

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра системного анализа
и автоматического управления

**Исследование производственных систем с маршрутизацией,
зависящей от состояния**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА СПЕЦИАЛИСТА

студента 5 курса 511 группы
специальности 010501 — прикладная математика и информатика
факультета компьютерных наук и информационных технологий
Салина Романа Владимировича

Научный руководитель
доцент, к.ф.-м.н.

В. И. Долгов

Заведующий кафедрой
д.т.н., профессор

Ю. И. Митрофанов

Саратов 2014

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Основные понятия массового обслуживания	5
1.1 Некоторые распределения случайных величин	5
1.2 Процесс размножения и гибели	7
1.3 Основные параметры и характеристики сетей массового обслуживания	11
2 Гибкие производственные системы с маршрутизацией, зависящей от состояния	16
2.1 Описание модели	16
2.2 Решение уравнения равновесия	20
3 Алгоритм метода анализа производственных систем с маршрутизацией, зависящей от состояния	24
3.1 Описание алгоритма	24
3.2 Структурная схема алгоритма	28
4 Описание и назначение программы	33
4.1 Список идентификаторов	33
4.2 Описание программы	34
4.3 Описание и назначение функций	38
5 Аспекты практического применения	42
ЗАКЛЮЧЕНИЕ	47
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	48
Приложение А. Код программы	49

ВВЕДЕНИЕ

Теория массового обслуживания является областью прикладной математики, объектами исследования которой являются математические модели определённого класса дискретных систем со стохастическим характером функционирования. Принадлежащие к этому классу системы служат для обслуживания поступающих объектов, называемых в рамках теории массового обслуживания требованиями. Процессы поступления и обслуживания требований в общем случае являются случайными.

В последнее время наблюдается значительное количество исследований, направленных на моделирование гибких производственных систем с помощью сетей массового обслуживания. Практическое значение этого направления определяется широким использованием сетей массового обслуживания в качестве математических моделей гибких производственных систем, необходимостью исследования производственных систем, повышения качества функционирования и оптимизации.

В данной выпускной квалификационной работе рассматривается сеть массового обслуживания с зависимой от состояния сети маршрутизацией как модель гибкой производственной системы. В сети используется вероятностная маршрутизация в кратчайшую очередь (PSQ–маршрутизация). При данной маршрутизации маршрутные вероятности больше для систем с бóльшим числом свободных приборов. В этом случае стационарное распределение вероятностей состояний сети имеет мультипликативную форму, для которого имеется эффективный алгоритм вычисления. Важно отметить, что в рассматриваемой модели допускаются как ограниченные, так и неограниченные очереди в системах.

PSQ–маршрутизация — это рандомизированная версия детерминированной маршрутизации в кратчайшую очередь (DSQ–маршрутизации), при которой требования всегда (т.е. с вероятностью 1) поступают в систему с кратчайшей очередью. Хотя DSQ–маршрутизация в общем не поддается точному моделированию, она может быть смоделирована приближенно с помощью схемы PSQ–маршрутизации при определенных условиях. В общем случае маршрутизация (детерминированная или вероятностная) в кратчайшую очередь не обязательно должна быть оптимальной с точки зрения минимизации ожида-

ния требованием обслуживания или максимизации пропускной способности сети. Однако, у PSQ–маршрутизации есть преимущество в снижении блокирования требований и повышении коэффициентов использования обслуживающих приборов, а также в простоте и удобстве для анализа.

Целью выпускной квалификационной работы является исследование производственных систем с маршрутизацией, зависящей от состояния, разработка алгоритма метода анализа данных производственных систем, программная реализация алгоритма и проведение численных экспериментов с разработанной программой.

Выпускная квалификационная работа состоит из введения, пяти разделов, заключения, списка использованных источников и приложения.

В разделе 1 приводятся основные понятия и определения теории сетей массового обслуживания, рассматриваются их параметры и характеристики. Также приводятся некоторые распределения случайных величин и процесс размножения и гибели, являющийся основополагающим в теории массового обслуживания.

В разделе 2 представлено описание гибких производственных систем, дано определение PSQ–маршрутизации, приведена теорема и доказательство того, что стационарное распределение вероятностей состояний сети имеет мультипликативную форму.

В разделе 3 приводится алгоритм метода анализа производственных систем с маршрутизацией, зависящей от состояния, описание алгоритма и структурная схема.

В разделе 4 представлено описание и назначение разработанной программы, предназначенной для анализа производственных систем с маршрутизацией, зависящей от состояния.

В разделе 5 приведены результаты проведенных экспериментов для иллюстрации возможностей модели.

Приложение А содержит программный код алгоритма анализа.

1 Основные понятия массового обслуживания

1.1 Некоторые распределения случайных величин

В теории массового обслуживания особое место занимают экспоненциальное распределение, распределение Эрланга и гиперэкспоненциальное распределение.

Экспоненциальное распределение. Непрерывная случайная величина ξ имеет экспоненциальное распределение с параметром $\lambda > 0$, если функция распределения имеет вид

$$F(t) = P\{\xi < t\} = 1 - e^{-\lambda t}, \quad t \geq 0.$$

Соответствующая функция плотности распределения

$$f(t) = \lambda e^{-\lambda t}, \quad t \geq 0.$$

Математическое ожидание и дисперсия случайной величины равны

$$E\xi = \frac{1}{\lambda}; \quad \text{var}\xi = \frac{1}{\lambda^2}.$$

Экспоненциальное распределение имеет свойство отсутствия памяти, которое ставит его на одно из центральных мест в теории массового обслуживания.

Распределение Эрланга. Непрерывная случайная величина ξ имеет распределение Эрланга порядка k с параметром $\lambda > 0$, если ее функция распределения имеет вид

$$F_k(t) = 1 - e^{-\lambda t} \sum_{i=0}^{k-1} \frac{(\lambda t)^i}{i!}, \quad t \geq 0.$$

Функция плотности распределения величины ξ имеет вид

$$f(t) = \frac{\lambda^k t^{k-1}}{(k-1)!} e^{-\lambda t}, \quad t \geq 0.$$

(при $k = 1$ получим плотность экспоненциального распределения.)

Распределение Эрланга непосредственно связано с распределением

Пуассона: если начать измерение времени в момент совершения i -го скачка пуассоновского процесса, то случайная величина ξ является длительностью интервала времени между i -м и $(i + k)$ -м скачками (точками) пуассоновского процесса. Этот интервал, очевидно, равен сумме k подынтервалов, каждый из которых является случайной величиной, имеющей экспоненциальное распределение с параметром λ .

Случайная величина ξ , имеющая распределение Эрланга порядка k с параметром λ , представима в виде суммы k стохастических независимых и одинаково экспоненциально распределенных (с параметром λ) случайных величин $\xi_i, i = 1, \dots, k$, т. е. $\xi = \xi_1 + \xi_2 + \dots + \xi_k$. Математическое ожидание и дисперсия случайной величины ξ равны

$$E\xi = \frac{k}{\lambda}; \quad \text{var}\xi = \frac{k}{\lambda^2}.$$

Гиперэкспоненциальное распределение. Непрерывная случайная величина ξ имеет гиперэкспоненциальное распределение порядка k с параметрами $\lambda_1, \lambda_2, \dots, \lambda_k > 0$ и q_1, q_2, \dots, q_k , если ее функция плотности распределения имеет вид

$$f(t) = q_1 \lambda_1 e^{-\lambda_1 t} + q_2 \lambda_2 e^{-\lambda_2 t} + \dots + q_k \lambda_k e^{-\lambda_k t},$$

где q_i — вероятность, $i = 1, \dots, k$, $\sum_{i=1}^k q_i = 1$, с которой параметр экспоненциально распределенной величины ξ примет значение λ_i . Математическое ожидание и дисперсия случайной величины ξ равны

$$E\xi = \sum_{i=1}^k \frac{q_i}{\lambda_i}; \quad \text{var}\xi = 2 \sum_{i=1}^k \frac{q_i}{\lambda_i^2} - \left(\sum_{i=1}^k \frac{q_i}{\lambda_i} \right)^2.$$

При гиперэкспоненциальном распределении интервалы между последовательными независимыми событиями распределены по экспоненциальному закону, однако параметр этого закона для каждого интервала может принимать значение $\lambda_1, \lambda_2, \dots, \lambda_k$ с соответствующими вероятностями q_1, q_2, \dots, q_k . Тогда плотность распределения величины ξ будет представлять собой суперпозицию экспоненциальных распределений.

Коэффициент вариации. Сравним коэффициенты вариации $C_\xi^M, C_\xi^E, C_\xi^H$ величины ξ , имеющей соответственно экспоненциальное распределение,

распределение Эрланга порядка k и гиперэкспоненциальное распределение ($C_\xi = \sqrt{\text{var}\xi}/E\xi$):

$$C_\xi^M = 1; \quad C_\xi^E = \frac{1}{\sqrt{k}} < 1, \quad k > 1; \quad C_\xi^H > 1.$$

При одном и том же значении математического ожидания распределение Эрланга порядка $k > 1$ обладает меньшим рассеиванием, чем экспоненциальное распределение, и, наоборот, гиперэкспоненциальное распределение обладает большим рассеиванием.

1.2 Процесс размножения и гибели

Рассмотрим случайный процесс $\xi = \{\xi(t) : t \in T = [0, +\infty)\}$, принимающий значения из $E = \{0, 1, 2, \dots\}$, и предположим, что для любого целого числа $n \geq 1$, любых моментов $t_1 < \dots < t_{n+1}$ из T и любых состояний i_1, \dots, i_{n+1} из E таких, что $P\{\xi(t_1) = i_1, \dots, \xi(t_n) = i_n\} > 0$, выполняется равенство:

$$P\{\xi(t_{n+1}) = i_{n+1} | \xi(t_1) = i_1, \dots, \xi(t_n) = i_n\} = P\{\xi(t_{n+1}) = i_{n+1} | \xi(t_n) = i_n\}.$$

О случайных величинах $\xi(t)$, $t \in T$, удовлетворяющим равенству выше, говорят, что они связаны в цепь Маркова, а сам случайный процесс ξ называется цепью Маркова с непрерывным временем. Такая цепь называется однородной (относительно времени), если вероятность

$$P\{\xi(s+t) = j | \xi(s) = i\} = p_{ij}^{(t)}$$

не зависит от $s \in T$ для любых $i, j \in E$ и $t \in T$. При этом $p_{ij}^{(t)}$ называется вероятностью перехода из состояния i в состояние j за время t .

Однородная цепь Маркова $\{\xi(t) : t \geq 0\}$ называется стандартной, если $p_{ij}^{(t)} \rightarrow \delta_{ij}$ при $t \rightarrow 0$ для всех i и j из E , где δ_{ij} — элементы единичной матрицы. Для стандартной цепи Маркова со счетным множеством состояний при $i \neq j$

$$\lim_{t \rightarrow 0} \frac{p_{ij}^{(t)}}{t} = a_{ij} \geq 0$$

существует и конечен, а

$$\lim_{t \rightarrow 0} \frac{1}{t} (1 - p_{ii}^{(t)}) = -a_{ii}$$

существует, но может быть бесконечным. Элементы матрицы $A = (a_{ij})$ удовлетворяют условию

$$a_{ij} \geq 0 \text{ при } i \neq j, \quad \sum_j a_{ij} \leq 0 \quad \forall i.$$

Оператор A называется инфинитезимальным оператором цепи Маркова.

Частным случаем однородной цепи Маркова с непрерывным временем и счетным множеством состояний является процесс размножения и гибели $\{\xi(t) : t \geq 0\}$. Для данного процесса допускаются переходы из состояния n только в соседние состояния $n - 1$ и $n + 1$.

Обозначим через λ_n и μ_n соответственно интенсивности переходов $n \rightarrow n + 1$ и $n \rightarrow n - 1$, т. е. $\lambda_n = a_{n,n+1}$, $\mu_n = a_{n,n-1}$. Заметим, что введенные интенсивности не зависят от времени, а зависят только от состояния n ; следовательно, имеем однородную цепь Маркова с непрерывным временем. Требование о допустимости переходов только в ближайшие состояния означает, что $a_{n,j} = 0$ при $|n - j| > 1$. Более того, так как $\sum_j a_{n,j} = 0$ (заметим, что это равенство справедливо только для консервативных цепей), это требование означает, что $a_{n,n} = -(\lambda_n + \mu_n)$.

Таким образом, инфинитезимальная матрица для общего процесса размножения и гибели

$$A = \begin{bmatrix} -\lambda_0 & \lambda_0 & 0 & & \\ \mu_1 & -(\lambda_1 + \mu_1) & \lambda_1 & \mathbf{0} & \\ 0 & \mu_2 & -(\lambda_2 + \mu_2) & & \\ 0 & 0 & \mu_3 & & \\ & \mathbf{0} & & \ddots & \end{bmatrix},$$

то есть A является трехдиагональной матрицей.

Некоторый процесс представляет собой процесс размножения и гибели, если он является однородной цепью Маркова с непрерывным временем и счетным множеством состояний и если выполняются следующие условия:

- С1: $P \{ \text{точно одного перехода } n \rightarrow n+1 \text{ за время } \Delta t \} = \lambda_n \Delta t + o(\Delta t);$
С2: $P \{ \text{точно одного перехода } n \rightarrow n-1 \text{ за время } \Delta t \} = \mu_n \Delta t + o(\Delta t);$
С3: $P \{ \text{отсутствия переходов } n \rightarrow n+1 \text{ в течение } \Delta t \} = 1 - \lambda_n \Delta t + o(\Delta t);$
С4: $P \{ \text{отсутствия переходов } n \rightarrow n-1 \text{ в течение } \Delta t \} = 1 - \mu_n \Delta t + o(\Delta t).$

Обозначим через $P_n(t) = P \{ \xi(t) = n \}$, $t \geq 0$, $n \in E$, вероятность пребывания процесса в некоторый момент t в состоянии n , а через $p_{ij}^{(\Delta t)}$ — вероятность перехода процесса из состояния $i \rightarrow j$ за время Δt . Тогда

$$P_n(t + \Delta t) = P_n(t)p_{n,n}^{(\Delta t)} + P_{n-1}(t)p_{n-1,n}^{(\Delta t)} + P_{n+1}(t)p_{n+1,n}^{(\Delta t)} + o(\Delta t), \quad n \geq 1.$$

Причем, очевидно, в момент времени t все возможные состояния должны удовлетворять нормирующему условию $\sum_{n=0}^{\infty} P_n(t) = 1$. Теперь воспользуемся предположениями С1 – С4. Получим

$$P_n(t + \Delta t) = P_n(t)[1 - \lambda_n \Delta t][1 - \mu_n \Delta t] + P_{n-1}(t)\lambda_{n-1}\Delta t + \\ + P_{n+1}(t)\mu_{n+1}\Delta t + o(\Delta t), \quad n \geq 1;$$

$$P_0(t + \Delta t) = P_0(t)[1 - \lambda_0 \Delta t][1 - \mu_0 \Delta t] + P_1(t)\mu_1\Delta t + o(\Delta t), \quad n = 0.$$

Раскрывая скобки, получим

$$P_n(t + \Delta t) = P_n(t) - (\lambda_n + \mu_n)\Delta t P_n(t) + \lambda_{n-1}\Delta t P_{n-1}(t) + \\ + \mu_{n+1}\Delta t P_{n+1}(t) + o(\Delta t), \quad n \geq 1;$$

$$P_0(t + \Delta t) = P_0(t) - \lambda_0\Delta t P_0(t) + \mu_1\Delta t P_1(t) + o(\Delta t), \quad n = 0.$$

Выделяя теперь $P_n(t)$ из обеих частей в обоих равенствах, деля на Δt и переходя к пределу при $\Delta t \rightarrow 0$, получим

$$\frac{dP_n(t)}{dt} = -(\lambda_n + \mu_n)P_n(t) + \lambda_{n-1}P_{n-1}(t) + \mu_{n+1}P_{n+1}(t), \quad n \geq 1, \quad (1.1)$$

$$\frac{dP_0(t)}{dt} = -\lambda_0 P_0(t) + \mu_1 P_1(t), \quad n = 0.$$

Эта система дифференциальных уравнений описывает динамику рассматриваемого процесса.

При $\mu_n = 0$ для всех n процесс размножения и гибели определяется как процесс чистого размножения. И если предположить, что $\lambda_n = \lambda$ для всех $n = 0, 1, 2, \dots$, то получим процесс Пуассона.

Обозначим p_n , $n \in E$, предельные вероятности пребывания процесса в состояниях n , т. е.

$$p_n = \lim_{t \rightarrow \infty} P_n(t).$$

Предполагая, что пределы p_n существуют, производные в уравнениях (1.1) при $t \rightarrow \infty$ обращаются в нуль и (1.1) принимает вид:

$$0 = -(\lambda_n + \mu_n)p_n + \lambda_{n-1}p_{n-1} + \mu_{n+1}p_{n+1}, \quad n \geq 0. \quad (1.2)$$

В уравнении (1.2) принято, что $\mu_0 = \mu_i = \lambda_i = p_i = 0$ при $i < 0$. Из (1.2) можно непосредственно определить, что

$$p_n = \frac{\lambda_0 \lambda_1 \dots \lambda_{n-1}}{\mu_1 \mu_2 \dots \mu_n} p_0,$$

или

$$p_n = p_0 \prod_{i=0}^{n-1} \frac{\lambda_i}{\mu_{i+1}}, \quad n = 1, 2, \dots \quad (1.3)$$

Используя нормирующее условие $\sum_{n=0}^{\infty} p_n = 1$, получим

$$p_0 = \frac{1}{1 + \sum_{n=1}^{\infty} \prod_{i=0}^{n-1} \frac{\lambda_i}{\mu_{i+1}}}. \quad (1.4)$$

Это решение для p_n в виде произведения является основополагающим равенством элементарной теории массового обслуживания. Для того, чтобы выражения (1.3) и (1.4) задавали вероятности, обычно накладывается требование, чтобы $p_0 > 0$ [1].

1.3 Основные параметры и характеристики сетей массового обслуживания

Сеть массового обслуживания (СеМО) представляет собой совокупность взаимосвязанных систем массового обслуживания (СМО), обеспечивающих в процессе функционирования сети прием, хранение, обработку и выдачу требований, поступающих в системы обслуживания.

Сети массового обслуживания, в которых обслуживаются требования только одного класса, называются *однородными*, а сети, обслуживающие требования различных классов, — *неоднородными*.

По отношению к внешнему источнику требований сети обслуживания делятся на открытые, замкнутые и смешанные. *Открытые* СеМО имеют внешний источник требований бесконечной емкости; требования поступают в сеть из источника, обслуживаются в сети и возвращаются в источник. Число требований, пребывающих в открытой СеМО в процессе ее эволюции, является дискретной случайной величиной. *Замкнутые* СеМО не имеют внешних источников требований. Число требований, пребывающих в замкнутой СеМО, является постоянной величиной. Сети обслуживания, являющиеся открытыми для одних классов требований и замкнутыми для других классов требований, называются *смешанными*.

По типам функций распределения длительностей обслуживания требований во входящих в сеть системах обслуживания сети обслуживания делятся на экспоненциальные и общего вида. В *экспоненциальной* СеМО длительности обслуживания требований во всех СМО сети являются непрерывными случайными величинами с экспоненциальным распределением; разумеется, параметры этих функций распределения могут быть различными. В СеМО *общего вида* функции распределения длительностей обслуживания требований в СМО могут быть произвольными (в частности, экспоненциальными).

Одним из основных параметров сети массового обслуживания является *маршрутная матрица*, элементами которой являются вероятности перехода требований между системами обслуживания сети. Рассмотрим в качестве примера однородную замкнутую экспоненциальную СеМО, включающую L систем обслуживания. Обозначим через $\Theta = (\theta_{ij})$, $i, j = 1, \dots, L$, маршрутную матрицу данной сети; элемент θ_{ij} равен вероятности перехода требования из системы обслуживания с номером i в систему обслуживания с номером j по-

сле завершения обслуживания данного требования в i -ой СМО. Матрица Θ является стохастической. Как правило, маршрутные матрицы являются разреженными — матрицами с большим числом нулевых элементов. Однородная открытая СеМО с L системами обслуживания имеет маршрутную матрицу $\Theta = (\theta_{ij})$, $i, j = 0, 1, \dots, L$, порядка $L + 1$. Элементы θ_{0j} и θ_{i0} , $1 \leq i, j \leq L$, определяют соответственно вероятности поступления требований из источника в j -ую СМО и из i -ой СМО в источник. Переход требований между системами обслуживания в сетях происходит мгновенно.

Полное определение сети обслуживания включает также задание типов дисциплин обслуживания в системах массового обслуживания, состава СМО (число одинаковых параллельных приборов в СМО), зависимости интенсивности обслуживания в СМО от ее состояния и другие характеристики.

Введем основные обозначения, связанные с СеМО. При этом будут использоваться следующие обозначения и сокращения: $B = |\mathcal{B}|$ — мощность множества \mathcal{B} ; $a = (a_i)$ — вектор-строка; ф. р. — функция распределения; м. о. — математическое ожидание.

$\mathcal{L} = \{1, 2, \dots, L\}$ — конечное множество номеров систем массового обслуживания в СеМО;

$L = |\mathcal{L}|$ — число СМО в СеМО;

$\mathcal{K} = \{1, 2, \dots, K\}$ — конечное множество номеров классов требований в СеМО;

$K = |\mathcal{K}|$ — число классов требований в СеМО (для однородной СеМО $K = 1$), требования класса с номером k будем называть k -требованиями, $k \in \mathcal{K}$;

A — параметр, определяющий тип СеМО (открытая или замкнутая),

$$A = \begin{cases} \lambda_0 — \text{однородная открытая сеть } (K = 1), \\ \Lambda — \text{неоднородная открытая сеть } (K > 1), \\ N — \text{однородная замкнутая сеть } (K = 1), \\ H — \text{неоднородная замкнутая сеть } (K > 1); \end{cases}$$

λ_0 — интенсивность внешнего потока требований, поступающих в сеть из источника и из сети в источник;

$\Lambda = (\lambda_{0k})$ — вектор интенсивностей внешнего потока требований, $k = 1, \dots, K$;

λ_{0k} — интенсивность внешнего потока k -требований;

N — число требований в СеМО;

$H = (H_k)$, $k = 1, \dots, K$, — вектор начального числа требований в сети массового обслуживания,

H_k — начальное число k -требований в СеМО;

$\hat{H} = \sum_{k=1}^K H_k$ — общее число требований в СеМО;

C_i — СМО с номером i , входящая в состав СеМО, $i \in \mathcal{L}$;

C_0 — внешний источник (и сток) требований (в открытых СеМО);

$n = (n_i)$ — вектор состояния сети обслуживания, $i = 1, \dots, L$;

$n_i = (n_{ik})$ — вектор состояния системы обслуживания C_i , $i \in \mathcal{L}$, $k = 1, \dots, K$;

n_{ik} — число k -требований в C_i ;

$W = (W_i)$ — вектор типов функций распределения длительностей обслуживания в СМО сети, $i = 1, \dots, L$;

W_i — тип ф. р. длительности обслуживания в C_i ; длительности обслуживания требований в каждой системе C_i предполагаются одинаково распределенными случайными величинами с функциями распределения либо экспоненциальными ($W_i = M$), либо общего вида ($W_i = GI$);

W_0 — тип ф. р. длительности интервалов времени между последовательными требованиями во внешнем потоке (для открытых сетей обслуживания);

$\Theta = (\theta_{ik,jl})$ — маршрутная матрица, $k, l = 1, \dots, K$,

$$i, j = \begin{cases} 0, 1, \dots, L, & \text{если СеМО открытая,} \\ 1, \dots, L, & \text{если СеМО замкнутая;} \end{cases}$$

$\theta_{ik,jl}$ — вероятность того, что k -требование после обслуживания в C_i поступает в C_j и изменяет свой класс на l -й;

$\kappa = (\kappa_i)$ — вектор числа приборов в системах обслуживания СеМО, $i = 1, \dots, L$;

κ_i — число идентичных обслуживающих приборов в СМО C_i ;

$D = (D_i)$ — вектор дисциплин обслуживания в системах обслуживания СеМО, $i = 1, \dots, L$;

D_i — дисциплина обслуживания в системе C_i ;

FCFS — обслуживание требований в порядке поступления;

RANDOM — очередное требование выбирается на обслуживание наугад;

$LCFSPR$ — обслуживание требований в порядке, обратном поступлению, с приоритетным дообслуживанием;

PS — обслуживание требований с разделением производительности обслуживания прибора;

IS — обслуживание требований «бесконечным» числом приборов;

FS — обслуживание требований «конечным» числом приборов;

$\mu(n) = (\mu_{ik}(n_i))$ — матрица интенсивностей обслуживания требований системами СеМО, $i = 1, \dots, L$, $k = 1, \dots, K$;

n_i — состояние системы C_i ;

$\mu_{ik}(n_i)$ — интенсивность обслуживания в системе C_i k -требований при условии, что система C_i находится в состоянии n_i .

С учетом введенных обозначений будем считать, что сеть массового обслуживания определена, если задан набор

$$\Gamma = \langle L, K, A, W, \Theta, \kappa, \mu, D \rangle.$$

При анализе сетей массового обслуживания основной интерес представляют, как правило, точные или приближенные значения следующих характеристик СеМО для стационарного режима; $i, j \in \mathcal{L}$, $k, l \in \mathcal{K}$:

\bar{n}_{ik} — м. о. числа k -требований в C_i ;

\bar{n}_i — м. о. числа требований в C_i ;

\bar{b}_{ik} — м. о. числа k -требований, ожидающих обслуживания в очереди системы C_i ;

\bar{b}_i — м. о. числа требований, ожидающих обслуживания в очереди системы C_i ;

\bar{h}_i — м. о. числа занятых приборов в C_i ;

\bar{g}_i — м. о. числа свободных приборов в C_i ;

λ_{ik} — интенсивность потока k -требований в C_i ;

λ_i — интенсивность потока требований в C_i ;

Ψ_{ik} — коэффициент использования обслуживающих приборов системы C_i k -требованиями;

Ψ_i — коэффициент использования обслуживающих приборов системы C_i ;

\bar{u}_{ik} — м. о. длительности пребывания k -требований в C_i ;

$\bar{\nu}_{ik}$ — м. о. длительности обслуживания k -требований в C_i ;

\bar{w}_{ik} — м. о. длительности ожидания k -требований в очереди системы C_i ;

$\bar{\tau}_{ik}$ — м. о. длительности реакции для k -требований в C_i (м. о. длительности интервала времени от момента выхода k -требования из C_i до момента его следующего поступления в C_i);

$\bar{\sigma}_{ik}$ — м. о. длительности цикла для k -требований в C_i (м. о. длительности интервала времени от момента поступления k -требования в C_i до момента его следующего поступления в C_i);

$\bar{\varphi}_{ik,jl}$ — м. о. длительности перехода k -требований из C_i в C_j с изменением класса требований на l -й (м. о. длительности интервала времени от момента поступления k -требования в C_i до момента его первого поступления в C_j с изменением класса требования на l -й);

$P\{n_i = s\}$ — стационарное распределение состояний системы C_i ;

$n_i = (n_{i1}, n_{i2}, \dots, n_{iK})$, n_{ik} — число k -требований в C_i ;

$P\{b_i = m\}$ — стационарное распределение длины очереди в C_i ;

$P\{h_i = m\}$ — стационарное распределение числа занятых приборов в системе C_i .

2 Гибкие производственные системы с маршрутизацией, зависящей от состояния

2.1 Описание модели

Рассмотрим гибкую производственную систему (ГПС), основные компоненты которой следующие.

1. Множество рабочих станций (систем) C_i с номерами из множества $I \equiv \{i \mid i = 1, \dots, L\}$. Каждая рабочая станция C_i производит обработку деталей и может выполнять один или несколько типов производственных операций t (например, сверление, фрезерование, нарезка, растачивание и т.д.). На станции C_i есть κ_i параллельно работающих машин (приборов). Максимальное число деталей, которое допускается в любой момент времени (включая как обрабатываемые детали, так и детали, ожидающие в локальном хранилище), ограничены s_i (емкость рабочей станции), где $s_i \geq \kappa_i$, $i = 1, \dots, L$. Определим вектор числа приборов в рабочих станциях $\kappa = (\kappa_i)$ и вектор емкостей рабочих станций $s = (s_i)$, $i = 1, \dots, L$.

2. Система транспортировки материалов (MHS), обозначаемая как станция C_0 , которая состоит из κ_0 транспортеров (тележки, конвейеры и т.д.), которые осуществляют транспортировку деталей между рабочими станциями, и центрального хранилища. Помимо транспортировки на этой станции выполняется погрузка/разгрузка деталей (на и из палет), закрепление и т.д. То есть вся подготовительная работа, которая делается в период между последовательными посещениями детали двух различных станций).

3. Палеты, на которых перемещаются детали. Для каждой детали выделяется одна палета. Общее количество палет, доступных в ГПС, постоянно и равно N . Для удобства рассуждений будем считать, что $N \leq \sum_{i=1}^L s_i$. В противном случае, в центральном хранилище всегда будут существовать по крайней мере $N - \sum_{i=1}^L s_i$ деталей (эта ситуация может быть учтена в модели исключением соответствующих точек из пространства состояний).

Схема гибкой производственной системы представлена на рисунке 2.1.

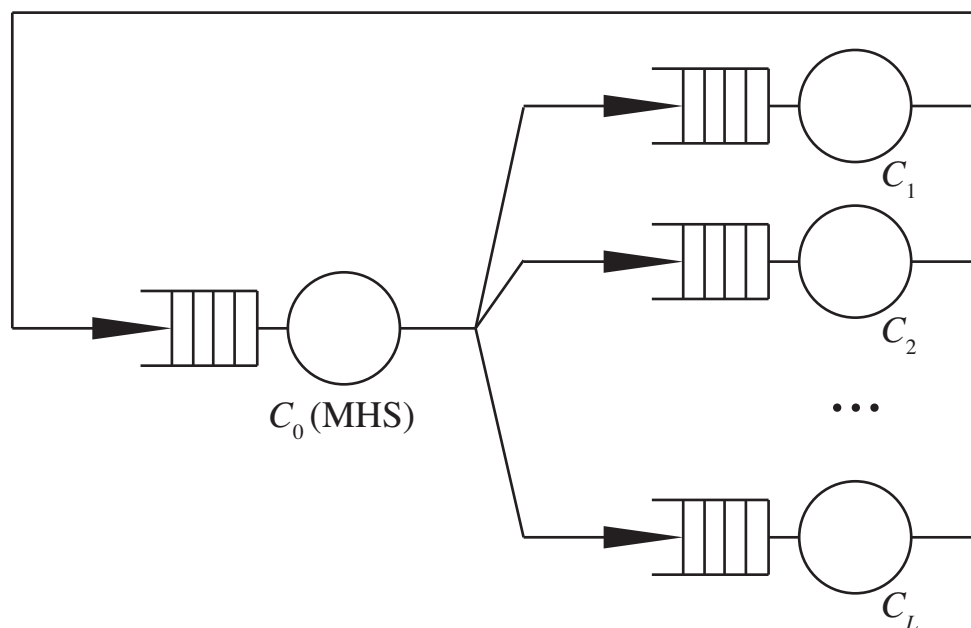


Рисунок 2.1 – Схема гибкой производственной системы

Функционирование гибкой производственной системы происходит следующим образом.

Система должна обработать $t = 1, 2, \dots, T$ типов деталей. Общее число деталей в системе в любой момент времени постоянно и равно N . Другими словами, все доступные палеты все время заняты. Всякий раз, когда обработанная деталь покидает систему, другая деталь того же типа сразу же поступает в систему. Это неявно предполагает, что имеется непрерывное снабжение системы деталями для их обработки. Далее будет рассматриваться только долгосрочное поведение гибкой производственной системы или стационарный режим функционирования. Число палет N_t , выделяемых для деталей типа t , постоянно и $\sum_t N_t = N$. Определим вектор начального числа деталей $\mathbf{N} = (N_t), t = 1, \dots, T$.

Рабочие станции или машины могут быть свободны (простаивать), но они никогда не блокируются. На практике это, как правило, обеспечивается за счет *возвратного конвейера*, который постоянно забирает из рабочих станций детали, завершившие обработку, и доставляет их обратно в центральное хранилище (предполагается, что в центральном хранилище достаточно мест для размещения всех деталей в случае необходимости, т.е. $s_0 = N$). Задержка деталей на возвратном конвейере не учитывается. В этом смысле емкость хранилища возвратного конвейера является частью емкости центрального хранилища.

Деталь каждого типа t нуждается во множестве операций, которые будут выполняться на наборе рабочих станций с номерами $I_t \subseteq I$. Длительность обработки детали типа t на станции C_i , $i \in I_t$, имеет экспоненциальное распределение с параметром μ_{it} . Пусть s_{it} — емкость места хранения, выделенного для деталей типа t на станции C_i . Пусть $s_{it} = 0$, если $i \notin I_t$, тогда $\sum_t s_{it} = s_i$.

Длительность обработки детали на станции C_0 включает в себя время перехода детали на следующую станцию, а также время для погрузки/разгрузки и повторной установки в случае необходимости. Распределение длительности обработки деталей типа t также предполагается экспоненциальным с параметром μ_{0t} .

Дисциплина обработки на всех станциях C_i , $i = 0, \dots, L$, — *RANDOM* (то есть детали выбираются для обработки случайным образом). Для определенности предполагается, что для каждой станции C_i среднее число приборов (механизмов/транспортёров), обрабатывающих детали типа t , пропорционально n_{it} — количеству деталей типа t на станции C_i . Определим вектор размерности T

$$\bar{\eta}_i = (n_{i1}, \dots, n_{iT}), \quad \text{где } n_{it} = 0, \text{ если } i \notin I_t, \text{ и пусть } n_i = \sum_{t=1}^T n_{it}.$$

Вектор $\bar{\eta}_i$ определяет состояние станции C_i как вектор числа деталей различных типов, находящихся на этой станции. Суммарная интенсивность обработки деталей типа t на станции C_i может быть выражена в виде

$$\mu_{it} n_{it} \nu_i(n_i), \quad \text{где } \nu_i(n_i) = \frac{\min(n_i, \kappa_i)}{n_i}.$$

На станции C_0 решения, связанные с маршрутизацией, принимаются всякий раз, когда транспортёр становится доступным. Согласно описанной выше дисциплине обслуживания *RANDOM*, если деталь типа t должна быть доставлена следующей, то для доставки выбирается некоторая станция с номером $i \in I_t$, которая имеет наибольшее число свободных мест, т.е. $s_{it} - n_{it} \geq s_{kt} - n_{kt}$ для всех $k \in I_t$ и $k \neq i$. Одна из деталей типа t , ожидающая в центральном хранилище обработки на станции C_i (эту деталь будем называть «очередной» деталью), затем доставляется на станцию C_i . Эта схема маршрутизации называется детерминированной маршрутизацией в кратчай-

шую очередь или DSQ-маршрутизацией.

Пусть $\Theta = (\theta_{it,jt})$ — маршрутная матрица, $t = 1, \dots, T$, $i, j = 0, \dots, L$, где $\theta_{it,jt}$ — вероятность того, что деталь типа t после обработки на станции C_i поступает на станцию C_j . Заметим, что допускаются переходы только из системы транспортировки на рабочие станции и из рабочих станций в систему транспортировки, т.е. $\theta_{it,jt} = 0$ при $i, j > 0$. Также заметим, что деталь типа t может перемещаться только на рабочую станцию из множества $I_t \subseteq I$, то есть $\theta_{it,jt'} = 0$ при $t \neq t'$.

Примечания

1. Назначение палет и емкости хранилища для различных типов деталей (например, N_t и s_{it}) будет непосредственно влиять на маршрутизацию и, следовательно, пропускную способность для каждого типа деталей. Выбор этих параметров должен быть согласован с желаемой пропускной способностью типов деталей.

2. Две главных цели гибких производственных систем состоят в том, чтобы уменьшить в хранилище число необработанных деталей и увеличить коэффициент использования приборов на станциях. Известно, что с использованием приборов главным образом связаны два фактора: блокирование и простаивание (здесь не рассматриваются проблемы надежности). Ранее сделанные предположения о возвратном конвейере и центральном хранилище позволяют увеличить коэффициент использования приборов за счет исключения возможности их блокирования. С другой стороны, также известно, что простаивание прибора в первую очередь влияет на интенсивность потока и загрузку каждой станции. Локальные буферы могут уменьшить влияние простаивания прибора, но, конечно, не в значительной мере. Тогда, в целях сокращения в хранилище числа необработанных деталей, предполагается наличие ограниченных локальных буферов на станциях. Исследования существующих гибких производственных систем показывают, что большинство из них имеют очень маленькие локальные буферы. Некоторые производственные системы, которые используют замкнутые конвейеры в качестве системы транспортировки материалов, имеют нулевые локальные буферы: ожидающие детали просто стоят на конвейере, который функционирует и как устройство транспортировки материалов, и как центральное хранилище.

3. В рассматриваемой в данной работе модели маршрутизация деталей

определяется для *потока* деталей в рабочую станцию (на обрабатывающие приборы), а не для *отдельных* деталей. Всякий раз, когда решение, связанное с маршрутизацией, должно быть принято, сначала определяется соответствующая рабочая станция (прежде всего те, которые содержат наименьшее число деталей), а затем некоторая деталь «вытаскивается» из центрального хранилища для доставки на эту станцию.

Данная гибкая производственная система с введенными выше предположениями описывается неоднородной замкнутой экспоненциальной сетью массового обслуживания $\Gamma = \langle L, T, \mathbf{N}, N, M, \Theta, \kappa, \mu, RANDOM \rangle$. В дальнейшем термины «гибкая производственная система», «рабочая станция», «обрабатывающий прибор», «детали», «обработка деталей», используемые для описания гибких производственных систем, будем отождествлять соответственно с терминами «сеть массового обслуживания», «система», «обслуживающий прибор», «требования», «обслуживание требований».

2.2 Решение уравнения равновесия

Определим состояние сети Γ , соответствующей рассмотренной выше гибкой производственной системе, как $\bar{\eta} = (\bar{\eta}_0, \bar{\eta}_1, \dots, \bar{\eta}_L)$, где $\bar{\eta}_i = (n_{i1}, \dots, n_{iT})$, $i = 0, 1, \dots, L$, обозначает вектор числа требований в системе C_i . Из предположений в разделе 2.1 нетрудно видеть, что $\{\bar{\eta}(\tau)\}$ определяет процесс Маркова со следующим конечным пространством состояний:

$$S = \left\{ \bar{\eta} \in Z_+^{(L+1)T} \mid n_{it} \leq s_{it} \ (i \in I), \sum_{i \in I_t^+} n_{it} = N_t, \ t = 1, \dots, T \right\}, \quad (2.1)$$

где Z_+ обозначает множество неотрицательных чисел и $I_t^+ = \{0\} \cup I_t$.

В дальнейшем внесем элемент случайности в детерминированную маршрутизацию в кратчайшую очередь (DSQ–маршрутизацию), при которой требования направляются в кратчайшую очередь с наибольшей вероятностью (с вероятностью один), с помощью PSQ–маршрутизации. Установлено, что PSQ–маршрутизация — точное приближение DSQ–маршрутизации при условии, что емкости систем намного меньше общего числа требований в сети N (которое, как правило, справедливо для реальных производственных систем) [5].

Сформулируем PSQ–маршрутизацию следующим образом. Вероятности

перехода требований класса t из системы C_0 в систему C_i , $i \in I_t$, зависят от n_{0t} и от n_{it} — числа требований класса t в двух системах, и принимают форму

$$\theta_{0t,it} = \frac{r_{it}(n_{it})}{r_{0t}(n_{0t})}, \quad (2.2)$$

где $r_{it}(\cdot)$ и $r_{0t}(\cdot)$ — две линейные функции:

$$r_{it}(n_{it}) = s_{it} - n_{it} \text{ и } r_{0t}(n_{0t}) = \sum_{C_i \in I_t} s_{it} + n_{0t} - N_t.$$

Заметим, что $r_{it}(\cdot)$ показывают число свободных мест в системе C_i для требований класса t , а $r_{0t}(n_{0t}) = \sum_{C_i \in I_t} r_{it}(n_{it})$ такие, что маршрутные вероятности в сумме по всем $i \in I_t$ равны единице. Несложно заметить следующие особенности этой схемы маршрутизации:

- маршрутные вероятности выше для систем с бóльшим числом свободных приборов;
- требования класса t никогда (т.е. с вероятностью ноль) не направляются в систему, в которой все места в очереди для ожидания требованиями этого класса заняты (т.е. когда $n_{it} = s_{it}$).

Для того, чтобы получить решение для сети Γ как модели гибкой производственной системы, понадобятся следующие предварительные сведения об обратимых марковских процессах.

Определение 1. Случайный процесс $\{X(\tau)\}$ (время $\tau \in T$ может быть непрерывным или дискретным), определенный в пространстве состояний S (конечном или счетном), является обратимым, если $\{X(\tau_1), \dots, X(\tau_n)\}$ имеет то же распределение, что и $\{X(\tau_0 - \tau_1), \dots, X(\tau_0 - \tau_n)\}$ для всех $\tau_0, \tau_1, \dots, \tau_n \in T$ [9].

Обратимый процесс, как известно, является стационарным. Для стационарных марковских процессов имеет место следующее утверждение.

Лемма 1. Стационарный марковский процесс $\{X(\tau)\}$ является обратимым, если существует положительный набор чисел $\pi(j)$, $j \in S$, в сумме дающий единицу, который удовлетворяет следующим уравнениям равновесия:

$$\pi(j)q(j, k) = \pi(k)q(k, j) \quad \forall j, k \in S, \quad (2.3)$$

где $q(j, k)$ — интенсивность перехода из состояния j в состояние k . Если существует такой набор $\pi(j), j \in S$, то он является стационарным распределением процесса $X(\tau)$ [9].

Стационарное решение для сети Γ как модели гибкой производственной системы можно теперь обобщить следующим образом.

Теорема 1. Марковский процесс $\bar{\eta}(\tau)$, определенный в пространстве состояний S и управляемый PSQ-маршрутизацией, как определено в (2.2), является обратимым относительно времени и имеет следующую мультипликативную форму стационарного распределения вероятностей:

$$\pi(\bar{\eta}) = G^{-1} \prod_{i=0}^L \left[\prod_{j=1}^{n_i} \nu_i^{-1}(j) \right] \left[\prod_{t=1}^T \prod_{j=1}^{n_{it}} \frac{r_{it}(j-1+\delta_{i0})}{j\mu_{it}} \right], \quad \bar{\eta} \in S, \quad (2.4)$$

где $\delta_{i0} = 1$, если $i = 0$, иначе $\delta_{i0} = 0$, и G — нормализующая константа.

Доказательство. Пусть вектор $\bar{\eta}'$ обозначает состояние сети Γ с компонентами

$$\bar{\eta}'_0 = (n_{01}, \dots, n_{0u} - 1, \dots, n_{0T}), \quad \bar{\eta}'_k = (n_{k1}, \dots, n_{ku} + 1, \dots, n_{kT}),$$

а все остальные компоненты такие же как у вектора $\bar{\eta}$ (т.е. требование класса u перешло из системы C_0 в систему C_k). Основываясь на лемме, достаточно показать, что определенное в (2.4) распределение $\pi(\bar{\eta})$ (для всех $\bar{\eta} \in S$) удовлетворяет следующим уравнениям равновесия: для $\bar{\eta}, \bar{\eta}' \in S$

$$\pi(\bar{\eta}) [n_{0u} \mu_{0u} \nu_0(n_0)] \left[\frac{r_{ku}(n_{ku})}{r_{0u}(n_{0u})} \right] = \pi(\bar{\eta}') [(n_{ku} + 1) \mu_{ku} \nu_k(n_k + 1)], \quad (2.5)$$

где величины в квадратных скобках левой части дают вероятность перехода сети из состояния $\bar{\eta}$ в состояние $\bar{\eta}'$, а величины в квадратных скобках правой части дают вероятность перехода сети из состояния $\bar{\eta}'$ в состояние $\bar{\eta}$. Выражение (2.5) эквивалентно следующему:

$$\pi(\bar{\eta}') \left[\frac{r_{0u}(n_{0u})}{n_{0u} \mu_{0u} \nu_0(n_0)} \right] \left[\frac{(n_{ku} + 1) \mu_{ku} \nu_k(n_k + 1)}{r_{ku}(n_{ku})} \right] = \pi(\bar{\eta}). \quad (2.5a)$$

Из (2.4) запишем $\pi(\bar{\eta})$ как

$$\pi(\bar{\eta}) = G^{-1} \prod_{i=0}^L F_i(\bar{\eta}_i),$$

где $F_i(\bar{\eta}_i)$ обозначают величины в двух квадратных скобках правой части выражения (2.4). Тогда $\pi(\bar{\eta}')$ может быть соответственно выражено следующим образом:

$$\pi(\bar{\eta}') = G^{-1} \prod_{i \neq 0, k} F_i(\bar{\eta}_i) [F_0(\bar{\eta}'_0) F_k(\bar{\eta}'_k)],$$

где

$$F_0(\bar{\eta}'_0) = \prod_{j=1}^{n_0-1} \nu_0^{-1}(j) \cdot \left[\prod_{t \neq u} \prod_{j=1}^{n_{0t}} \frac{r_{0t}(j)}{j \mu_{0t}}, \prod_{j=1}^{n_{0u}-1} \frac{r_{0u}(j)}{j \mu_{0u}} \right],$$

$$F_k(\bar{\eta}'_k) = \prod_{j=1}^{n_k+1} \nu_k^{-1}(j) \cdot \left[\prod_{t \neq u} \prod_{j=1}^{n_{kt}} \frac{r_{kt}(j-1)}{j \mu_{kt}}, \prod_{j=1}^{n_{ku}+1} \frac{r_{ku}(j-1)}{j \mu_{ku}} \right].$$

Тогда очевидно, что

$$F_0(\bar{\eta}'_0) \cdot \left[\frac{r_{0u}(n_{0u})}{n_{0u} \mu_{0u} \nu_0(n_0)} \right] = F_0(\bar{\eta}_0)$$

$$F_k(\bar{\eta}'_k) \cdot \left[\frac{(n_{ku} + 1) \mu_{ku} \nu_k(n_k + 1)}{r_{ku}(n_{ku})} \right] = F_k(\bar{\eta}_k)$$

и, следовательно, равенство (2.5а) верно. Таким образом, теорема доказана. \square

3 Алгоритм метода анализа производственных систем с маршрутизацией, зависящей от состояния

3.1 Описание алгоритма

Рассмотрим однородную замкнутую экспоненциальную сеть массового обслуживания $\Gamma' = \langle L, 1, N, M, \Theta, \kappa, \mu, FCF S \rangle$ с L системами $C_i, i = 1, \dots, L$, (включая систему транспортировки материалов, которая теперь обозначается как любая C_i) и N требованиями (палетами в ГПС). Сначала рассмотрим случай одного класса требований ($T = 1$). Обобщение на случай нескольких классов требований рассматривается в конце раздела.

При $T = 1$ будем опускать индекс t , выразим $\pi(\bar{\eta})$ следующим образом:

$$\pi(\bar{\eta}) = G^{-1}(L, N) \prod_{i=1}^L F_i(n_i) = G^{-1}(L, N) \prod_{i=1}^L \prod_{j=1}^{n_i} f_i(j), \quad (3.1)$$

где $G(L, N)$ — нормализующая константа, а $f_i(\cdot)$ — множитель $F_i(\cdot)$, равный

$$f_i(j) = \nu_i^{-1}(j) \frac{r_i(j-1 + \delta_{i0})}{j\mu_i}.$$

При $T = 1$ дисциплина обслуживания эквивалентна дисциплине «первым пришел — первым обслужен» (FCFS).

Кроме того, обозначим через $G(L-1, N)$ нормализующую константу, связанную с сетью из $L-1$ системы, где L -ая система удалена. Пусть

$$R(L, N) = \frac{G(L, N-1)}{G(L, N)}$$

и определим аналогичным образом $R(L-1, N)$.

Не теряя общности, предположим, что требуется получить предельное маргинальное (частное) распределение вероятностей $\pi_L(k, N)$ ($k = 0, 1, \dots, s_L$) для системы C_L сети Γ' с L системами и N требованиями. Имеют место следующие рекуррентные формулы.

Предложение 1.

$$\pi_L(0, N) = \pi_L(0, N-1) R^{-1}(L-1, N) R(L, N), \quad (3.2)$$

$$\pi_L(k, N) = \pi_L(k - 1, N - 1)f_L(k)R(L, N) \quad (k = 1, 2, \dots, s_L). \quad (3.3)$$

Доказательство. Из (3.1) и обозначений, определенных выше, имеем

$$\begin{aligned} \pi_L(0, N) &= \frac{G(L - 1, N)}{G(L, N)} = \frac{G(L - 1, N - 1)}{G(L, N - 1)} \cdot \frac{G(L - 1, N)}{G(L - 1, N - 1)} \cdot \frac{G(L, N - 1)}{G(L, N)} = \\ &= \pi_L(0, N - 1)R^{-1}(L - 1, N)R(L, N). \end{aligned}$$

Для $k = 1, 2, \dots, s_L$ имеем

$$\begin{aligned} \pi_L(k, N) &= F_L(k) \cdot \frac{G(L - 1, N - k)}{G(L, N)} = \\ &= f_L(k)F_L(k - 1) \cdot \frac{G(L - 1, N - 1 - (k - 1))}{G(L, N - 1)} \cdot \frac{G(L, N - 1)}{G(L, N)} = \\ &= \pi_L(k - 1, N - 1)f_L(k)R(L, N). \end{aligned}$$

□

На основе приведенных выше результатов разработан алгоритм, аналогичный алгоритму нормализованной свертки Райзера [8], который предназначен в основном для замкнутых сетей с постоянной маршрутизацией требований и неограниченными очередями в системах сети.

Предположим, что мы хотим проанализировать сеть с N требованиями и L системами. Для рекурсивного вычисления $R(L, N)$ и π_L имеется три цикла: цикл I по числу систем от $m = 1$ до $m = L$;

цикл II для каждого m по числу требований от $n = 0$ до $n = \min(N, \sum_{i=1}^m s_i)$;

цикл III для каждого n по общему числу требований в системе C_m от $k = 0$ до $k = \min(s_m, n)$.

Заметим, что II и III цикл здесь отличаются от циклов в алгоритме свертки Райзера. Пределы здесь также ограничены: $R(1, n) = f_1^{-1}(n)$, $n = 1, \dots, s_1$. Как и в алгоритме свертки Райзера, при расчете значений π_m итоговая константа $R(m, n)$ сначала может не учитываться, а затем может быть рассчитана

как нормирующий множитель:

$$R(m, n) = \left[\sum_{k=0}^{s_m} \pi_m(k, n) \right]^{-1} \quad (\pi_m\text{-ые в сумме ненормированные}).$$

Таким образом, $R(L, n)$ и π_L -ые по-прежнему могут быть получены одновременно. После нахождения $R(L, n)$ ($n = 1, \dots, N$) сразу получаем

$$G(L, N) = \prod_{n=1}^N R^{-1}(L, n). \quad (3.4)$$

Алгоритм анализа однородной сети следующий:

Шаг 0. (Инициализация)

$$m = 1$$

$$NN_1 = s_1$$

Для $n = 1$ до NN_1 : $R(n) = f_1^{-1}(n)$: следующее n .

Шаг 1. (Цикл I)

$$m = m + 1$$

$$NN_m = \min(N, NN_{m-1} + s_m)$$

$$\pi(0) = 1$$

Шаг 2. (Цикл II)

Для $n = 1$ до NN_m

$$\text{sum} = 0$$

$$kk = \min(s_m, n)$$

(Цикл III)

Для $k = kk$ до 1

$$\pi(k) = \pi(k-1) * f_m(k)$$

$$\text{sum} = \text{sum} + \pi(k)$$

следующее k .

Если $n > NN_{m-1}$, то $\pi(0) = 0$, иначе $\pi(0) = \pi(0)/R(n)$

$$R(n) = 1/[\text{sum} + \pi(0)]$$

Для $k = 0$ до kk : $\pi(k) = \pi(k) * R(n)$: следующее k

следующее n .

Шаг 3. Если $m < L$ то перейти на шаг 1

иначе вернуть результат. Останов.

Как и в алгоритме нормализованной свертки Райзера, алгоритм использует два одномерных массива: $R(n)$ (размерности $N + 1$) и $\pi(k)$ (размерности $\max_i s_i + 1$).

В общем случае, для сети массового обслуживания с несколькими классами требований определим векторы размерности T : $\mathbf{N} = (N_1, \dots, N_T)$, $\mathbf{n} = (n_1, \dots, n_T)$, $\mathbf{e}_t = (0, \dots, 1, \dots, 0)$ (t -ый элемент равен 1, а все остальные нули) и $\mathbf{0} = (0, \dots, 0)$. Пусть $G(m, \mathbf{N})$ будет нормализующей константой, где \mathbf{N} — вектор начального числа требований в сети. Кроме того, определим

$$R_t(m, \mathbf{N}) = \frac{G(m, \mathbf{N} - \mathbf{e}_t)}{G(m, \mathbf{N})}, \quad t = 1, \dots, T.$$

Обозначим через $\pi_m(\mathbf{n}, \mathbf{N})$ предельную вероятность того, что общее число требований в системе m будет равно \mathbf{n} , когда общее число требований всей сети равно \mathbf{N} . Тогда справедливо следующее следствие.

Следствие 1. Если $\mathbf{n} > \mathbf{0}$ (т.е. все элементы вектора \mathbf{n} положительны), то для всех $t = 1, \dots, T$

$$\pi_m(\mathbf{n}, \mathbf{N}) = \pi_m(\mathbf{n} - \mathbf{e}_t, \mathbf{N} - \mathbf{e}_t) f_{mt}(\mathbf{n}) R_t(m, \mathbf{N}), \quad (3.5)$$

иначе для всех $t = 1, \dots, T$

$$\pi_m(\mathbf{n}, \mathbf{N}) = \pi_m(\mathbf{n}, \mathbf{N} - \mathbf{e}_t) R_t^{-1}(m - 1, \mathbf{N} - \mathbf{n}) R_t(m, \mathbf{N}). \quad (3.6)$$

Разработанный ранее алгоритм применим и здесь со следующими изменениями.

1. Необходимы два двухмерных массива для $\pi(\mathbf{n})$ и R_t , $t = 1, \dots, T$.
2. В этом случае имеется T нормализующих множителей R_t , $t = 1, \dots, T$, которые в дополнение к условию нормировки значений π_m удовлетворяют следующим уравнениям:

$$R_t(m, \mathbf{N}) = R_t(m - 1, \mathbf{N}) \frac{\pi_m(\mathbf{0}, \mathbf{N})}{\pi_m(\mathbf{0}, \mathbf{N} - \mathbf{e}_t)}, \quad t = 1, \dots, T. \quad (3.7)$$

Эти уравнения следуют непосредственно из (3.6) при предположении, что $\mathbf{n} = \mathbf{0}$.

Алгоритм для неоднородной сети приведен в следующем разделе.

3.2 Структурная схема алгоритма

Алгоритм метода анализа сети массового обслуживания Γ с маршрутизацией, зависящей от состояния, имеет блочную структуру, представленную на рисунке 3.1.

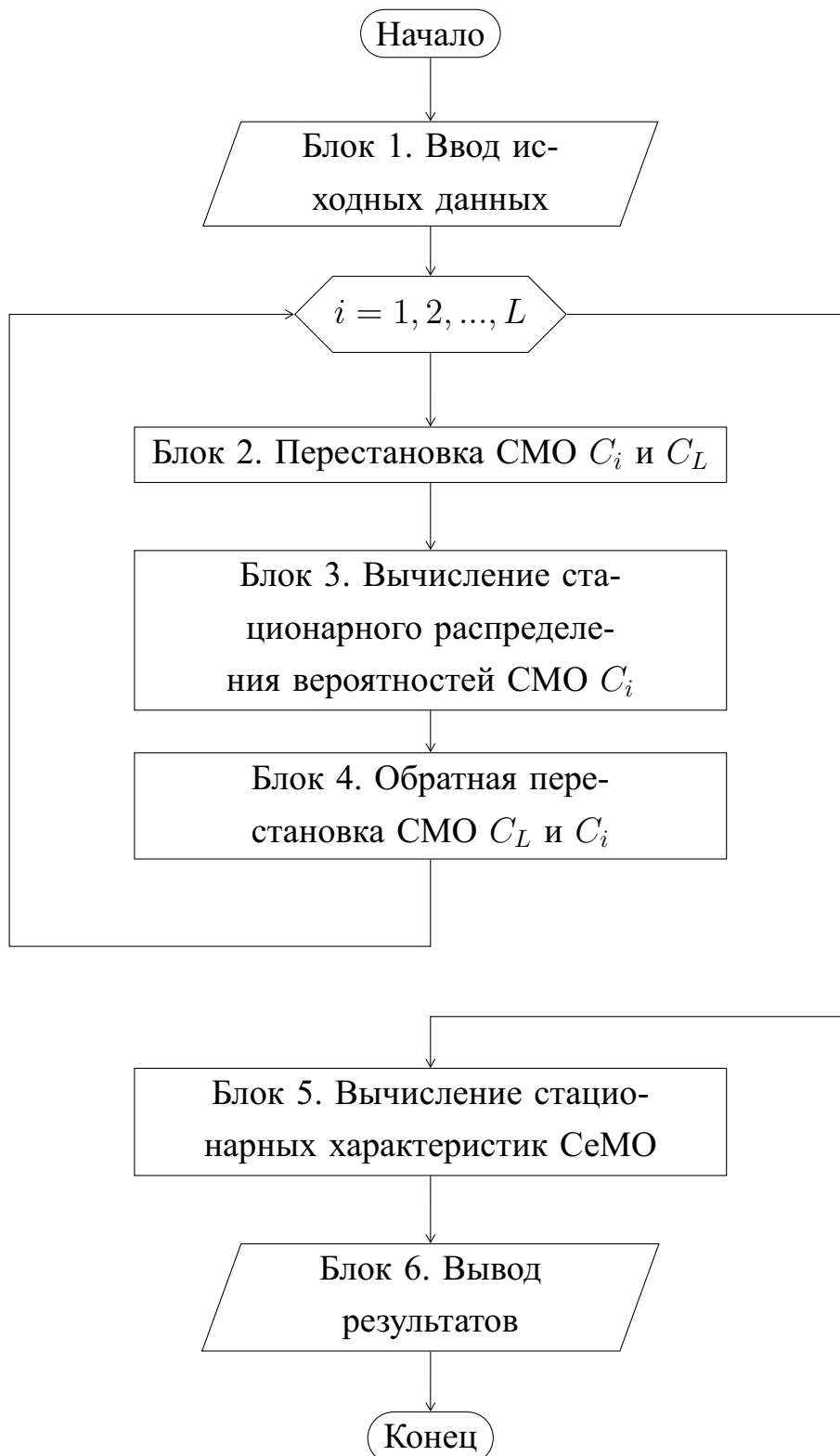


Рисунок 3.1 – Блок-схема алгоритма метода анализа сетей массового обслуживания с маршрутизацией, зависящей от состояния

Блок 1. Ввод исходных данных

На начальном этапе работы алгоритма вводятся параметры сети массового обслуживания Γ :

L — число СМО в СеМО;

$\mathbf{N} = (N_t)$ — вектор начального числа требований в СеМО, $t = 1, \dots, T$;

$\kappa = (\kappa_i)$ — вектор числа приборов в системах обслуживания СеМО, $i = 0, \dots, L$;

$s = (s_{it})$ — матрица емкостей систем в СеМО, $i = 0, \dots, L$, $t = 1, \dots, T$;

$\mu = (\mu_{it})$ — матрица интенсивностей обслуживания требований системами СеМО, $i = 0, \dots, L$, $t = 1, \dots, T$.

Блок 2. Перестановка СМО C_i и C_L

Во втором блоке для вычисления стационарного распределения вероятностей состояний системы C_i , $i = 1, \dots, L$, происходит перестановка системы C_i с последней системой C_L . Это осуществляется путем перестановки элементов κ_i и κ_L в векторе κ , s_{it} и s_{iL} в матрице s , μ_{it} и μ_{iL} в матрице μ , $i = 1, \dots, L$, $t = 1, \dots, T$.

$$\text{Входные данные: } \kappa = (0, \dots, \kappa_i, \dots, \kappa_L), s = \begin{pmatrix} s_{1t} \\ \dots \\ s_{it} \\ \dots \\ s_{Lt} \end{pmatrix}, \mu = \begin{pmatrix} \mu_{1t} \\ \dots \\ \mu_{it} \\ \dots \\ \mu_{Lt} \end{pmatrix}, t = 1, \dots, T.$$

$$\text{Выходные данные: } \kappa' = (0, \dots, \kappa_L, \dots, \kappa_i), s' = \begin{pmatrix} s_{1t} \\ \dots \\ s_{Lt} \\ \dots \\ s_{it} \end{pmatrix}, \mu' = \begin{pmatrix} \mu_{1t} \\ \dots \\ \mu_{Lt} \\ \dots \\ \mu_{it} \end{pmatrix}, t = 1, \dots, T.$$

Блок 3. Вычисление стационарного распределения вероятностей состояний СМО C_i

Алгоритм для вычисления стационарного распределения вероятностей состояний для системы C_i следующий:

Входные данные: $L, T, \mathbf{N} = (N_t), \kappa = (\kappa_i), s = (s_{it}), \mu = (\mu_{it}), i = 0, \dots, L, t = 1, \dots, T$.

Шаг 0.

$$t = 0$$

$$m = 0$$

Шаг 1. (Цикл I)

$$t = t + 1$$

$$NN_{t0} = s_{0t}$$

Для $n = 1$ до NN_{t0} : $R_t(n) = f_{t0}^{-1}(n)$: следующее n .

Шаг 2. (Цикл II)

$$m = m + 1$$

$$NN_{tm} = \min(N_t, N_{t,m-1} + s_{mt})$$

$$\pi_i(0, N_t) = 1$$

Шаг 3. (Цикл III)

Для $n = 1$ до NN_{tm}

$$\text{sum} = 0$$

$$kk = \min(s_{mt}, n)$$

(Цикл IV)

Для $k = kk$ до 1

$$\pi_i(k, N_t) = \pi_i(k - 1, N_t) * f_{tm}(k)$$

$$\text{sum} = \text{sum} + \pi_i(k, N_t)$$

следующее k .

Если $n > NN_{t,m-1}$, то $\pi_i(0, N_t) = 0$, иначе $\pi_i(0, N_t) = \pi_i(0, N_t) / R_t(n)$

$$R_t(n) = 1 / [\text{sum} + \pi_i(0, N_t)]$$

Для $k = 0$ до kk : $\pi_i(k, N_t) = \pi_i(k, N_t) * R_t(n)$: следующее k

следующее n .

Шаг 4. Если $m < L$ то перейти на шаг 3

Шаг 5. Если $t < T$ то перейти на шаг 2

иначе вернуть результат. Останов.

Функция $f_{tm}(k)$ в алгоритме определяется как

$$f_{tm}(k) = \nu_m^{-1}(k) \frac{r_{it}(k - 1 + \delta_{m0})}{k\mu_{mt}}.$$

Выходные данные: $\pi_i(\mathbf{n}, \mathbf{N})$, $i = 1, \dots, L$.

Блок 4. Обратная перестановка СМО C_L и C_i

Происходит обратная перестановка системы C_L с системой C_i . Таким образом, получаем исходный вектор κ и матрицы s, μ .

$$\begin{aligned}
\text{Входные данные: } \kappa' &= (0, \dots, \kappa_L, \dots, \kappa_i), \quad s' = \begin{pmatrix} s_{1t} \\ \dots \\ s_{Lt} \\ \dots \\ s_{it} \end{pmatrix}, \quad \mu' = \begin{pmatrix} \mu_{1t} \\ \dots \\ \mu_{Lt} \\ \dots \\ \mu_{it} \end{pmatrix}, \quad t = 1, \dots, T. \\
\text{Выходные данные: } \kappa &= (0, \dots, \kappa_i, \dots, \kappa_L), \quad s = \begin{pmatrix} s_{1t} \\ \dots \\ s_{it} \\ \dots \\ s_{Lt} \end{pmatrix}, \quad \mu = \begin{pmatrix} \mu_{1t} \\ \dots \\ \mu_{it} \\ \dots \\ \mu_{Lt} \end{pmatrix}, \quad t = 1, \dots, T.
\end{aligned}$$

Блок 5. Вычисление стационарных характеристик СеМО

На данном этапе происходит вычисление следующих стационарных характеристик СеМО:

- м. о. числа t -требоаний в СМО;
- м. о. числа занятых t -требованиями приборов в СМО;
- интенсивность входящего потока t -требоаний в СМО;
- коэффициенты использования обслуживающих приборов СМО t - требованиями.

Эти характеристики вычисляются по формулам (3.8) – (3.14).

$$\bar{n}_{it} = \sum_{k=0}^{s_{it}} k \pi_i(\mathbf{n}, \mathbf{N}), \quad (3.8)$$

$$\bar{h}_{it} = \begin{cases} \sum_{k=0}^{\kappa_i} k \pi_i(\mathbf{n}, \mathbf{N}) + \kappa_i \sum_{k=\kappa_i+1}^{s_{it}} \pi_i(\mathbf{n}, \mathbf{N}), & \kappa_i < s_{it}, \\ \sum_{k=0}^{s_{it}} k \pi_i(\mathbf{n}, \mathbf{N}), & \kappa_i \geq s_{it}, \end{cases} \quad (3.9)$$

$$\lambda_{it} = \mu_{it} \bar{h}_{it}, \quad (3.10)$$

$$\psi_{it} = \frac{\lambda_{it}}{\min(\kappa_i, s_{it}) \mu_{it}}, \quad (3.11)$$

где $i = 1, \dots, L$, $t = 1, \dots, T$.

$$\bar{n}_{0t} = N_t - \sum_{i=1}^L \bar{n}_{it}, \quad (3.12)$$

$$\lambda_{0t} = \sum_{i=1}^L \lambda_{it}, \quad (3.13)$$

$$\psi_{0t} = \frac{\lambda_{0t}}{\kappa_0 \mu_{0t}}, \quad (3.14)$$

где $t = 1, \dots, T$.

Входные данные: $L, T, \mathbf{N} = (N_t), \pi_m(\mathbf{n}, \mathbf{N}), \kappa = (\kappa_i), s = (s_{it}), \mu = (\mu_{it}), i = 0, \dots, L, m = 1, \dots, L, t = 1, \dots, T$.

Выходные данные: $\bar{n}_{it}, \lambda_{it}, \psi_{it}, i = 0, \dots, L, t = 1, \dots, T$.

Блок 6. Вывод результатов

В данном блоке происходит вывод (на экран или в файл) стационарного распределения и стационарных характеристик, полученных в блоке 4 и 5.

4 Описание и назначение программы

Программа, предназначенная для анализа производственных систем с маршрутизацией, зависящей от состояния, была реализована на языке программирования Java на платформе JDK 7 с использованием стандартной библиотеки для создания графического интерфейса Swing.

Программа позволяет вычислить стационарное распределение и основные характеристики производственных систем с маршрутизацией, зависящей от состояния. Вычисления могут производиться как для однородной, так и для неоднородной сети массового обслуживания. В случае необходимости произвести вычисления для однородной сети, в окне программы или во входном файле необходимо положить $T = 1$.

4.1 Список идентификаторов

В таблице 4.1 приведены идентификаторы, используемые в программе. Здесь $i = 0, \dots, L$, где C_0 — система транспортировки в ГПС; $m = 1, \dots, L$; $t = 1, \dots, T$; $k = 1, \dots, s_{it}$; $\mathbf{N} = (N_1, \dots, N_T)$; $\mathbf{n} = (n_1, \dots, n_T)$.

Таблица 4.1 – Список идентификаторов

Обозначение в алгоритме	Содержание	Обозначение в программе
L	Число СМО в сети	L
T	Число классов требований в СеМО	T
N	Общее число требований в СеМО	$NTotal$
$\mathbf{N} = (N_t)$	Вектор числа t -требований в сети	$N[]$
N_t	Число t -требований в сети	$N[t]$
κ_i	Число обслуживающих приборов в СМО C_i	$kappa[i]$
s_{it}	Емкость системы C_i для t -требований	$s[i][t]$

Продолжение таблицы 4.1

Обозначение в алгоритме	Содержание	Обозначение в программе
μ_{it}	Интенсивность обслуживания в системе C_i t -требуваний	$mu[i][t]$
\bar{n}_{it}	М. о. числа t -требуваний в C_i	$charN[i][t]$
\bar{h}_{it}	М. о. числа занятых t -требуваниями приборов в C_i	h
λ_{it}	Интенсивность потока t -требуваний в C_i	$charLambda[i][t]$
ψ_{it}	Коэффициент использования обслуживающих приборов системы C_i t -требуваниями	$charPsi[i][t]$
$\pi_m(\mathbf{n}, \mathbf{N})$	Стационарное распределение	$PI[m][t][k]$
$R_t(m, \mathbf{N})$	Нормирующий множитель	$R[t][n_m]$
$\nu_i(n_i)$	$\nu_i(n_i) = \min(n_i, \kappa_i)/n_i$	nu
$r_{it}(n_{it})$	Число свободных мест в системе C_i для t -требуваний	r

4.2 Описание программы

Исходный код программы представляет собой три класса: *AnalysisFMS*, *AnalysisQueueingNetwork* и *ResultFrame*, которые расположены в трех файлах с соответствующим именем и расширением *.java*. *AnalysisFMS* является главным классом, с которого начинается старт программы. В нем описаны процедуры запуска главного окна, считывания данных из файла и из формы для последующей передачи на анализ. В классе *AnalysisQueueingNetwork* происходит проверка корректности данных и анализ сети. Класс *ResultFrame* содержит процедуры запуска окна результата, вывода результатов анализа на экран, а также сохранения в файл.

Разработанная программа имеет графический интерфейс. Входные дан-

ные считываются с формы, проверяются на корректность, и в соответствии с проверкой либо производится анализ, либо выдается сообщение об ошибке. Для рассматриваемой сети входными данными являются: число систем, число классов требований, вектор числа требований определенного класса, вектор числа обслуживающих приборов, емкости систем и интенсивности обслуживания. Выходными данными являются стационарное распределение и основные стационарные характеристики систем, а именно: м. о. числа требований, интенсивности потока требований и коэффициенты использования обслуживающих приборов.

При запуске программы появляется окно, изображенное на рисунке 4.1.

Анализ гибких производственных систем

Число СМО (L): Число классов требований (T):

Вектор числа t-требований (N[t]):

Вектор числа приборов (каппа[i]):

Емкости систем (s[i][t]):		Интенсивности обслуживания (mu[i][t]):	
20	30	3.0	3.0
4	0	2.0	0.0
4	0	2.0	0.0
4	0	2.0	0.0
0	6	0.0	1.5
0	6	0.0	1.5
0	6	0.0	1.5
3	4	1.0	0.75
3	4	1.0	0.75
3	4	1.0	0.75

Рисунок 4.1

Для удобства существует возможность открыть файл с заданными в нем входными данными (см. рисунок 4.2).

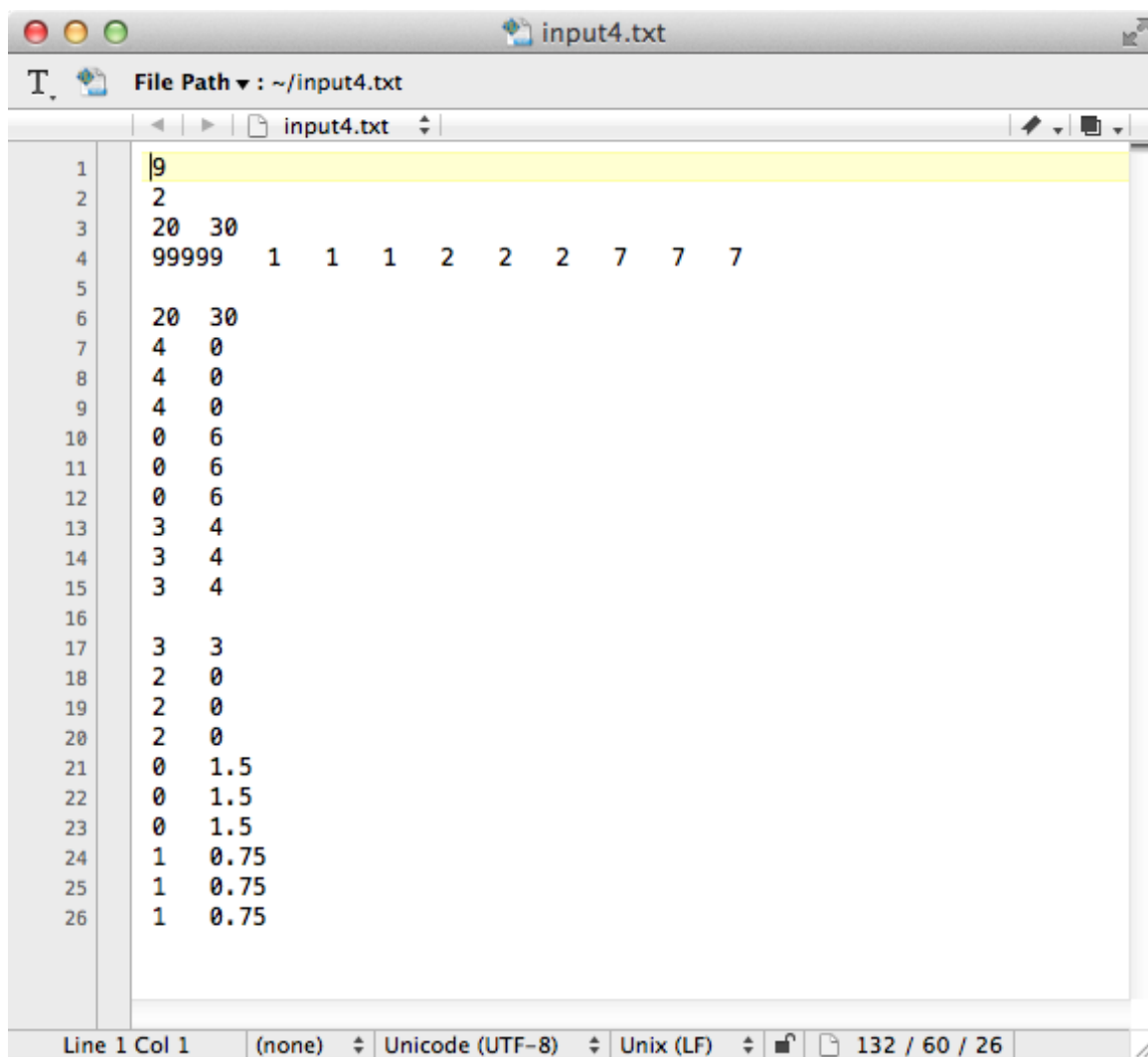


Рисунок 4.2

Для анализа при заданных входных данных служит кнопка «Получить результаты». При ее нажатии открывается окно с подсчитанными в ходе анализа основными характеристиками СМО и стационарным распределением (см. рисунок 4.3).

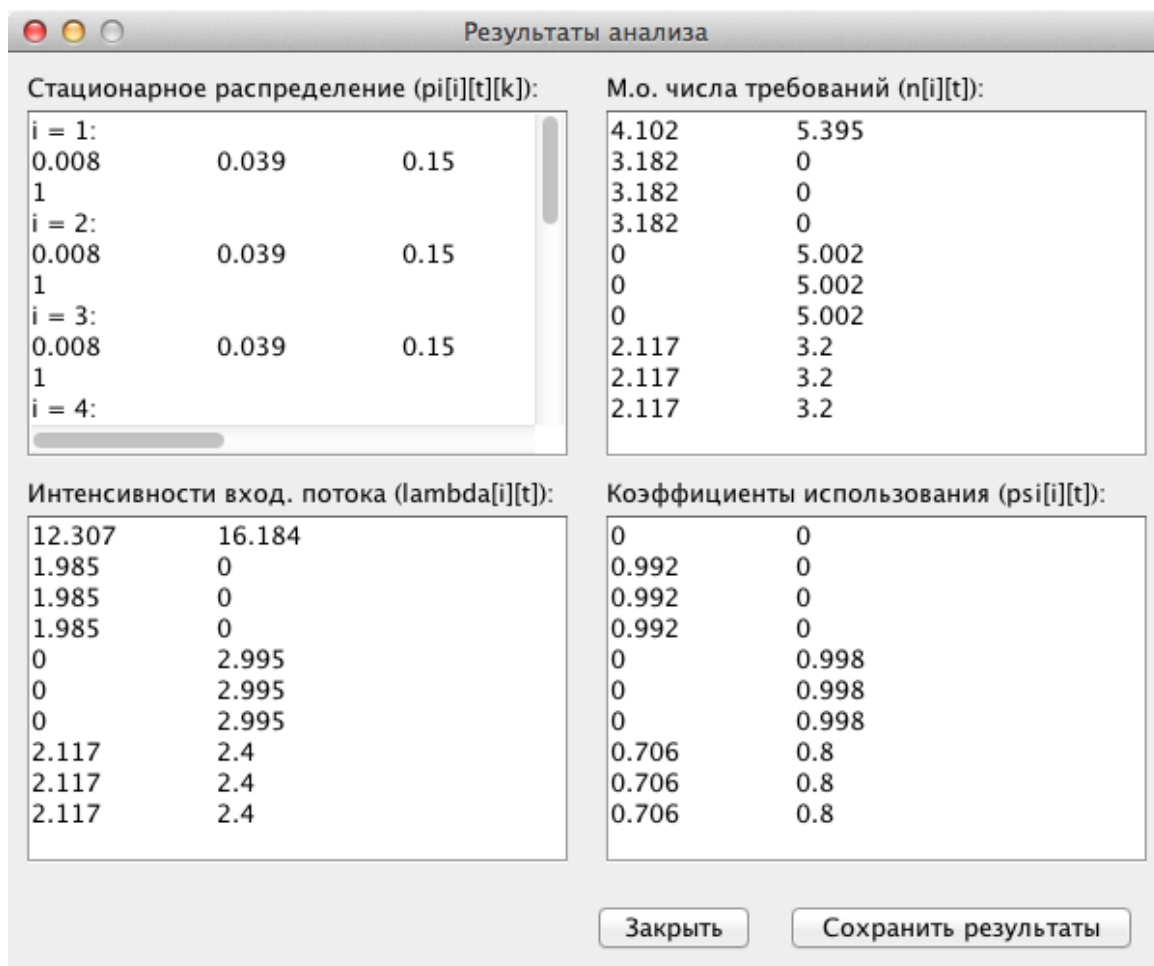


Рисунок 4.3

В случае введения некорректных начальных данных, на экран будет выведено сообщение об ошибке (см. рисунок 4.4).

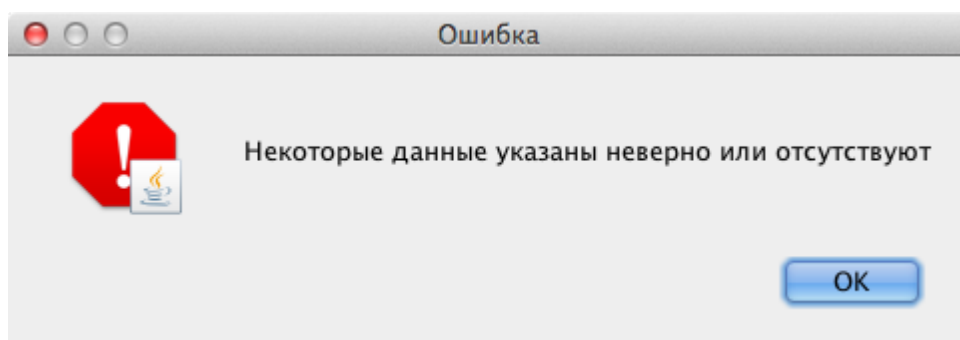
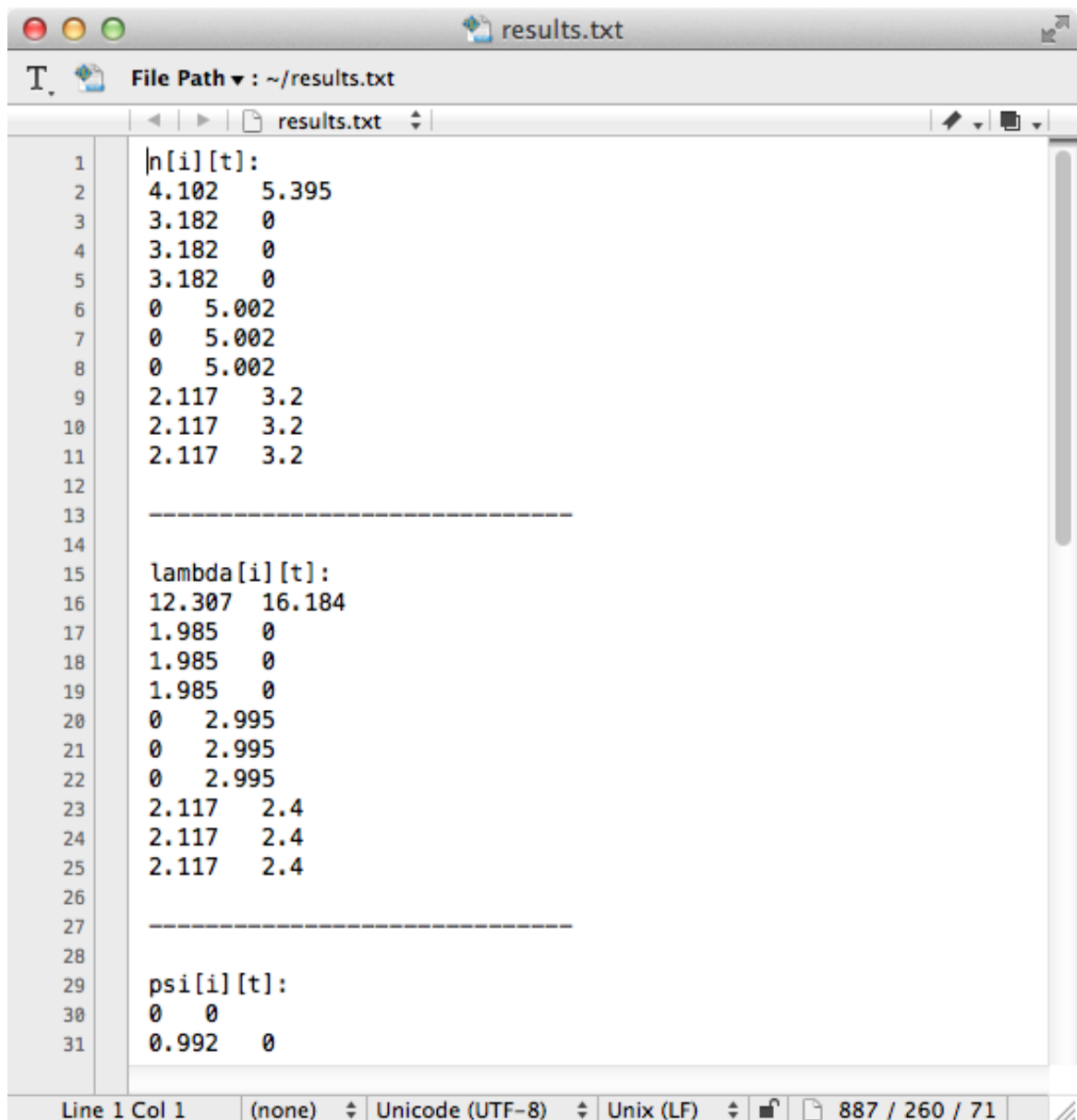


Рисунок 4.4

В окне результатов анализа пользователю предлагается на выбор два действия: закрыть окно или сохранить результаты в файл. Характеристики будут сохранены в указанный файл. На рисунке 4.5 изображен файл с результатами анализа для вышеприведенного примера (показаны не полностью):



```
1 | n[i][t]:
2 | 4.102  5.395
3 | 3.182  0
4 | 3.182  0
5 | 3.182  0
6 | 0  5.002
7 | 0  5.002
8 | 0  5.002
9 | 2.117  3.2
10 | 2.117  3.2
11 | 2.117  3.2
12 |
13 | -----
14 |
15 | lambda[i][t]:
16 | 12.307  16.184
17 | 1.985  0
18 | 1.985  0
19 | 1.985  0
20 | 0  2.995
21 | 0  2.995
22 | 0  2.995
23 | 2.117  2.4
24 | 2.117  2.4
25 | 2.117  2.4
26 |
27 | -----
28 |
29 | psi[i][t]:
30 | 0  0
31 | 0.992  0
```

Рисунок 4.5

4.3 Описание и назначение функций

Функция *AnalysisQueueingNetwork* является конструктором, инициализирующим переменные класса (входные данные). Функция имеет следующую сигнатуру:

```
public AnalysisQueueingNetwork(int L, int T, int[] N, int[] kappa,  
    int[][] s, double[][] mu);
```

Описание параметров функции представлено в таблице 4.2.

Таблица 4.2

Параметр	Тип	Содержание
L	int	Число СМО в сети
T	int	Число классов требований в СеМО
N	int[]	Вектор числа t -требований в сети
κ	int[]	Вектор числа обслуживающих приборов в СМО
s	int[][]	Матрица емкостей систем
μ	double[][]	Матрица интенсивностей обслуживания

Функция *analyze* является управляющей процедурой, с нее начинается анализ. Процедура для каждой системы C_i , $i = 1, \dots, L$, вызывает функции *swapServers* и *getStateDistribution*, а затем *estimateCharacteristics*. Функция имеет следующую сигнатуру:

```
public void analyze();
```

Функция *isInputDataValid* служит для проверки корректности входных данных, в частности, проверяются предположения, принятые в разделе 2.1. Функция имеет следующую сигнатуру:

```
public static boolean isInputDataValid(int L, int T, int[] N, int[]  
    kappa, int[][] s, double[][] mu);
```

Параметры функции были описаны выше. Функция возвращает true, если данные корректны, и false в противном случае.

Функция *swapServers* для анализа текущей системы переставляет ее с последней системой сети путем перестановки соответствующих индексов входных данных. Функция имеет следующую сигнатуру:

```
private void swapServers(int i);
```

Описание параметров функции представлено в таблице 4.3.

Таблица 4.3

Параметр	Тип	Содержание
i	int	Номер СМО C_i

Функция *getStateDistribution* вычисляет стационарное распределение вероятностей состояний для текущей системы. Функция имеет следующую сигнатуру:

```
private double[][] getStateDistribution();
```

Описание вычисляемых функцией данных представлено в таблице 4.4.

Таблица 4.4

Имя переменной	Тип	Содержание
pi	double[][]	Стационарное распределение вероятностей состояний для системы C_i

Функция *f* вычисляет множитель в выражении (2.4). Функция имеет следующую сигнатуру:

```
private double f(int i, int t, int n);
```

Описание параметров функции представлено в таблице 4.5.

Таблица 4.5

Параметр	Тип	Содержание
i	int	Номер СМО C_i
t	int	Номер класса требований
n	int	Число t -требований в системе i

Функция *estimateCharacteristics* вычисляет стационарные характеристики сети, а именно м. о. числа t -требований в СМО C_i , интенсивности потока t -требований в C_i и коэффициенты использования обслуживающих приборов

системы C_i t -требованиями, $i = 1, \dots, L$. Также дополнительно происходит вычисление вышеперечисленных характеристик для C_0 , которая представляет собой систему транспортировки в ГПС. Функция имеет следующую сигнатуру:

private void estimateCharacteristics();

Описание вычисляемых функцией данных представлено в таблице 4.6.

Таблица 4.6

Имя переменной	Тип	Содержание
<i>charN</i>	double[][]	М. о. числа t -требований в СМО C_i
<i>charLambda</i>	double[][]	Интенсивности потока t -требований в C_i
<i>charPsi</i>	double[][]	Коэффициенты использования обслуживающих приборов системы C_i t -требованиями

5 Аспекты практического применения

Было проведено несколько серий экспериментов с использованием разработанной программы анализа производственных систем с маршрутизацией, зависящей от состояния.

Эксперимент 1

Рассмотрим гибкую производственную систему с двумя рабочими станциями (C_1 и C_2), каждая из которых имеет два параллельных прибора [5]. Две рабочие станции связаны друг с другом циклическим конвейером C_0 , где каждая деталь имеет единичную среднюю задержку (время в пути), т.е. $\mu_0 = 1$. Пусть $N = 8$ (общее количество деталей в системе), а $s_1 = s_2 = 4$ (т.е. существует 2 лишних места в очереди каждой станции). Также предположим, что длительности обработки на обеих станциях имеют экспоненциальное распределение, причем средняя длительность обработки на обеих станциях равна 1,9.

Используя алгоритм анализа производственных систем с PSQ-маршрутизацией, получим результаты, приведенные в таблице 5.1.

Таблица 5.1

C_i	C_0	$C_{1,2}$
\bar{n}_i	2,014	2,993
λ_i	2,014	1.007
ψ_i	—	0,957

Эксперимент 2

Рассмотрим гибкую производственную систему с 14 машинами, сгруппированными по 7 рабочим станциям. Число приборов, интенсивность обработки детали одним прибором и емкость локального хранилища на каждой станции соответственно равны:

$$\kappa_1 = \kappa_2 = 1; \kappa_3 = \kappa_4 = \kappa_5 = 2; \kappa_6 = \kappa_7 = 3;$$

$$\mu_1 = \mu_2 = 3; \mu_3 = \mu_4 = \mu_5 = 1,5; \mu_6 = \mu_7 = 1;$$

$$s_1 = s_2 = 5; s_3 = s_4 = s_5 = 6; s_6 = s_7 = 7.$$

Число транспортеров $\kappa_0 = 7$, каждый из которых имеет интенсивность обра-

ботки $\mu_0 = 4$. В данной системе есть $N = 30$ палет, т.е. общее число деталей (одного типа) в любой момент времени равно 30.

В таблице 5.2 приведены основные характеристики гибкой производственной системы с PSQ–маршрутизацией.

Таблица 5.2

C_i	C_0	$C_{1,2}$	$C_{3,4,5}$	$C_{6,7}$
\bar{n}_i	5,511	2,549	3,491	4,460
λ_i	19,804	2,768	2,839	2,875
ψ_i	0,707	0,923	0,946	0,958

Эксперимент 3

Сравним PSQ–маршрутизацию с DSQ–маршрутизацией. Рассмотрим производственную систему с 18 машинами, которые сгруппированы по 9 рабочим станциям. Число приборов, интенсивность обработки детали одним прибором и емкость локального хранилища на каждой станции соответственно равны:

$$\kappa_1 = \kappa_2 = \kappa_3 = 1; \kappa_4 = \kappa_5 = \kappa_6 = 2; \kappa_7 = \kappa_8 = \kappa_9 = 3;$$

$$\mu_1 = \mu_2 = \mu_3 = 2; \mu_4 = \mu_5 = \mu_6 = 1,5; \mu_7 = \mu_8 = \mu_9 = 1;$$

$$s_1 = s_2 = s_3 = 4; s_4 = s_5 = s_6 = 6; s_7 = s_8 = s_9 = 7.$$

Число транспортеров $\kappa_0 = 9$, каждый из которых имеет интенсивность обработки $\mu_0 = 3$. В данной системе есть $N = 50$ палет, т.е. общее число деталей (одного типа) в любой момент времени равно 50. Мы сравним производительность производственной системы, имеющей PSQ–маршрутизацию, с производственной системой, имеющей DSQ–маршрутизацию, где вероятность перехода от станции C_0 к станциям C_1 , C_2 и C_3 равна $2/24$, а к другим станциям — $3/24$.

Математическое ожидание числа деталей (\bar{n}_i), пропускная способность (λ_i) и коэффициенты использования приборов (ψ_i) для каждой станции приведены в таблице 5.3 (в скобках указаны результаты для DSQ–маршрутизации). PSQ–маршрутизация имеет очевидные преимущества с точки зрения увеличения пропускной способности, а также коэффициентов использования приборов и очередей производственной системы [5].

Таблица 5.3

C_i	C_0	$C_{1,2,3}$	$C_{4,5,6}$	$C_{7,8,9}$
\bar{n}_i	11,070 (18,937)	3,003 (2,230)	4,492 (3,638)	5,482 (4,486)
λ_i	23,595 (21,168)	1,954 (1,687)	2,950 (2,672)	2,961 (2,697)
ψ_i	0,874 (0,784)	0,977 (0,844)	0,983 (0,891)	0,987 (0,899)

Эксперимент 4

Теперь изменим вышеприведенный пример следующим образом: в системе имеется два типа деталей и 50 палет разделяются на $N_1 = 20$ и $N_2 = 30$. Первый тип деталей посещает только станции C_1, C_2, C_3 и C_7, C_8, C_9 , а второй тип — только станции C_4, C_5, C_6 и C_7, C_8, C_9 . Станции C_7, C_8 и C_9 имеют по 7 приборов без дополнительного ожидания. На каждой из этих трех станций первый тип деталей может занимать до 3 приборов и интенсивность обработки деталей $\mu_{i1} = 1$ ($i = 7, 8, 9$), второй тип может занимать до 4 приборов и интенсивность обработки деталей $\mu_{i2} = 0,75$ ($i = 7, 8, 9$). Система транспортировки материалов представляет собой циклический конвейер, моделируемый как система с бесконечным числом приборов (то есть все требования обслуживаются сразу после поступления в систему), где среднее время пребывания каждого класса требований $\mu_0^{-1} = 1/3$ [5].

То есть мы имеем следующие данные:

$$L = 9; T = 2; \mathbf{N} = (20, 30);$$

$$I_1 = \{1, 2, 3, 7, 8, 9\}, I_2 = \{4, 5, 6, 7, 8, 9\};$$

$$\kappa_1 = \kappa_2 = \kappa_3 = 1; \kappa_4 = \kappa_5 = \kappa_6 = 2; \kappa_7 = \kappa_8 = \kappa_9 = 7;$$

$$\mu_{11} = \mu_{21} = \mu_{31} = 2; \mu_{42} = \mu_{52} = \mu_{62} = 1,5; \mu_{71} = \mu_{81} = \mu_{91} = 1;$$

$$\mu_{72} = \mu_{82} = \mu_{92} = 0,75;$$

$$s_{11} = s_{21} = s_{31} = 4; s_{42} = s_{52} = s_{62} = 6; s_{71} = s_{81} = s_{91} = 3;$$

$$s_{72} = s_{82} = s_{92} = 4.$$

$$\text{Для } C_0: \mu_{01} = \mu_{02} = 3; s_{01} = 20; s_{02} = 30.$$

В таблице 5.4 приводятся соответствующие результаты анализа.

Таблица 5.4

C_i	C_0		$C_{1,2,3}$	$C_{4,5,6}$	$C_{7,8,9}$	
t	Тип 1	Тип 2	Тип 1	Тип 2	Тип 1	Тип 2
\bar{n}_i	4,102	5,395	3,182	5,002	2,117	3,200
λ_i	12,307	16,184	1,985	2,995	2,117	2,400
ψ_i	—	—	0,993	0,998	0,706	0,800

Эксперимент 5

Возьмем гибкую производственную систему из примера 3 и посмотрим, как будут изменяться характеристики рабочих станций C_i , $i = 1, 2$, с изменением числа приборов, емкостей рабочих станций и интенсивностей обработки деталей (эти данные предполагаются одинаковыми для обеих рабочих станций). Результаты представлены на рисунках 5.1–5.3.

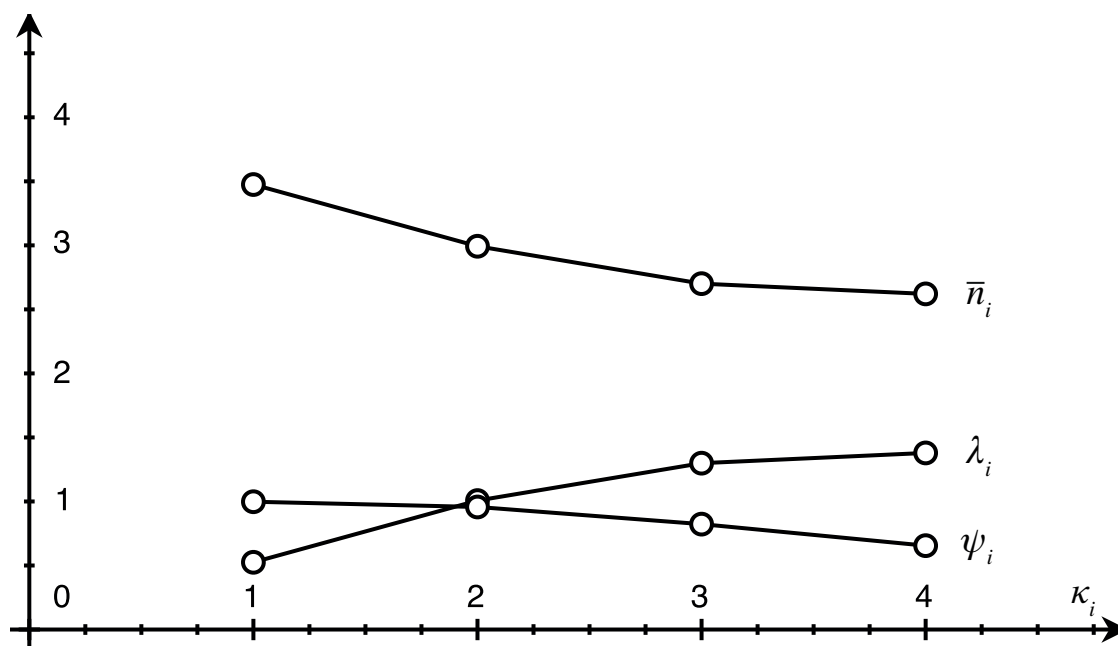


Рисунок 5.1 – Зависимость стационарных характеристик ГПС от числа приборов

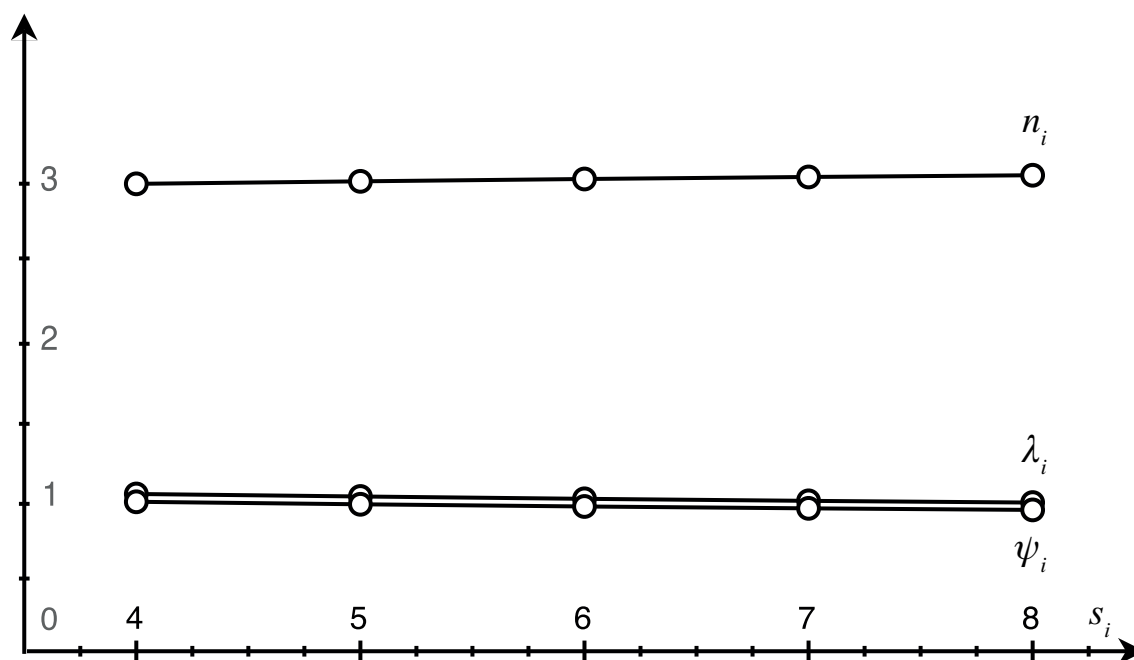


Рисунок 5.2 – Зависимость стационарных характеристик ГПС от емкостей рабочих станций

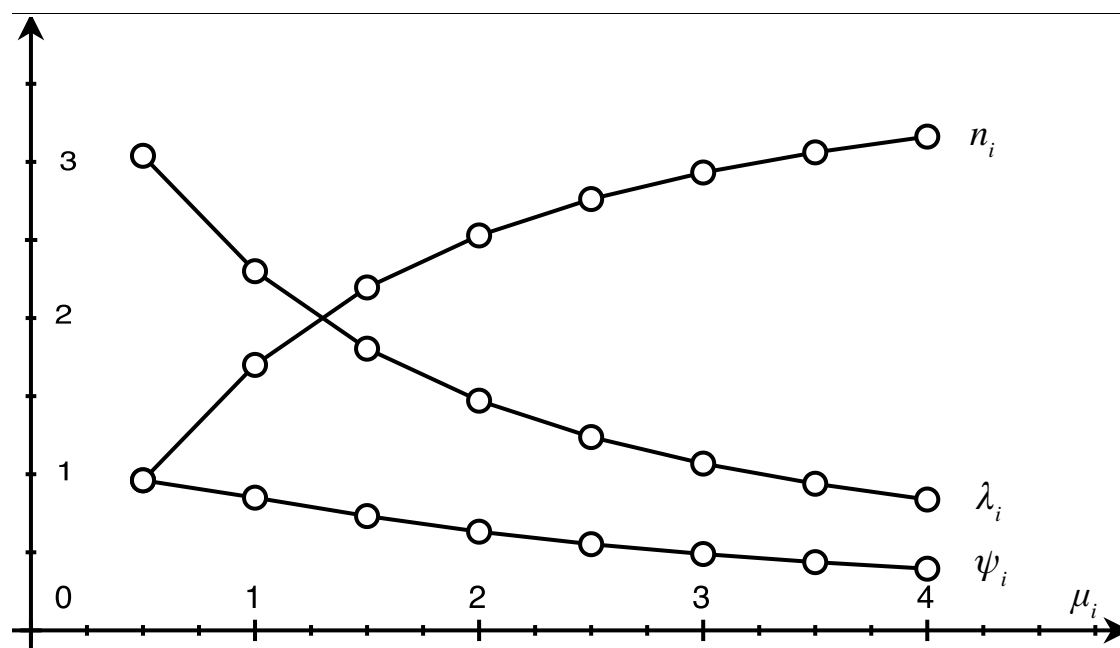


Рисунок 5.3 – Зависимость стационарных характеристик ГПС от интенсивностей обработки деталей

Помимо характеристик гибкой производственной системы, программа генерирует все распределения вероятностей. Время выполнения этих примеров на современном персональном компьютере — доли секунды.

ЗАКЛЮЧЕНИЕ

Основной целью выпускной квалификационной работы являлось исследование производственных систем с маршрутизацией, зависящей от состояния, разработка алгоритма метода анализа данных производственных систем, программная реализация алгоритма. Результатами работы являются следующие:

- рассмотрены производственные системы с маршрутизацией, зависящей от состояния;
- приведено доказательство того, что при PSQ-маршрутизации марковский процесс обратим относительно времени и имеет мультипликативную форму стационарного распределения;
- разработан алгоритм метода анализа производственных систем с маршрутизацией, зависящей от состояния;
- разработана программа, вычисляющая основные стационарные характеристики;
- проведены численные эксперименты с разработанной программой и приведены соответствующие результаты.

Эта модель может быть использована при решении задач анализа и оптимизации гибких производственных систем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Митрофанов Ю. И. Анализ сетей массового обслуживания. – Саратов: Научная книга, 2005. – 175 с.
- 2 Клейнрок Л. Теория массового обслуживания / Пер. с англ. – М.: Машиностроение, 1979. – 432 с.
- 3 Клейнрок Л. Вычислительные системы с очередями / Пер. с англ. – М.: Мир, 1979. – 600 с.
- 4 Гнеденко Б. В., Коваленко И. Н. Введение в теорию массового обслуживания. – М.: Наука, ГРФМЛ, 1966. – 432 с.
- 5 Yao D. D., Buzacott J. A. Modeling a class of state-dependent routing in flexible manufacturing systems // Annals of Operations Research. – 1985. – No. 3. – P. 153-167.
- 6 Yao D. D., Buzacott J. A. On queueing network as flexible manufacturing systems // Queueing Systems. –1986. – No 1. – P. 5-27.
- 7 Yao D. D., and Buzacott J. A. Modeling the performance of flexible manufacturing systems // International Journal of Production Research. –1984. – Vol. 23. – P. 945-955.
- 8 Reiser M., Lavenberg S. S. Mean-value analysis of closed multichain queueing networks // J. of the Association for Computing Machinery. – 1980. – Vol. 27, No. 2. – P. 313-322.
- 9 Kelly F. P. Reversibility and stochastic networks. – New York: Wiley, 1979.

ПРИЛОЖЕНИЕ А

Код программы

```
/* AnalysisFMS.java */

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.io.*;
import java.util.*;

public class AnalysisFMS extends JFrame {

    /**
     * Инициализация фрейма
     */
    public AnalysisFMS() {
        initComponents();
    }

    /**
     * Метод вызывается из конструктора для инициализации фрейма
     */
    private void initComponents() {
        jSpinnerL = new JSpinner();
        jSpinnerT = new JSpinner();
        jTextFieldKappa = new JTextField();
        jTextFieldN = new JTextField();
        JLabel jLabelL = new JLabel();
        JLabel jLabelT = new JLabel();
        JLabel jLabelKappa = new JLabel();
        JLabel jLabelS = new JLabel();
        JLabel jLabelMu = new JLabel();
        JLabel jLabelN = new JLabel();
        JButton jButtonGetResults = new JButton();
        JButton jButtonOpenInput = new JButton();
        JScrollPane jScrollPaneS = new JScrollPane();
        JTextAreaS = new JTextArea();
    }
}
```

```

JScrollPane jScrollPaneMu = new JScrollPane();
jTextAreaMu = new JTextArea();

setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
setTitle("Анализ гибких производственных систем");
setPreferredSize(new Dimension(600, 410));
setResizable(false);
getContentPane().setLayout(null);

jLabelL.setText("Число СМО (L):");
getContentPane().add(jLabelL);
jLabelL.setBounds(10, 16, 96, 16);

SpinnerNumberModel model = new SpinnerNumberModel(1, 1,
    999, 1);
jSpinnerL.setModel(model);
getContentPane().add(jSpinnerL);
jSpinnerL.setBounds(110, 10, 60, 28);

jLabelT.setText("Число классов требований (T):");
getContentPane().add(jLabelT);
jLabelT.setBounds(224, 16, 200, 16);

SpinnerNumberModel model2 = new SpinnerNumberModel(1, 1,
    999, 1);
jSpinnerT.setModel(model2);
getContentPane().add(jSpinnerT);
jSpinnerT.setBounds(425, 10, 60, 28);

jLabelN.setText("Вектор числа t-требований (N[t]):");
getContentPane().add(jLabelN);
jLabelN.setBounds(10, 55, 220, 16);

getContentPane().add(jTextFieldN);
jTextFieldN.setBounds(230, 50, 364, 28);

jLabelKappa.setText("Вектор числа приборов (каппа[i]):");
getContentPane().add(jLabelKappa);
jLabelKappa.setBounds(10, 95, 220, 16);

getContentPane().add(jTextFieldKappa);

```

```

jTextFieldKappa.setBounds(230, 90, 364, 28);

jLabelS.setText("Емкости систем (s[i][t]):");
getContentPane().add(jLabelS);
jLabelS.setBounds(10, 130, 218, 16);

jLabelMu.setText("Интенсивности обслуживания (mu[i][t]):");
getContentPane().add(jLabelMu);
jLabelMu.setBounds(310, 130, 300, 16);

jScrollPaneS.setViewportViewView(jTextAreaS);

getContentPane().add(jScrollPaneS);
jScrollPaneS.setBounds(10, 150, 280, 180);

jScrollPaneMu.setViewportViewView(jTextAreaMu);

getContentPane().add(jScrollPaneMu);
jScrollPaneMu.setBounds(310, 150, 280, 180);

jButtonOpenInput.setText("Открыть файл");
jButtonOpenInput.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        jButtonOpenInputActionPerformed(evt);
    }
});
getContentPane().add(jButtonOpenInput);
jButtonOpenInput.setBounds(270, 350, 130, 29);

jButtonGetResults.setText("Получить результаты");
jButtonGetResults.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        jButtonGetResultsActionPerformed(evt);
    }
});
getContentPane().add(jButtonGetResults);
jButtonGetResults.setBounds(410, 350, 180, 29);

pack();
}

```

```

/**
 * Формирование результата
 */
private void jButtonGetResultsActionPerformed(ActionEvent evt)
{
    try {
        int L = (Integer) jSpinnerL.getValue();
        int T = (Integer) jSpinnerT.getValue();

        int N[] = new int[T],
            kappa[] = new int[L + 1],
            s[][] = new int[L + 1][T];
        double mu[][] = new double[L + 1][T];

        Scanner sc = new Scanner(jTextFieldN.getText());
        for (int t = 0; t < T; ++t) {
            N[t] = sc.nextInt();
        }

        Scanner sc1 = new Scanner(jTextFieldKappa.getText());
        Scanner sc2 = new Scanner(jTextAreaS.getText());
        Scanner sc3 = new Scanner(jTextAreaMu.getText());
        for (int i = 0; i <= L; ++i) {
            kappa[i] = sc1.nextInt();
            for (int t = 0; t < T; ++t) {
                s[i][t] = sc2.nextInt();
                mu[i][t] = Double.valueOf(sc3.next());
            }
        }

        // Проверка входных данных
        if (!AnalysisQueueingNetwork.isInputDataValid(L, T, N,
            kappa, s, mu)) {
            throw new Exception();
        }

        // Создание объекта с результатами анализа
        AnalysisQueueingNetwork qNetwork = new
            AnalysisQueueingNetwork(L, T, N, kappa, s, mu);
        qNetwork.analyze();
    }
}

```

```

        // Создание объекта окна
        ResultFrame resultFrame = new ResultFrame();
        resultFrame.setVisible(true);

        // Передача объекта с результатами анализа окну для
        вывода
        resultFrame.showResults(qNetwork);
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null,
            "Некоторые данные указаны неверно или
            отсутствуют",
            "Ошибка", JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Считывание данных из файла
 */
private void jButtonOpenInputActionPerformed(ActionEvent evt) {
    JFileChooser chooser = new JFileChooser();
    FileNameExtensionFilter filter = new
        FileNameExtensionFilter("Text Files", "txt");
    chooser.setFileFilter(filter);

    int ret = chooser.showOpenDialog(null);
    if (ret == JFileChooser.APPROVE_OPTION) {
        File file = chooser.getSelectedFile();
        Scanner sc = null;
        try {
            sc = new Scanner(file);
            sc.useLocale(new Locale("US"));
            String string = "";

            int L = sc.nextInt();
            jSpinnerL.setValue(L);

            int T = sc.nextInt();
            jSpinnerT.setValue(T);

            int N[] = new int[T];
            for (int t = 0; t < T; ++t) {

```

```

        N[t] = sc.nextInt();
        string += String.valueOf(N[t]) + " ";
    }
    jTextFieldN.setText(string);

    int kappa[] = new int[L + 1];
    string = "";
    for (int i = 0; i <= L; ++i) {
        kappa[i] = sc.nextInt();
        string += String.valueOf(kappa[i]) + " ";
    }
    jTextFieldKappa.setText(string);

    int s[][] = new int[L + 1][T];
    string = "";
    for (int i = 0; i <= L; ++i) {
        for (int t = 0; t < T; ++t) {
            s[i][t] = sc.nextInt();
            string += String.valueOf(s[i][t]) + "\t";
        }
        string += "\n";
    }
    jTextAreaS.setText(string);

    double mu[][] = new double[L + 1][T];
    string = "";
    for (int i = 0; i <= L; ++i) {
        for (int t = 0; t < T; ++t) {
            mu[i][t] = sc.nextDouble();
            string += String.valueOf(mu[i][t]) + "\t";
        }
        string += "\n";
    }
    jTextAreaMu.setText(string);

    if (!AnalysisQueueingNetwork.isInputDataValid(L, T,
        N, kappa, s, mu)) {
        throw new Exception();
    }
} catch (Exception e) {
    JOptionPane.showMessageDialog(null,

```

```

        "Ошибка чтения из файла.\nНекорректные
        данные",
        "Ошибка", JOptionPane.ERROR_MESSAGE);
    } finally {
        if (sc != null) {
            sc.close();
        }
    }
}

/**
 * Создание и показ фрейма
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new AnalysisFMS().setVisible(true);
        }
    });
}

private JSpinner jSpinnerL;
private JSpinner jSpinnerT;
private JTextArea jTextAreaMu;
private JTextArea jTextAreaS;
private JTextField jTextFieldN;
private JTextField jTextFieldKappa;
}

```

```

/* AnalysisQueueingNetwork.java */

public class AnalysisQueueingNetwork {

    public int L, // число СМО
        T, // число классов требований
        NTotal, // общее число требований
        N[], // вектор числа t-требований
        kappa[], // вектор числа приборов
        s[][]; // вектор емкостей

    public double mu[][][], // интенсивности обслуживания требований
        PI[][][], // стационарное распределение
        charN[][][], // м. о. числа требований
        charLambda[][][], // интенсивности входящего потока
        charPsi[][][]; // коэффициенты использования приборов

    /**
     * Инициализация данных
     */
    public AnalysisQueueingNetwork(int L, int T, int[] N, int[]
        kappa, int[][] s, double[][] mu) {
        this.L = L;
        this.T = T;
        this.N = N;
        this.kappa = kappa;
        this.s = s;
        this.mu = mu;
        this.PI = new double[L + 1][T][L + 1];
        this.charN = new double[L + 1][T];
        this.charLambda = new double[L + 1][T];
        this.charPsi = new double[L + 1][T];
        this.NTotal = 0;
        for (int n = 0; n < N.length; ++n) {
            this.NTotal += N[n];
        }
    }

    /**
     * Перестановка системы i с последней
     */
    private void swapServers(int i) {

```



```

if (i != L) {
    int tmp1, tmp2[];
    double tmp3[];

    tmp1 = kappa[L];
    kappa[L] = kappa[i];
    kappa[i] = tmp1;

    tmp2 = s[L];
    s[L] = s[i];
    s[i] = tmp2;

    tmp3 = mu[L];
    mu[L] = mu[i];
    mu[i] = tmp3;
}
}

/**
 * Вспомогательная функция f
 */
private double f(int i, int t, int n) {
    int r;
    double f, nu;

    nu = Math.min(n, kappa[i]) / (double) n;
    if (i > 0) {
        r = s[i][t] - (n - 1);
    } else {
        r = n - N[t];
        for (int j = 1; j <= L; ++j) {
            r += s[j][t];
        }
    }
    f = r / (nu * n * mu[i][t]);
    return f;
}

/**
 * Вычисление стационарного распределения
 */

```

```

private double[][] getStateDistribution() {
    int NN[][] = new int[T][L + 1];
    double R[][] = new double[T][NTotal + 1],
        pi[][] = new double[T][NTotal + 1];
    int kk;
    double sum;

    for (int t = 0; t < T; ++t) {
        NN[t][0] = s[0][t];
        for (int n = 1; n <= NN[t][0]; ++n) {
            R[t][n] = Math.pow(f(0, t, n), -1);
        }

        for (int i = 1; i <= L; ++i) {
            NN[t][i] = Math.min(N[t], NN[t][i - 1] + s[i][t]);
            pi[t][0] = 1;

            for (int n = 1; n <= NN[t][i]; ++n) {
                sum = 0;
                kk = Math.min(s[i][t], n);

                for (int k = kk; k >= 1; --k) {
                    pi[t][k] = pi[t][k - 1] * f(i, t, k);
                    sum += pi[t][k];
                }

                if (n > NN[t][i - 1]) {
                    pi[t][0] = 0;
                } else {
                    pi[t][0] /= R[t][n];
                }

                R[t][n] = 1 / (sum + pi[t][0]);

                for (int k = 0; k <= kk; ++k) {
                    pi[t][k] *= R[t][n];
                }
            }
        }
    }

    return pi;
}

```

```

}

/**
 * Вычисление стационарных характеристик
 */
private void estimateCharacteristics() {
    double h, sum1, sum2;

    for (int i = 1; i <= L; ++i) {
        for (int t = 0; t < T; ++t) {
            // charN
            charN[i][t] = 0;
            for (int k = 0; k <= s[i][t]; ++k) {
                charN[i][t] += k * PI[i][t][k];
            }

            // charLambda
            sum1 = 0;
            sum2 = 0;
            if (kappa[i] < s[i][t]) {
                for (int k = 0; k <= kappa[i]; ++k) {
                    sum1 += k * PI[i][t][k];
                }
                for (int k = kappa[i] + 1; k <= s[i][t]; ++k) {
                    sum2 += PI[i][t][k];
                }
            } else {
                for (int k = 0; k <= s[i][t]; ++k) {
                    sum1 += k * PI[i][t][k];
                }
            }
            h = sum1 + kappa[i] * sum2;
            charLambda[i][t] = mu[i][t] * h;

            // charPsi
            if (mu[i][t] != 0) {
                int S = Math.min(kappa[i], s[i][t]);
                charPsi[i][t] = charLambda[i][t] / (S * mu[i][t]);
            } else {
                charPsi[i][t] = 0;
            }
        }
    }
}

```

```

        }
    }
}

// Для i = 0 (MHS)
for (int t = 0; t < T; ++t) {
    charN[0][t] = N[t];
    charLambda[0][t] = 0;
    for (int i = 1; i <= L; ++i) {
        charN[0][t] -= charN[i][t];
        charLambda[0][t] += charLambda[i][t];
    }
    charPsi[0][t] = charLambda[0][t] / (kappa[0] * mu[0][t]);
}

}

/**
 * Анализ СеМО (управляющая процедура)
 */
public void analyze() {
    for (int i = 1; i <= L; ++i) {
        swapServers(i);
        PI[i] = getStateDistribution();
        swapServers(i);
    }
    estimateCharacteristics();
}

/**
 * Проверка входных данных на корректность
 */
public static boolean isInputDataValid(int L, int T, int[] N,
int[] kappa, int[][] s, double[][] mu) {
    if (L <= 0) {
        return false;
    }

    if (T <= 0) {
        return false;
    }
}

```

```

for (int t = 0; t < T; ++t) {
    if (N[t] <= 0) {
        return false;
    }
}

for (int i = 0; i <= L; ++i) {
    if (kappa[i] <= 0) {
        return false;
    }
    for (int t = 0; t < T; ++t) {
        if (s[i][t] < 0) {
            return false;
        }
        if (mu[i][t] < 0) {
            return false;
        }
    }
}

int s_t[] = new int[T];
for (int t = 0; t < T; ++t) {
    for (int i = 1; i <= L; ++i) {
        s_t[t] += s[i][t];
    }
    if (N[t] > s_t[t]) {
        return false;
    }
}

int s_i[] = new int[L + 1];
for (int i = 1; i <= L; ++i) {
    for (int t = 0; t < T; ++t) {
        s_i[i] += s[i][t];
    }
    if (s_i[i] < kappa[i]) {
        return false;
    }
}

```

```
    for (int t = 0; t < T; ++t) {  
        if (s[0][t] != N[t]) {  
            return false;  
        }  
    }  
  
    return true;  
}  
}
```

```

/* ResultFrame.java */

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.text.DecimalFormat;
import java.util.Locale;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;

public class ResultFrame extends JFrame {

    public ResultFrame() {
        initComponents();
    }

    private void initComponents() {
        JScrollPane jScrollPanePI = new JScrollPane();
        JTextAreaPI = new JTextArea();
        JScrollPane jScrollPaneN = new JScrollPane();
        JTextAreaN = new JTextArea();
        JScrollPane jScrollPaneLambda = new JScrollPane();
        JTextAreaLambda = new JTextArea();
        JScrollPane jScrollPanePsi = new JScrollPane();
        JTextAreaPsi = new JTextArea();
        JLabel jLabelPI = new JLabel();
        JLabel jLabelPsi = new JLabel();
        JLabel jLabelN = new JLabel();
        JLabel jLabelLambda = new JLabel();
        JButton jButtonClose = new JButton();
        JButton jButtonSaveOutput = new JButton();

        setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);
        setTitle("Результаты анализа");
        setPreferredSize(new Dimension(600, 500));
        setResizable(false);
        getContentPane().setLayout(null);

        jLabelPI.setText("Стационарное распределение ( $\pi[i][t][k]$ ):");
    }

```

```

getContentPane().add(jLabelPI);
jLabelPI.setBounds(10, 10, 280, 16);

jScrollPanePI.setViewportViewView(jTextAreaPI);

getContentPane().add(jScrollPanePI);
jScrollPanePI.setBounds(10, 30, 280, 180);

jLabelN.setText("М.о. числа требований (n[i][t]):");
getContentPane().add(jLabelN);
jLabelN.setBounds(310, 10, 200, 16);

jScrollPaneN.setViewportViewView(jTextAreaN);

getContentPane().add(jScrollPaneN);
jScrollPaneN.setBounds(310, 30, 280, 180);

jLabelLambda.setText("Интенсивности вход. потока (lambda[i][t]):");
getContentPane().add(jLabelLambda);
jLabelLambda.setBounds(10, 220, 320, 16);

jScrollPaneLambda.setViewportViewView(jTextAreaLambda);

getContentPane().add(jScrollPaneLambda);
jScrollPaneLambda.setBounds(10, 240, 280, 180);

jLabelPsi.setText("Коэффициенты использования (psi[i][t]):");
getContentPane().add(jLabelPsi);
jLabelPsi.setBounds(310, 220, 260, 16);

jScrollPanePsi.setViewportViewView(jTextAreaPsi);

getContentPane().add(jScrollPanePsi);
jScrollPanePsi.setBounds(310, 240, 280, 180);

jButtonClose.setText("Закрыть");
jButtonClose.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        jButtonCloseActionPerformed();
    }
});

```



```

        }
    });
    getContentPane().add(jButtonClose);
    jButtonClose.setBounds(300, 440, 90, 29);

    jButtonSaveOutput.setText("Сохранить результаты");
    jButtonSaveOutput.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            jButtonSaveOutputActionPerformed(evt);
        }
    });
    getContentPane().add(jButtonSaveOutput);
    jButtonSaveOutput.setBounds(400, 440, 188, 29);

    pack();
}
/**
 * Заккрытие окна
 */
private void jButtonCloseActionPerformed() {
    this.setVisible(false);
}

/**
 * Запись результата анализа в файл
 */
private void jButtonSaveOutputActionPerformed(ActionEvent evt)
{
    JFileChooser chooser = new JFileChooser();
    FileNameExtensionFilter filter = new
        FileNameExtensionFilter("Text Files", "txt");
    chooser.setFileFilter(filter);

    int ret = chooser.showSaveDialog(null);
    if (ret == JFileChooser.APPROVE_OPTION) {
        File file = chooser.getSelectedFile();
        try {
            FileWriter fw = new FileWriter(file + ".txt");
            fw.write("n[i][t]:\n");
            jTextAreaN.write(fw);
        }
    }
}

```

```

        fw.write("\n-----\n");

        fw.write("\nlambda[i][t]:\n");
        jTextAreaLambda.write(fw);

        fw.write("\n-----\n");

        fw.write("\npsi[i][t]:\n");
        jTextAreaPsi.write(fw);

        fw.write("\n-----\n");

        fw.write("\nPI[i][t][k]:\n");
        jTextAreaPI.write(fw);

        fw.close();
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null,
            "Ошибка сохранения файла",
            "Ошибка", JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Вывод результата анализа
 */
public void showResults(AnalysisQueueingNetwork qNetwork) {
    Locale.setDefault(new Locale("US"));
    DecimalFormat df = new DecimalFormat("#.###");

    String result = "";
    for (int i = 1; i <= qNetwork.L; ++i) {
        result += "i = " + i + ":\n";
        for (int t = 0; t < qNetwork.T; ++t) {
            for (int k = 0; k <= qNetwork.s[i][t]; ++k) {
                result += df.format(qNetwork.PI[i][t][k]) + "\t";
            }
            result += "\n";
        }
    }
}

```

```

    }
    jTextAreaPI.setText(result);

    result = "";
    for (int i = 0; i <= qNetwork.L; ++i) {
        for (int t = 0; t < qNetwork.T; ++t) {
            result += df.format(qNetwork.charN[i][t]) + "\t";
        }
        result += "\n";
    }
    jTextAreaN.setText(result);

    result = "";
    for (int i = 0; i <= qNetwork.L; ++i) {
        for (int t = 0; t < qNetwork.T; ++t) {
            result += df.format(qNetwork.charLambda[i][t]) + "\t";
        }
        result += "\n";
    }
    jTextAreaLambda.setText(result);

    result = "";
    for (int i = 0; i <= qNetwork.L; ++i) {
        for (int t = 0; t < qNetwork.T; ++t) {
            result += df.format(qNetwork.charPsi[i][t]) + "\t";
        }
        result += "\n";
    }
    jTextAreaPsi.setText(result);
}

public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new ResultFrame().setVisible(true);
        }
    });
}

private JTextArea jTextAreaLambda;

```

```
private JTextArea jTextAreaN;  
private JTextArea jTextAreaPI;  
private JTextArea jTextAreaPsi;  
}
```