > 0 \tag{44}$$

- Preparation to produce product j takes place, and the setup cost is computed:

$$j = PL(sys, 1);\ product\_in\_m(sys) = j;\ status(sys) = 0 \tag{45}$$

$$setup\_c(sys) = setup\_c(sys) + B(j) \tag{46}$$

Second case: machine event, end of part production

A) The net inventory of product j is increased and the inventory holding cost is computed:

$$X(sys, j) = X(sys, j) + 1;\ inv\_c(sys) = inv\_c(sys) - h(j) * t \tag{47}$$

B) The order that requested product j is searched for:

$$while\ demand\_for\_j == 0\ \&\&\ i < E(sys);\ i = i + 1 \tag{48}$$

- The order is found, and the outstanding demand for product j in order i is reduced:

$$RQ(sys, i, j) = RQ(sys, i, j) - 1 \tag{49}$$

- A check is performed to see if all products of order i have been completed; it is then assumed that the order is fully processed. Otherwise, the order remains pending. Accordingly:

$$if\ RQ(sys, i, j) == 0\ ; fully\_satisfied\_order = 1 \tag{50}$$

$$if\ RQ(sys, i, j_1) > 0\ ; fully\_satisfied\_order = 0 \tag{517}$$

- If the order is fully satisfied, the delay and inventory costs for the quantity of product j in the order are computed, and the number of pending orders is reduced:

$$inv\_c(sys) = inv\_c(sys) + h(j_1) * originalQ(sys, i, j_1) * t \tag{528}$$

$$delay\_c(sys) = delay\_c(sys) + g(j_1) * originalQ(sys, i, j_1) * t \quad (539)$$

$$E(sys) = E(sys) - 1 \quad (54)$$

C) A check is performed to see whether the inventory has reached the target $S(j)$, and a no-production command is issued for product type j:

$$if\ X(sys, j) == S(sys, j)\ ; MustMake(sys, j) = 0 \quad (55)$$

- The production list is updated:

$$PL(sys, k-1) = PL(sys, k) \quad (56)$$

D) Preparation for production begins, and the setup cost of the new product type is computed:

$$setup\_c(sys) = setup\_c(sys) + B(j) \quad (57)$$

5.2.3 Sample Estimates of Performance Measures for State k

The sample estimates record the contribution of each state k to the accumulation of the performance measures throughout the simulation. During each state, the cumulative sum of the previous cost quantities is computed, so that at every moment in time the total accumulated value from the beginning of the simulation is available. The computation of the sample estimates for any state k is carried out using the following formulas:

A) When the event is the arrival of a new order:

- For orders that have just been served and leave the system, Little's Law is applied. The inventory cost is equal to:

$$inv\_c(sys) = inv\_c(sys) + h(j) * Q(j) * t \quad (58)$$

- For orders that have not been fully satisfied and enter the waiting line, Little's law is applied. The delay cost is equal to:

$$delay\_c(sys) = delay\_c(sys) - g(j) * Q(j) * t \quad (59)$$

B) When the event is a machine event:

- Upon completion of the production of a part, inventory cost is added. The part is stored at the current, most recent, time:

$$invc(sys) = invc(sys) - h(j) * t \tag{60}$$

- For the units of each product j_1, which were missing from the order and kept it pending, but were eventually produced, inventory and delay costs are added, and Little's law is applied:

$$inv\_c(sys) = inv\_c(sys) + h(j_1) * originalQ(sys, i, j_1) * t; \tag{61}$$

$$delay\_c(sys) = delay\_c(sys) + g(j_1) * originalQ(sys, i, j_1) * t \tag{62}$$

The term $Q(j)$ denotes the quantity of product j requested by the new order, while the term $originalQ(sys, i, j_1)$ denotes the total quantity requested by order i at its arrival. Finally, the total inventory cost for the net inventory of each product j of the system, and the total delay and inventory costs for the units concerning the orders not yet completed, are computed.

$$inv\_c(sys) = inv\_c(sys) + h(j) * X(sys, j) * t \tag{63}$$

$$inv\_c(sys) = inv\_c(sys) + h(j) * \big(originalQ(sys, i, j) - RQ(sys, i, j)\big) * t; \tag{64}$$

$$delayc(sys) = delayc(sys) + g(j) * originalQ(sys, i, j) * t; \tag{65}$$

The difference $originalQ(sys, i, j) - RQ(sys, i, j)$, represents the items j of order i that remain in storage while waiting for i to be completed.

The simulation time is increased: $t = t + 1/mean\_rate\_all\_evts$.

5.2.4 Randomized Determination of Next System State

In section 3a. Find next event, the new state l to which the system will move from state k is determined using a random state generator with known probabilities. To develop the standard clock technique, it is necessary to apply the thinning algorithm. The purpose of the thinning algorithm is to reduce the number of events that the computer needs to process, while accurately representing the system dynamics. During sampling, events are distinguished as real and virtual, maintaining statistical consistency. Initially, a random number U is generated from a uniform distribution $U(0,1)$. This number is stored in the code under the variable name: U_event_type. It is generated using the command:

$$U\_event\_type = rand; \tag{66}$$

The possible outcomes are examined for the product type $product\_in\_m(sys)$. For production, $event = 1$, for setup, $event = 2$, and for non-occurrence (virtual event),

$event = -1$. Given the arrival, production, and setup rates defined in the file StClock_LotSize_in.txt (equation (5.4)), the probability thresholds determining the next candidate event are calculated:

A) $arrival = 1$: $U\_event\_type \leq \left(\frac{lambda}{mean\_rate\_all\_evts}\right) \Rightarrow$

$$U\_event\_type \leq \frac{0.03}{10.03} \approx 0{,}0029 \approx 0{,}3\% \tag{67}$$

This means that the random number U_event_type, has 0,3% probability of falling within the above range, and the next event will be arrival.

B) $arrival = 0; status = 1;$

$$0{,}0029 < U\_event\_type \leq \frac{(lambda + mu(product\_in\_m(sys))}{mean\_rate\_all\_evts} \Rightarrow \tag{70}$$

$$0{,}0029 < U\_event\_type \leq \frac{(10.03)}{10.03} = 1 \tag{68}$$

This means that with a 0,3% probability, the next event will be an arrival, and with the remaining probability, the next event will be production ($event(sys) = 1$). Production is the candidate event:

$$100 - 0{,}3 = 99{,}7\% \tag{71}$$

C) $arrival = 0; status = 0;$

$$0{,}0029 < U\_event\_type \leq \frac{(lambda + v(product\_in\_m(sys))}{mean\_rate\_all\_evts} \Rightarrow$$

$$0{,}0029 < U\_event\_type \leq \frac{(1.03)}{10.03} \approx 0{,}1026 \approx 10{,}26\% \tag{72}$$

Additionally, the remaining probability is: $100 - 10{,}26 - 0{,}3 = 89{,}44\%$.

$$0{,}1026 < U\_event\_type \leq 1 \tag{73}$$

In this case, there is still a 0,3% chance of an order arrival, a 10,26% chance that the next event will be machine setup, and the remaining 89,44% represents a virtual event. A very important factor for selecting the new state is the most recent machine status. The previous state will determine whether the next event can be production or setup, based on the following conditions:

$$\%\ product\_in\_m > 0\ and\ status = 0\ together, mean\ machine\ setup\ to\ start\ production\ of\ product\_in\_m$$

$\%\ product\_in\_m > 0\ and\ status = 1\ together, mean\ production\ of\ product\_in\_m$

$$\%product\_in\_m = 0\ and\ any\ status\ mean\ full\ machine\ idle$$

Then, the system transitions from state k to state l. The steps of the algorithm are repeated.

5.3 Performance Metrics Evaluation after Simulation

The simulation has been completed. The performance measures of the system represent the overall performance of each policy throughout the simulation duration t. They allow the comparison of different production policies using objective criteria and provide information for decision-making regarding the optimal policy. The system performance measures are computed as follows:

$$inv\_c(sys) = inv\_c(sys)/t; \tag{74}$$

$$delay\_c(sys) = delay\_c(sys)/t; \tag{75}$$

$$prod\_c(sys) = prod\_c(sys)/t; \tag{76}$$

$$setup\_c(sys) = setup\_c(sys)/t; \tag{77}$$

A key parameter for determining the optimal policy is the computation of the total average cost:

$$t\_c(sys) = inv\_c(sys) + delay\_c(sys) + prod\_c(sys) + setup\_c(sys); \tag{78}$$

5.4 Output Analysis

In this section we present the simulation results of all systems ($sensitivity = 1$) and only the base system ($sensitivity = 0$). The objective is to compare the required simulation times and assess the benefit (or not) of the standard clock technique.

5.4.1 Output Analysis Using Sensitivity = 1

The sensitivities using the standard clock method ($sensitivity = 1$) are presented below:

Table 5.2. **SENSITIVITY ANALYSIS OF LOT SCHEDULING POLICIES USING A STANDARD CLOCK**

sys	Final E	s1	S1	s2	S2	s3	S3	s4	S4	s5	S5	s6	S6	s7	S7	s8	S8	s9	S9	s10	S10
1	0	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15
2	0	-1	16	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15
3	0	-1	15	-1	16	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15

sys	Final E	s1	S1	s2	S2	s3	S3	s4	S4	s5	S5	s6	S6	s7	S7	s8	S8	s9	S9	s10	S10
4	0	-1	15	-1	15	-1	16	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15
5	0	-1	15	-1	15	-1	15	-1	16	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15
6	1	-1	15	-1	15	-1	15	-1	15	-1	16	-1	15	-1	15	-1	15	-1	15	-1	15
7	1	-1	15	-1	15	-1	15	-1	15	-1	15	-1	16	-1	15	-1	15	-1	15	-1	15
8	0	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	16	-1	15	-1	15	-1	15
9	0	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	16	-1	15	-1	15
10	0	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	16	-1	15
11	0	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	16
12	0	0	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15
13	0	-1	15	0	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15
14	0	-1	15	-1	15	0	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15
15	0	-1	15	-1	15	-1	15	0	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15
16	0	-1	15	-1	15	-1	15	-1	15	0	15	-1	15	-1	15	-1	15	-1	15	-1	15
17	0	-1	15	-1	15	-1	15	-1	15	-1	15	0	15	-1	15	-1	15	-1	15	-1	15
18	1	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	0	15	-1	15	-1	15	-1	15
19	0	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	0	15	-1	15	-1	15
20	0	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	0	15	-1	15
21	0	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	0	15
OPTIMAL																					
21	0	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	0	15

sys	TotOrders	Backorders	SetCstRt	InvCstRt	DelCstRt	PrdCstRt	TotCstRt
1	1.84E+06	1.67E+06	0.4354	9.6038	1.0067	0.8397	11.8856
2	1.84E+06	1.67E+06	0.4327	9.6193	0.9947	0.8397	11.8864
3	1.84E+06	1.67E+06	0.4326	9.6203	0.9958	0.8397	11.8884
4	1.84E+06	1.67E+06	0.4327	9.6187	0.9955	0.8397	11.8866
5	1.84E+06	1.67E+06	0.4327	9.6204	0.9956	0.8397	11.8884
6	1.84E+06	1.67E+06	0.4327	9.6205	0.9957	0.8397	11.8887
7	1.84E+06	1.67E+06	0.4327	9.6216	0.9957	0.8397	11.8897
8	1.84E+06	1.67E+06	0.4327	9.6183	0.9942	0.8397	11.8848
9	1.84E+06	1.67E+06	0.4327	9.6188	0.9942	0.8397	11.8854
10	1.84E+06	1.67E+06	0.4327	9.6202	0.9952	0.8397	11.8879
11	1.84E+06	1.67E+06	0.4327	9.6193	0.9943	0.8397	11.886
12	1.84E+06	1.66E+06	0.4389	9.5429	0.9418	0.8397	11.7632
13	1.84E+06	1.66E+06	0.4388	9.5446	0.9425	0.8397	11.7656
14	1.84E+06	1.66E+06	0.4388	9.5419	0.9411	0.8397	11.7615
15	1.84E+06	1.66E+06	0.4388	9.5416	0.9413	0.8397	11.7613
16	1.84E+06	1.66E+06	0.4388	9.5405	0.9401	0.8397	11.759
17	1.84E+06	1.66E+06	0.4388	9.5397	0.9397	0.8397	11.7579
18	1.84E+06	1.66E+06	0.4388	9.537	0.9382	0.8397	11.7536
19	1.84E+06	1.66E+06	0.4387	9.5351	0.9374	0.8397	11.751
20	1.84E+06	1.66E+06	0.4387	9.5357	0.9377	0.8397	11.7518
21	1.84E+06	1.66E+06	0.4387	9.5327	0.9364	0.8397	11.7475
OPTIMAL							
21	1.84E+06	1.66E+06	0.4387	9.5327	0.9364	0.8397	11.7475

The total number of systems that were simulated simultaneously was 21. The time required by the computer to complete the simulation was: $CPUtime = 90.284748\ s$. The number of

total arrivals from the simulation with a horizon of 100,000,000 events, is 18,500,000. The cost, in processing time, for generating the arrival event by the computer is:

$$\frac{CPU_{time}}{number\ of\ arrivals} = \frac{90.284748}{18{,}500{,}000} \approx 4.880 \times 10^{-6} s/arrival \tag{79}$$

5.4.2 Output Analysis Using Sensitivity = 0

Here, $sensitivity = 0$ has been set, and only the base system is simulated. The simulation results are presented below:

Table 5.3.: SIMULATION OF A SINGLE SYSTEM

sys	Final E	s_1	S_1	s_2	S_2	s_3	S_3	s_4	S_4	s_5	S_5	s_6	S_6	s_7	S_7	s_8	S_8	s_9	S_9	s_{10}	S_{10}
1	2	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	-1	15	1	-1

sys	TotOrders	Backorders	SetCstRt	InvCstRt	DelCstRt	PrdCstRt	TotCstRt
1	3.28E+07	2.98E+07	0.4353	9.6059	1.007	0.8397	11.8886

```
CPU t=10.875526
```

The time the computer needed to simulate the basic system is: $CPU_{time} = 10.875526\ s$. The total number of arrivals for the simulation with the horizon of 100,000,000 events is 32,800,000. The cost, in processing time, for generating the arrival event by the computer is:

$$\frac{CPU_{time}}{number\ of\ arrivals} = \frac{10.875526}{32{,}800{,}000} \approx 3.3157 \times 10^{-7} s/arrival \tag{80}$$

During the simulation, what matters is to study the flow of orders. For the comparison of the two methods to be fair, the method of simultaneous simulation of many systems together (standard clock) and the method of separate individual simulations must use the same number of actual arrivals as a reference. The standard clock method has fewer actual arrivals because it contains virtual events. To achieve this equality in the number of arrivals, the simulation time horizon must be increased. In detail:

$$scale = \frac{32{,}800{,}000}{18{,}500{,}000} = 1.77297 \tag{81}$$

The number of arrivals in the simulation of only the basic system is 1.77297 times greater than the standard clock method. In order for the standard clock method to produce the same arrivals, the number of events would have to be:

$$1.77297 \times 100{,}000{,}000 \approx 177{,}297{,}000\ events \tag{82}$$

The adjusted time for the simulation using the standard clock method is:

$$CPU_{time} = (177{,}297{,}000 \times 90.284748)/100{,}000{,}000 = 160.0721\ s.$$

The corresponding total time for 21 independent simulations is:

$$CPU_{time} = 21 \times 10.875521 = 228.3865\ s. \tag{83}$$

The time savings due to the use of the standard clock technique is calculated as:

$$speedup = \frac{228.3865}{160.0721} \approx 1.43 \tag{84}$$

Therefore, the standard clock method is approximately 1.43 times faster than the technique of separate simulations for the same number of arrivals.

6 CONCLUSIONS

For the simulation of the system, 10 different types of products were used, which correspond to 21 different systems, including the basic one. The identification of the best among the 21 policies, which had the lowest total cost rate, was based on sensitivity analysis. Additionally, the standard clock technique was used, which allowed sensitivity analysis from the first simulation, in contrast to sensitivity analysis that would require multiple simulations. The difference between these two methods was demonstrated by initially simulating all the systems simultaneously and then simulating only the basic system in a second step. The criterion used was the time the computer needed to execute the simulation of the 21 systems with the same number of arrivals, 32,800,000, in both cases.

- The simulation time when all systems were simulated simultaneously was
- $t_1 = 160.0721s$, while
- The total time for the simulation of 21 systems using independent simulations was $t_2 = 228.3865s$.

Therefore, the standard clock method is approximately 1.43 times faster than the technique of separate simulations for the same number of arrivals.

REFERENCES

1. Phillis Y, Kouikoglou V. Stochastic Processes. [Lecture Notes]. Technical University of Crete, Chania, Crete, 2006, p.147.
2. Kouikoglou V, Ioannidis E. Simulation. [Lecture Notes]. Technical University of Crete, Chania, Crete, 2022, p.120.
3. Kouikoglou V, Konstantas D. Simulation of Discrete Event Systems (in Greek). 1st ed. Disigma, 2016, p.335.
4. Scarf H. The optimality of (S, s) policies in the dynamic inventory problem. In Mathematical Methods in the Social Sciences, K. J. Arrow, S. Karlin, and P. Suppes (eds.), 1960; Palo Alto, CA: Stanford University Press.
5. Zipkin PH. Models for design and control of stochastic, multi-item batch production systems. Operations Research. 1986 Feb; 34(1):91-104.
6. Vakili P. Using a standard clock technique for efficient simulation. Operations Research Letters. 1991 Nov 1; 10(8):445-52.

APPENDIX I: ALGORITHM

The algorithm shown below has borrowed several parts from a Matlab function that has been developed at the Computer Aided Manufacturing laboratory of Technical University of Crete and is available upon request from Dr. Vassilis Kouikoglou (vkouikoglou@tuc.gr).

```
function[]=SC_LotSize()
clear all
clc

% Nominal+perturbed (standard clock);

% Batch production of products with (sj, Sj) policies

% We simulate the system until several million events occur,
% ievent: event counter, nevents: total number of events for the simulation to end

% 1. Read data
% Open the data file
infile=fopen('StClock_LotSize_in.txt');
% Load the first three lines which have instructions. The algorithm ignores them.
nextline=fgetl(infile);
nextline=fgetl(infile);
nextline=fgetl(infile);
% Load the next line, convert it to a text vector suffix
nextline=fgetl(infile);
suffix=sscanf(nextline,'%s');

% The next line has comments and we ignore it
nextline=fgetl(infile);
% Load the next line and convert it to a formatted vector A1
nextline=fgetl(infile);
A1=sscanf(nextline,'%d %d %d %f %d %d');
n=A1(1); seed0=A1(2); nevents=A1(3); lambda=A1(4);
start_print=A1(5); sensitivity=A1(6);
nextline=fgetl(infile);
for j=1:n
 nextline=fgetl(infile);
 A1=sscanf(nextline,'%f %f %f %f %f %f %f %f');
 s0(j)=A1(1); S0(j)=A1(2); v(j)=A1(3); mu(j)=A1(4);
 A(j)=A1(5); B(j)=A1(6); h(j)=A1(7); g(j)=A1(8);
end
% There are 2 categories of events: order arrival and machine event
% For each category we find the maximum possible rate and ADD THEM:
% for arrival there is no alternative other than lambda
nextline=fgetl(infile);
for j=1:n
%  number of possible product quantities j that may be requested in one order
%  except for the case where none is requested (quantity = 0)
 nextline=fgetl(infile);
 nextline=fgetl(infile);
 A1=sscanf(nextline,'%d');
```

```
 NQ(j)=A1(1);
 sumP=0;  % Initial value of cumulative probability function for quantity QM=0
 % For each INDEX q we read the corresponding quantity QM(j,q) and the
 % probability it is requested P(j,q). q is the INDEX of the quantity (the quantity is QM)
 nextline=fgetl(infile);
 for q=1:NQ(j)
  nextline=fgetl(infile);
  A1=sscanf(nextline,'%d %f');
  QM(j,q)=A1(1); P(j,q)=A1(2);
  sumP=sumP+P(j,q);
 end

% Cumulative distribution functions of possible quantities QM
 F(j,1)=P(j,1);
 for q=2:NQ(j)
  F(j,q)=F(j,q-1)+P(j,q);
 end
 F(j,NQ(j)+1)=1;
 % NQ(j)+1 represents the possibility that j is not requested (quantity=0)

end

% Seed for controlled random number generation
seed=uint32(seed0);
rng(seed,"twister");

% If sensitivity=1 then each of the next n has some S(j)+1,
% and each of the last n has some s(j)+1. Total of 1+n+n.
if sensitivity==0
 nsys=1; % We chose to simulate ONLY the base system (for simulation time comparison)
else
 nsys=2*n+1; % We simulate SIMULTANEOUSLY the base and the systems with one parameter change
end
% Parameter values sj Sj for each system sys=1,...,nsys.
for sys=1:nsys
 % Because differences from the base system are in one parameter, we initially assign base values.
 for j=1:n
  s(sys,j)=s0(j); S(sys,j)=S0(j);
 end
 omit(sys)=0;
end

if nsys>1
% Each system sys=2 to n+1 has nominal parameters except capital Sj + 1 for j=sys-1
% Each system sys=n+2 to 2n+1 has nominal parameters except lowercase sj + 1 for j=sys-n-1,
% provided that sj + 1 <= Sj, otherwise this policy makes no sense and the sys is omitted: omit(sys)=1
 for sys=2:n+1
  j=sys-1;
  S(sys,j)=S0(j)+1;
 end
 for sys=n+2:2*n+1
  j=sys-n-1;
  s(sys,j)=s0(j)+1;
  if s(sys,j)>S(sys,j)
   omit(sys)=1;
  end
 end
```

```
end
%* Display on screen % time until the end of the simulation
step0=nevents/50.;
sum_steps=step0;
txt=" % COMPLETE 0%"+blanks(21)+"50%"+blanks(21)+"100%";
fprintf('%s\n',txt);
txt=blanks(12)+"*";
fprintf('%s',txt);

% 2. Start

tic;
ievent = 0;
t=0;

% We examine each system sys, provided omit(sys)≠1

for sys=1:nsys
 if omit(sys)==1
  continue
 end
 E(sys) = 0; % Number of pending orders (currently). Initially = 0
 totE(sys)=0; % Total number of orders that were not served immediately
 totOrders(sys)=0; % Total number of orders placed
 setup_c(sys) = 0; % Total setup cost
 prod_c(sys) = 0; % Total production cost
 inv_c(sys) = 0;  % Total inventory cost
 delay_c(sys) = 0;  % Total delay cost in order delivery

 status(sys)=0;
 product_in_m(sys)=0; % product_in_m=type of product in machine. 0 means idle
 np_MustMake(sys)=0; % number of products with low stock that must be produced
 for j=1:n
  X(sys,j)=0;
  MustMake(sys,j)=0; % we have not yet checked if the product needs to be produced
  if X(sys,j)<s(sys,j)
   np_MustMake(sys)=np_MustMake(sys)+1;
   PL(sys,np_MustMake(sys))=j; % production list has product j at production priority
np_MustMake(sys)
   MustMake(sys,j)=1; % product j must be produced
   if product_in_m(sys)==0 % production command to machine
    %  product_in_m>0 and status=0 mean machine setup to start production of product_in_m
    %  product_in_m>0 and status=1 mean production of product_in_m
    %  product_in_m=0 and any status mean full machine idle
    product_in_m(sys)=j; status(sys)=0;
    setup_c(sys)=setup_c(sys)+B(j);
   end
  end
 end
end

% 3.
% Find and execute next event in each system sys
% To verify results and compare computational requirements, we use random numbers generated
from a fixed seed.
% Comparison of different systems with different simulations is unbiased when each simulation uses
the same sequence of random numbers
```

```
while ievent < nevents
 ievent = ievent + 1;
 mean_rate_all_evts = 0;
 for sys=1:nsys
  if omit(sys)==1
   continue
  end
  if product_in_m(sys)>0
   if status(sys)==1
    if mu(product_in_m(sys))>mean_rate_all_evts
     mean_rate_all_evts=mu(product_in_m(sys));
    end
   else
    if v(product_in_m(sys))>mean_rate_all_evts
     mean_rate_all_evts=v(product_in_m(sys));
    end
   end
  end
 end
 mean_rate_all_evts=mean_rate_all_evts+lambda; % Arrivals are always "active"
 t=t+1/mean_rate_all_evts; % Average time until the next event occurs
% 3a
% Find next event
 % Generate a random number from uniform distribution U(0,1), save it (henceforth U)
 % and from it determine which random event will occur in each system sys

 U_event_type = rand;  % Random number from uniform distribution U(0, 1) to find the event type
 if U_event_type <= lambda / mean_rate_all_evts
 %  single arrival, in all systems
  arrival = 1;
 else
  arrival = 0;
 % In each system there will be or not be a machine event depending on
 % its state and luck
  for sys=1:nsys
   if omit(sys)==1
    continue
   end
   event(sys) = -1;
   if product_in_m(sys)>0  % machine not idle
    if status(sys)==1 % machine produces product_in_m(sys)
     if U_event_type <= (lambda + mu(product_in_m(sys)))/mean_rate_all_evts
      event(sys) = 1; % Event 1: production of one more product of type product_in_m(sys)
     end
    else  % machine preparing to start production
     if U_event_type <= (lambda + v(product_in_m(sys)))/mean_rate_all_evts
      event(sys) = 2; % Event 2: end of setup, start production of product_in_m(sys)
     end
    end
   end
  end
 end
% 3β
% Execution of the next event

 if ievent>=start_print
  fprintf('\nRight Before event n.=%d arrival?=%d\n',ievent,arrival);
```

```
 for sys=1:nsys
  if omit(sys)==1 % if the policy of sys makes no sense, we don't simulate it and try the next sys
   continue
  end
  if arrival==1
   fprintf(' Arrival');
  elseif event(sys)==-1
   fprintf('No event');
  elseif event(sys)==1
   fprintf('Produces');
  elseif event(sys)==2
   fprintf('EndSetup');
  end
  fprintf(' Pending_orders_E=%4d MakesProd=%d Pr/setup=%d\t',...
        E(sys),product_in_m(sys),status(sys));
  for j=1:n-1
   fprintf('X%d=%5d MustMake%d=%d\t\t',j,X(sys,j),j,MustMake(sys,j));
  end
  fprintf('X%d=%5d MustMake%d=%d\n',n,X(sys,n),n,MustMake(sys,n));
%---------------------
  fprintf('sys\tFinal E\t');
  for j=1:n
   fprintf(' s%d\t S%d\t',j,j);
  end
  fprintf(' TotOrders\tBackorders\t SetCstRt \t');

  fprintf(' InvCstRt \t DelCstRt \t PrdCstRt \n');

   fprintf('%3d\t%7d\t',sys,E(sys));
   for j=1:n
    fprintf('%3d\t%3d\t',s(sys,j),S(sys,j));
   end
   fprintf(' %6.3E\t %6.3E\t',totOrders(sys),totE(sys));


   fprintf('%9.4f\t%9.4f\t%9.4f\t%9.4f\n',setup_c(sys),...
    inv_c(sys),delay_c(sys),prod_c(sys));
%---------------------
 end



  icont=...
  input('Enter s or S to STOP or any other SINGLE key to CONTINUE: ','s');
  if ~isempty(icont) && (icont=='s' || icont=='S') % isempty enables 'Enter' key to also mean 'continue'
   return;
  end
 end
% 3β.i EVENT=ARRIVAL
 if arrival == 1  % New order
 %  The following section is COMMON for all systems sys being simulated

 %  COMMON SECTION
  valid = 0;
  % The new order, how many products of each type does it request?
  for j=1:n
   % The order includes a different random quantity from EACH
```

```
 %  product j. So for each j we need a different random number
 U_quantity = rand;
 % How much of product j does the new order request?
 % Categories of quantities for product j: q=1,...,NQ(j) and q=NQ(j)+1 if no request
 q=0;
 ifound=0;
 while q<NQ(j) && ifound==0
  q=q+1;
  if U_quantity < F(j,q) % the order requests quantity QM(j,q)
   valid = 1;
   ifound=1;
   Q(j)=QM(j,q); % Q(j) = quantity of product j in the new order. SAME FOR ALL sys!
  end
 end
 if q==NQ(j) && ifound==0
  Q(j)=0;
 end
end
%  END COMMON SECTION
%  State changes for each system are checked.
 if valid==1   %
  for sys=1:nsys
   if omit(sys)==1
    continue
   end
   totOrders(sys)=totOrders(sys)+1;
   E(sys)=E(sys)+1;
   immediate_service = 1;
   for j = 1:n
    if Q(j)>0
     prod_c(sys) = prod_c(sys) + A(j)*Q(j);
     if X(sys,j) >= Q(j)
      RQ(sys,E(sys),j) = 0;
     elseif X(sys,j) > 0
      RQ(sys,E(sys),j) = Q(j)-X(sys,j);
     else
      RQ(sys,E(sys),j) = Q(j);
     end
     if RQ(sys,E(sys),j) > 0
      immediate_service = 0;
     end
     X(sys,j) = X(sys,j)-Q(j);
     % Check if for j a production batch order must be issued when before it didn't exist
     if X(sys,j)<s(sys,j) && MustMake(sys,j)==0
      MustMake(sys,j) = 1;
      np_MustMake(sys) = np_MustMake(sys) + 1; % increment pending production orders in machine
by 1
      PL(sys,np_MustMake(sys)) = j;  % store the product type corresponding to the order
     end
    else
     RQ(sys,E(sys),j)=0;
    end
   end
   if immediate_service == 1
    E(sys) = E(sys)-1;
    for j = 1:n
     % Little's law for stored Q(j) that leave now.
```

```
     % "Now" is calculated as MEAN time = t
     inv_c(sys) = inv_c(sys) + h(j)*Q(j)*t;
    end
   else
    totE(sys)=totE(sys)+1;
    % Delay cost of the new order depends also on the
    % quantities it includes. Even if only one unit of a product is missing,
    % the delay counts for all Q(j) of all j.
    %  Little's law:
     for j=1:n
      delay_c(sys) = delay_c(sys) - g(j)*Q(j)*t;  % delay in delivery of new order starts now and cost
depends on quantities
      originalQ(sys,E(sys),j) = Q(j);  % when delivered we must remember quantities to calculate delay
cost

     end

   end
 % If the machine was idle (product_in_m=0) but must work, then start setup on the highest priority
product
   if product_in_m(sys) == 0 && np_MustMake(sys) > 0
    j = PL(sys,1);
    product_in_m(sys)=j; status(sys)=0; % product_in_m=j>0 and status=0 mean setup for j
    setup_c(sys)=setup_c(sys)+B(j);
   end
  end
 end
% 3β.ii EVENT at the machine
 else
  for sys=1:nsys
   if omit(sys)==1
    continue
   end
   % 3β.ii (production of one unit)
   if event(sys) == 1 % end of production of one unit of product_in_m(sys)
    j=product_in_m(sys);
    X(sys,j) = X(sys,j) + 1;
    inv_c(sys) = inv_c(sys) - h(j)*t; % one unit is now stored and storage time starts
    % If there are orders that need product j, find which has priority
    % reduce its shortage by 1 and check if fully satisfied
    demand_for_j = 0; % Initially, no order needing j found
    i = 0;
    while demand_for_j == 0 && i<E(sys) % search for first priority order needing j
     i = i + 1;
     if RQ(sys,i,j) > 0
      demand_for_j = 1;
      RQ(sys,i,j) = RQ(sys,i,j) - 1;
      if RQ(sys,i,j)==0
       fully_satisfied_order = 1; % Initially assume order i is fully satisfied
       j1 = 0;
       while fully_satisfied_order == 1 && j1 < n  % If any product j1 is still missing, do not continue
search
        j1 = j1 + 1;
        if RQ(sys,i,j1) > 0  % There is still shortage in j1, so order i still pending
         fully_satisfied_order = 0;
        end
       end
```

```
      if fully_satisfied_order == 1
       for j1=1:n
        % originalQ units of product j1 of order i leave now.
        % Little's law for inventory and delay cost
        inv_c(sys) = inv_c(sys) + h(j1)*originalQ(sys,i,j1)*t;
        delay_c(sys) = delay_c(sys) + g(j1)*originalQ(sys,i,j1)*t;
       end
       for i1=i+1:E(sys)
        for j1 = 1:n
         RQ(sys,i1-1,j1)=RQ(sys,i1,j1);
         originalQ(sys,i1-1,j1)=originalQ(sys,i1,j1);
        end
       end
       E(sys)=E(sys)-1;
      end
     end
    end  % End of state changes due to partial or full satisfaction of order i
   end % End search for first priority order i needing this piece
   % Check if product j inventory reached target S_j
   if X(sys,j) == S(sys,j)
    MustMake(sys,j) = 0;
    for k = 2:np_MustMake(sys)
     PL(sys,k-1) = PL(sys,k);
    end
    np_MustMake(sys)=np_MustMake(sys)-1;
    if np_MustMake(sys) >= 1
     j = PL(sys,1);  % j : the NEW product type with priority
     product_in_m(sys)=j; status(sys)=0; % setup starts
     setup_c(sys)=setup_c(sys)+B(j);
    else
     product_in_m(sys)=0; status(sys)=0; % machine stops: product_in_m(sys)=0
    end
   end
  % 3β.ii (end of machine setup)
  elseif event(sys)==2 %
   status(sys)=1; % machine in sys starts producing product product_in_m(sys)
  end
 end % End execution of machine events for each sys=1:nsys
end
%3.β END OF EVENT EXECUTION

%* Display simulation time % on screen
 if ievent>sum_steps
  sum_steps=sum_steps+step0;
  fprintf('*');
 end

end

% END OF ALL EVENTS EXECUTION

CPUtime=toc;
% 4. Calculation of performance measures for all policies and finding the optimal policy
% For each sys, we calculate the inventory and delay areas and the total cost

name_out_file=append('SC_LotSize_out_',suffix,'_.txt');
out = fopen(name_out_file,'w');
```

```
fprintf(out,'SENSITIVITY ANALYSIS OF LOT SCHEDULING POLICIES USING A ');
fprintf(out,'STANDARD CLOCK\n');
fprintf(out,'-------------------------------------------------------');
fprintf(out,'--------------\n');
fprintf('\nSENSITIVITY ANALYSIS OF LOT SCHEDULING POLICIES USING A ');
fprintf('STANDARD CLOCK\n');
fprintf('-------------------------------------------------------');
fprintf('--------------\n');

fprintf(out,'sys\tFinal E\t');
fprintf('sys\tFinal E\t');
for j=1:n
 fprintf(out,' s%d\t S%d\t',j,j);
 fprintf(' s%d\t S%d\t',j,j);
end
fprintf(out,' TotOrders\tBackorders\t SetCstRt \t InvCstRt \t');
fprintf(out,' DelCstRt \t PrdCstRt \t TotCstRt\n');
fprintf(' TotOrders\tBackorders\t SetCstRt \t InvCstRt \t');
fprintf(' DelCstRt \t PrdCstRt \t TotCstRt\n');

for sys=1:nsys
 if omit(sys)==1
  continue
 end
 for j=1:n
  if X(sys,j)>0
   inv_c(sys) = inv_c(sys) + h(j)*X(sys,j)*t;
  end
 end
 for i=1:E(sys)
   for j=1:n
   % The difference originalQ(sys,i,j)-RQ(sys,i,j) is the pieces j of
   % order i remaining in inventory waiting for i to be completed:
   inv_c(sys) = inv_c(sys) + h(j)*(originalQ(sys,i,j)-RQ(sys,i,j))*t;
   % originalQ(sys,i,j) is the initial quantity of pieces j in
   % order i, which is still delayed:
   delay_c(sys) = delay_c(sys) + g(j)*originalQ(sys,i,j)*t;
    end
 end
 inv_c(sys)=inv_c(sys)/t;
 delay_c(sys)=delay_c(sys)/t;
  % The above (average cost per unit time) = (average total cost) / (average simulation time)
  prod_c(sys)=prod_c(sys)/t;
 setup_c(sys)=setup_c(sys)/t;
 t_c(sys)=inv_c(sys)+delay_c(sys)+prod_c(sys)+setup_c(sys); % Total average cost per unit time

 if sys>1
  if t_c(sys)<opt_c
   opt_c=t_c(sys);
   opt_sys=sys;
  end
 else
  opt_c=t_c(1);
  opt_sys=1;
 end
 fprintf(out,'%3d\t%7d\t',sys,E(sys));
 fprintf('%3d\t%7d\t',sys,E(sys));
```

```
 for j=1:n
  fprintf(out,'%3d\t%3d\t',s(sys,j),S(sys,j));
  fprintf('%3d\t%3d\t',s(sys,j),S(sys,j));
 end
 fprintf(out,' %6.3E\t %6.3E\t',totOrders(sys),totE(sys));
 fprintf(out,'%9.4f\t%9.4f\t%9.4f\t%9.4f\t%9.4f\n',setup_c(sys),...
        inv_c(sys),delay_c(sys),prod_c(sys),t_c(sys));
 fprintf(' %6.3E\t %6.3E\t',totOrders(sys),totE(sys));
 fprintf('%9.4f\t%9.4f\t%9.4f\t%9.4f\t%9.4f\n',setup_c(sys),...
        inv_c(sys),delay_c(sys),prod_c(sys),t_c(sys));
end
sys=opt_sys;
fprintf(out,'OPTIMAL\n');
fprintf('OPTIMAL\n');
fprintf(out,'%3d\t%7d\t',sys,E(sys));
fprintf('%3d\t%7d\t',sys,E(sys));
for j=1:n
 fprintf(out,'%3d\t%3d\t',s(sys,j),S(sys,j));
 fprintf('%3d\t%3d\t',s(sys,j),S(sys,j));
end
 fprintf(out,' %6.3E\t %6.3E\t',totOrders(sys),totE(sys));
 fprintf(out,'%9.4f\t%9.4f\t%9.4f\t%9.4f\t%9.4f\n',setup_c(sys),...
        inv_c(sys),delay_c(sys),prod_c(sys),t_c(sys));
 fprintf(' %6.3E\t %6.3E\t',totOrders(sys),totE(sys));
 fprintf('%9.4f\t%9.4f\t%9.4f\t%9.4f\t%9.4f\n',setup_c(sys),...
        inv_c(sys),delay_c(sys),prod_c(sys),t_c(sys));

fprintf(out,'\nEvents=%d\tE(time)=%.1f\tCPU t=%.6f s  ',nevents,...
 t,CPUtime);
fprintf('\nEvents=%d\tE(time)=%.1f\tCPU t=%.6f s  ',nevents,...
 t,CPUtime);
fclose('all'); % Close all open files
end
```
