МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) /специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Программная реализация алгоритмов синтеза расписаний обслуживания

Обучающийся С.А. Фофанов _____
(Инициалы Фамилия) (личная подпись)

Руководитель канд. тех. наук, доцент, Н.А. Сосина
(Ученая степень (при наличии), ученое звание (при наличии), Инициалы, Фамилия)

Консультант канд. филол. наук, доцент, М. В. Дайнеко
(Ученая степень (при наличии), ученое звание (при наличии), Инициалы, Фамилия)

Тольятти, 2025

Аннотация

Название выпускной квалификационной работы «Программная реализация алгоритмов синтеза расписаний обслуживания».

ВКР состоит из введения, трёх разделов, 11 рисунков, 3 таблиц, заключения и списка литературы из 21 источника, включая иностранные.

Целью данной ВКР является проектирование и программная реализация алгоритма составления расписания обслуживания.

Объектом ВКР являются алгоритмы синтеза расписания.

ВКР можно разделить на несколько логически связанных частей: постановки задачи, разработка алгоритма ее решения, программная реализация и тестирование.

В первой части подробно описывается математическая модель задачи составления расписания обслуживания множества заявок и приводится обоснование выбора метода динамического программирования для решения поставленной задачи.

Вторая часть состоит из примера решения поставленной задачи и описания общего алгоритма решения.

В третьем разделе представлены результаты работы программы и проведен анализ эффективности.

В заключение хотелось бы подчеркнуть, что в результате была разработана не только программа для планирования, но и общий алгоритм поиска оптимального решения задачи синтеза расписаний обслуживания.

Работа представляет интерес для широкого круга читателей, интересующихся в решении задач оптимизации бизнес-процессов методом динамического программирования.

Abstract

The title of the graduation work is *Software implementation of the algorithms for synthesizing the application processing schedules*.

The graduation work consists of an introduction, 3 parts, 11 figures, 3 tables, a conclusion and a list of 21 references including foreign sources.

The aim of this research is to design and provide software implementation of the scheduling algorithm.

The object of the graduation work is the scheduling synthesis algorithms.

The graduation work may be divided into several logically connected parts which are: problem statement, algorithm development, software implementation and testing.

The first part describes a mathematical model designed for scheduling the processing of the multiple applications. This part also explains the choice of a dynamic programming method to solve the problem.

The second part gives an example of solving the problem and describes the general solution algorithm.

The third part presents the results of the programme operation and conducts an analysis of its effectiveness.

In conclusion, it should be highlighted that the result of the graduation work is not only a programme designed for scheduling, but also the general algorithm to search for an optimal solution to the problem of synthesizing the schedules of processing the multiple applications.

The work is of interest for a wide circle of readers interested in solving business process optimization problems using the dynamic programming method.

Содержание

Введение

В современных условиях конкурентного рынка, где время - это деньги, а качество обслуживания напрямую влияет на уровень удовлетворенности клиентов, оптимизация процессов становится ключевым фактором успешного функционирования организаций.

Одной из наиболее актуальных задач управления является синтез расписаний обслуживания множества заявок, что представляет собой сложную задачу, требующую глубокого анализа существующих алгоритмов. Расписания могут касаться обработки заказов, распределения задач среди сотрудников или планирования использования оборудования. Применение эффективных алгоритмов синтеза расписаний позволяет значительно повышать производительность, снижать затраты и улучшать качество услуг.

Актуальность темы синтеза расписаний возрастает в условиях ограниченности ресурсов и необходимости быстрого реагирования на изменяющиеся требования со стороны клиентов. Например, в здравоохранении важно минимизировать время ожидания для пациентов, обеспечив оптимальное распределение врачей по расписанию; в производстве – добиться надлежащей загрузки станков минимизируя простой и максимизируя объемы выпускаемой продукции. Перечисленные аспекты подчеркивают важность исследования и разработки алгоритмов, способных эффективно решать задачу синтеза расписаний.

В ходе выполнения выпускной квалификационной работы предполагается выполнение следующих задач:

- обоснование актуальности работы;
- постановка задачи синтеза расписания обслуживания множества заявок;
- описание предметной области и анализ существующих методов решения задач составления расписания;

– построение алгоритма для решения задачи синтеза расписаний обслуживания множества заявок;
– подробное описание аналитического решения примера рассматриваемой задачи;
– выбор технических средств;
– программная реализация алгоритма, тестовой структуры данных и пользовательского интерфейса программного продукта;
– тестирование программного продукта.

Таким образом объектом исследования данной работы является математическая модель задачи составления расписаний обслуживания и алгоритмы решения этой задачи.

Цель работы: разработка и программная реализация алгоритма составления расписания обслуживания с использованием метода динамического программирования.

1 Постановка задачи и методы решения

1.1 Общая характеристика задачи синтеза расписания

В начале 20 века, с ростом промышленности, появилась необходимость в планировании работы машин и рабочих. В это время разрабатывались простые алгоритмы и правила для организации рабочего процесса, но эти методы были скорее эмпирическими, чем основанными на строгих математических моделях.

Появление линейного программирования в 1940-х годах, а затем и более сложных формул, таких как целочисленное программирование и динамическое программирование, позволило создать первые формализации задач планирования и синтеза расписаний.

Исследователи, такие как Джордж Данциг, который разработал симплекс метод, и Ритверд, который работал над задачами целочисленного программирования, внесли значительный вклад в создание моделей для оптимизации расписаний.

В 1970-х – 1980-х началась работа по созданию компьютерных систем для автоматизации задач планирования, что способствовало более широкому применению теории расписаний в промышленных и коммерческих процессах.

На сегодняшний день для многих компаний основным источником доходов является выполнение каких-либо услуг, связанных, например, с промышленностью, образованием или массовым обслуживанием. И компании, которые уже давно работают в определенной области всегда имеют множество клиентов, из-за чего возникает проблема, когда спрос значительно превышает возможности для выполнения этих услуг. Это означает, что организации должны ориентируясь на свои приоритеты, учитывать эффективность управления ресурсами, возможности снижения

затрат, снижения рисков или улучшения производительности, выбрать в какой последовательности необходимо выполнить имеющуюся работу – составить правильное расписание обслуживания.

Часто это расписание оказывается не эффективным, потому что иногда эта проблема может игнорироваться и, например, заявки будут обслуживается в порядке очереди или от самых длительных к более коротким. Правильным решением будет одновременно учитывать и приоритет этих заявок, и время их выполнения, и то, как они будут распределены для выполнения между работниками. И в таком случае возникает новая проблема: заказов может быть много, соответственно на составление порядка обслуживания может потребоваться значительное количество времени, хотя работу нужно начинать как можно раньше.

Благодаря правильному решению задачи построения расписаний можно добиться следующих положительных эффектов:

- сокращение времени ожидания клиентов и повышение качества услуг, что непосредственно влияет на уровень удовлетворенности клиентов;
- избежание простоев и неэффективного использования времени, что особенно важно в конкурентных отраслях;
- снижение затрат, связанных с выполнением заявок, что в итоге позволяет к увеличить прибыль компании.

Внедрение алгоритмов для автоматического синтеза расписаний могло бы освободить время руководителей компании, позволяя им сосредоточиться на более важных задачах, таких как стратегическое планирование и развитие бизнеса.

Таким образом для оптимизации работы предприятия имеет смысл составление алгоритма и реализация программы, для решения задачи построения эффективного расписания обслуживания множества заявок.

1.2 Постановка задачи

Похожие задачи, такие как оптимизация обслуживания очередей рассматриваются в моделях СМО.

Система массового обслуживания (СМО) – это математическая модель, разработанная в 1950-60-е годы, используемая для анализа процессов, в которых входящий поток заявок (клиентов, задач) обрабатывается ограниченными ресурсами (обслуживающими механизмами).

Для описания задачи синтеза расписания обслуживания множества заявок будет удобно использовать основные компоненты СМО, такие как:

- множество заявок, обладающих характеристиками для оценки трудоемкости и важности работы;
- множество обслуживающих механизмов – сотрудников компании;
- очередь, определяющая множество заявок и последовательность их обслуживания.

В задаче составления расписания обслуживания процессом решения является построение оптимальной для конкретных требований очереди.

После рассмотрения примеров задач составления расписаний и наблюдений процесса реального распределения и упорядочивания задач на практике была сформулирована задача синтеза расписания обслуживания множества заявок.

На данный момент в системе существует множество заявок {1, 2, ..., n}. Каждую заявку должен выполнить один из m специалистов. Для процесса обслуживания вводятся следующие ограничения:

- одновременно специалист может выполнять не более одной заявки;
- выполнение каждой заявки должно происходить без прерываний в рамках рабочего времени;
- нельзя не выполнив полностью одну заявку начать выполнение другой;

– переход к выполнению следующей заявки затрат времени не требует.

Обслуживание начинается в момент времени $t_0 = 0$.

Каждая заявка i обладает двумя характеристиками: временной оценкой выполнения заявки $T_з(i)$ и штрафом p(i) в единицу времени пребывания в системе. Характеристики всегда должны быть больше нуля:

$$T_з(i)>0; \text{ при } i=1,...,n, \quad (1)$$

$$p_з(i)\geqslant 0; \text{ при } i=1,...,n. \quad (2)$$

Всех специалистов считаем идентичными: каждый может выполнять любую из заявок и каждому для выполнения одной и той же заявки требуется одинаковое количество времени $T_з(i)$.

Расписанием обслуживания считается множество из m элементов $R = \{r_1, r_2, ..., r_m\}$, где каждый из элементов является перестановкой $r_j = \{i_1, i_2, ..., i_k\}$, задающей номера заявок, переданных j специалисту, и порядок, в котором эти заявки должны быть выполнены при реализации расписания R. Одна и та же заявка не может быть в очереди на выполнение одновременно в нескольких расписаниях.

Пусть $t_н(R, i_k)$ – время начала обслуживания k заявки, а $t_к(R, i_k)$ – время окончания обслуживания k заявки. Эти параметры зависят от очередности обслуживания, то есть от конкретного расписания R. Тогда по описанным выше правилам получаются соотношения (3) и (4):

$$t_0=t_н(R, i_1)=0, \quad (3)$$

$$t_k(R,i_k)=t_н+T_з(i_k); \text{ при } k=1,\ldots,n. \quad (4)$$

Тогда величина штрафа для произвольной заявки $S_i(R)$ при использовании расписания R будет равна произведению штрафа в единицу

времени и момента завершения выполнения заявки, как показано в формуле (5):

$$S_i(R)=p_з(i)\cdot t_к(R,i). \qquad (5)$$

Сумма штрафа по всем заявкам при реализации расписания R вычисляется по формуле (6):

$$S(R)=\sum_{i=1}^{n} S_i(R)=\sum_{i=1}^{n} p_з(i)\cdot t_к(R,i). \qquad (6)$$

Решением задачи считается такое расписание, при котором суммарный штраф по всем заявкам будет минимальным. Целевую функцию можно записать в виде формулы (7):

$$S(R)\rightarrow \min. \qquad (7)$$

В процессе решения необходимо решить две основные проблемы: определить в какой последовательности лучше выполнять заявки и как лучше их распределить между специалистами для выполнения.

1.3 Выбор метода решения задачи составления расписания

Самым очевидным способом решения задачи составления расписания является метод перебора. Можно просто рассмотреть все возможные очередности обслуживания и комбинации их распределения между сотрудниками и выбрать те, при которых штраф будет минимальным. Но используя такое решение, получится, что даже с небольшим количеством заявок и работников, между которыми их нужно распределить, и с использованием современного компьютерного обеспечения алгоритм будет

выполняться достаточно длительное время, ведь сложность алгоритма в таком случае соответствует количеству комбинаций всех возможных перестановок и всех возможных распределений заявок между исполнителями, ведь сложность такого алгоритма растет по функции факториала, как при увеличении числа заявок, так и при увеличении числа сотрудников. А факториальная сложность – это самая высокая степень роста времени выполнения алгоритма.

Одним из более эффективных подходов к решению задач синтеза расписаний является метод динамического программирования.

«Динамическое программирование (ДП) – это вычислительный метод для решения задач определенной структуры. В общей постановке задача динамического программирования формулируется следующим образом. Имеется некоторая управляемая физическая система, характеризующаяся определенным набором параметров. Требуется построить оптимальное управление (на множестве допустимых управлений), переводящее систему из начального состояния в конечное состояние, обеспечив целевой функции (показателю эффективности управления) экстремум.»[10]

При решении задачи методом динамического программирования основная задача разбивается на множество взаимосвязанных подзадач. Решение выражается через оптимальные решения подзадач. При решении подзадач предполагается сохранение их результатов, так как большинство из этих решений может использоваться несколько раз.

«В основе вычислительных алгоритмов динамического программирования лежит следующий принцип оптимальности, сформулированный Р. Беллманом: каково бы не было состояние системы S в результате k-1 шагов, управление на k-ом шаге должно выбираться так, чтобы оно в совокупности с управлениями на всех последующих шагах с (k+1)-го до N-го включительно доставляло экстремум целевой функции.»[10]

При решении задач динамического программирования формулируется рекуррентное соотношение (или уравнение) Беллмана. Это уравнение можно записать в математической форме следующим образом:

$$F_k(x_{k-1},u_k)=\underset{uk}{extr}\left(z_k(x_{k-1},u_k)+F_{k+1}(x_k)\right), \qquad (8)$$

где:

x_{k-1} – возможные состояния системы до текущего шага;

x_k – возможные состояния системы после текущего шага;

u_k – возможные управления на текущем шаге;

$z_k(x_{k-1}, u_k)$ – условно-оптимальное решение на предыдущем шаге;

$F_k(x_{k-1}, u_k)$ – условно-оптимальное решение на текущем шаге;

$F_{k+1}(x_k)$ – условно-оптимальное решение на следующем шаге. На последнем шаге можно предположить, что $F_{k+1}(x_k) = 0$.

Методы перебора обычно имеют экспоненциальное время выполнения, а динамическое программирование обычно выполняется за полиномиальное время, что делает динамическое программирование более эффективным. Метод динамического программирования также обладает и недостатками: из-за того, что он сохраняет решения подзадач, может потребляться больше памяти. Но при таких объемах, когда алгоритм будет потреблять слишком много памяти, метод перебора будет практически невыполним, так как потребует очень больших временных затрат.

На практике в компаниях часто требуется распределение десятков или сотен задач между десятками специалистов, что уже составляет высокую алгоритмическую сложность. В таких ситуациях решение методом перебора может легко стать невыполнимым из-за его высокой продолжительности, в то время как метод динамического программирования сможет решить задачу

с большими объемами входных данных, хоть и с большими затратами памяти компьютера.

При решении задач методом динамического программирования обычно строятся таблицы, графы состояний или деревья состояний.

Граф – это совокупность двух множеств: вершин и рёбер.

Путь – это последовательность рёбер и вершин, соединяющая две вершины.

Цепь – это путь, в котором вершины и рёбра не повторяются.

Цикл – это цепь, в которой первая и последняя вершины совпадают.

Дерево – это граф, в котором между любыми двумя вершинами существует единственный путь. При этом в дереве отсутствуют циклы. Обычно узлы дерева соотносятся друг с другом как родитель и потомок. Узел, у которой нет родительских узлов – это вершина дерева.

Длина пути – это количество рёбер, составляющих путь.

Высота дерева – это наибольшая длина пути от вершины дерева до узла не имеющего потомков.

Глубина узла - это длина пути от этого узла до вершины дерева.

Таким образом в работе будет рассмотрено решение задачи методом динамического программирования. Так как динамическое программирование – это эффективный метод для решения задач синтеза расписаний обслуживания, обеспечивающий оптимальные решения через стратегию разбиения на подзадачи и сохранения промежуточных решений. Он подходит для задач, где важны временные ограничения, приоритеты и ресурсы, и может быть адаптирован для различных сценариев с различными требованиями и ограничениями.

2 Разработка общего алгоритма решения

2.1 Пример решения задачи для одного специалиста

Одной из основных частей решения приведенной задачи является упорядочивание заявок. Поэтому сначала лучше рассмотреть частный случай задачи, когда количество специалистов, обрабатывающих заявки равно единице, тогда расписание имеет вид R = {r} = r, то есть состоит из одной последовательности – расписания обслуживания для одного работника. При составлении такого расписания распределять задачи между работниками не нужно, необходимо только правильно упорядочить их.

Допустим r_0 = {0, 1, ..., n} – оптимальное расписание для решаемой задачи, а r_1 - расписание, полученное из расписания r_0 перестановкой двух рядом стоящих заявок k и k + 1. Тогда для всех заявок, кроме тех, которые поменялись местами, время окончания выполнения для обоих расписаний будет одинаковым. Поэтому при вычислении суммарных штрафов у двух расписаний r_0 и r_1 будут отличаться только k и k + 1 слагаемые. Тогда получается разность (9):

$$\sum_{i=1}^{n} S(r_0) - \sum_{i=1}^{n} S_i(r_1) = p(k+1)\cdot T_з(k) - p(k)\cdot T_з(k+1). \qquad (9)$$

Эта разность не положительна, так как r_0 – оптимальное расписание, соответственно штраф при его реализации не может быть больше чем при реализации r_1. Получаем неравенство (10):

$$p(k+1)\cdot T_з(k) - p(k)\cdot T_з(k+1) \leqslant 0. \qquad (10)$$

Перенеся одно из слагаемых в правую часть, и разделив обе части на произведение $t_к(k)t_к(k+1)$, получается неравенство (11):

$$\frac{p(k+1)}{T_з(k+1)} \geqslant \frac{p(k)}{T_з(k)}. \qquad (11)$$

Исходя из неравенства (11) легко составляется простейший алгоритм отыскания оптимального расписания: необходимо для каждой заявки по формуле (12) определить показатель m(i):

$$m(i)=\frac{p(i)}{T_з(i)},\ i=1,2,...,n. \quad (12)$$

После все заявки упорядочиваются по убыванию значения показателя (12). Таким образом будет найдено оптимальное расписание для обслуживания данного множества заявок одним специалистом.

«Изложенный способ отыскания правила оптимального упорядочения заявок называется перестановочным приемом. Содержательный смысл его очевиден: чем меньше продолжительность обслуживания заявки и чем больше по этой заявке штраф за единицу времени пребывания в системе, тем раньше она должна обслуживаться.»[4]

2.2 Пример решения задачи для m специалистов с использованием метода динамического программирования

Так как в приведенном выше решении задачи с одним специалистом представлен алгоритм получения оптимального порядка выполнения заявок, то для решения задачи с количеством специалистов равным m, где m больше единицы, нужно только оптимально распределить эти заявки между ними, сохранив тот же порядок.

Очевидно, что задача в которой специалистов больше чем заявок не имеет смысла, поэтому рассматриваются решения только для задач, в которых число заявок n больше числа специалистов m.

Можно предположить, что оптимальное решение получится, если распределять заявки следующим образом: первые m заявок передаются в работу по одной каждому специалисту, каждая следующая заявка из {m + 1, m + 2, ..., n} передается тому специалисту, который раньше обслужит все уже переданные ему заявки.

Описанное выше решение для нескольких специалистов не всегда оптимально. Например, необходимо распределить три заявки между двумя специалистами. Заявки обладают следующими характеристиками: $p(1) = 4$, $T_з(1) = 1$, $p(2) = 6$, $T_з(2) = 3$, $p(3) = 10$, $T_з(3) = 10$. Тогда $m(1) = 4$, $m(2) = 2$, $m(3) = 1$. Заявки уже упорядочены в соответствии с перестановочным приемом. Тогда по описанному выше способу распределению получим: $R = \{r_1, r_2\}$, $r_1 = \{1, 3\}$, $r_2 = \{2\}$. Суммарный штраф при реализации такого расписания:

$$\sum_{i=1}^{n} S_i (R)=4\cdot1+6\cdot3+10\cdot11=132. \qquad (13)$$

Но с такими же исходными данными, при реализации расписания $R = \{r_1, r_2\}$, $r_1 = \{1, 2\}$, $r_2 = \{3\}$ суммарный штраф уменьшится:

$$\sum_{i=1}^{n} S_i (R)=4\cdot1+6\cdot4+10\cdot10=128. \qquad (14)$$

Видно, что решение, полученное таким распределением, оптимальным вообще говоря, не является.

Для точного решения задачи возможно использование метод динамического программирования. Так как для одного специалиста оптимальным расписанием является последовательность заявок, упорядоченная по убыванию параметра m(i), то и при решении задачи для нескольких специалистов эти заявки следует распределять в таком же порядке. В расписаниях отдельных специалистов заявки будут также

следовать в порядке убывания параметра m(i). При решении будет удобно ввести новую нумерацию, соответствующую порядку распределения заявки.

При составлении оптимального расписания необходимо для каждой следующей заявки определить какому из специалистов её направить в очередь на выполнение. При чем на каждом шаге все предыдущие заявки уже распределены, известны значения T_1, T_2, ..., T_m времени, которое будет затрачено каждым специалистом на выполнение всех направленных ему ранее заявок.

Пусть $S^*(i, T_1, T_2, ..., T_m)$ – минимально возможная величина суммарного штрафа по заявкам множества {1, 2, ..., i}. Здесь также можно рассмотреть и ситуацию, когда специалист был занят до начала распределения заявок, тогда на нулевом шаге $S^*(0, 0, 0, ..., T_j, ..., 0) = 0$ ему следует определить начальное время T_j отличное от нуля, и учитывать его при распределении заявок, далее называем такое состояние нулевым. В ситуациях, когда на момент начала составления расписания все специалисты свободны, то есть могут немедленно приступить к выполнению новых задач, нулевое состояние можно не учитывать, а первую заявку определить любому из специалистов, например первому, получим $S^*(1, T_з(1), 0, ..., 0) = p(1)T_з(1)$. Для каждой последующей заявки необходимо рассмотреть её распределение каждому из специалистов. При передаче i заявки j специалисту штраф определяется как сумма штрафа в предшествующем этому состоянии и штрафа от новой заявки:

$$S^*\left(i,T_1,\ldots,T_j+T_з(i),\ldots,T_m\right)=S^*\left(i,T_1,\ldots,T_j,\ldots,T_m\right)+$$
$$+\,p(i)\cdot\left(T_j+T_з(i)\right).(15)$$

В общем решение можно описать следующим рекуррентным соотношением (уравнением Беллмана):

$$S^*\left(i,T_1,\ldots,T_j+T_з(i),\ldots,T_m\right)=$$

$$=\min\left\{S^*(i-1,T_1,T_2,\ldots,T_m)+p(i)\cdot\left(T_j+T_з(i)\right)\right\}, \qquad (16)$$

где j = 1, 2, ... m соответствует номеру специалиста; i = 1, 2, ..., n соответствует номеру заявки.

Процесс решения удобно представить в виде дерева состояний. Каждому узлу дерева соответствует номер i рассматриваемой заявки, номер j специалиста, которому определена рассматриваемая заявка, время T_1, T_2, ..., T_m, когда освободится каждый из специалистов и накопленный при данном состоянии суммарный штраф. Вершиной дерева будет нулевое состояние, при котором определяются начальные занятости сотрудников до распределения новых заявок.

Все состояния, рассматривающие момент распределения некоторой i заявки находятся на одной глубине. Соответственно каждый узел имеет не более m потомков – вариантов распределения заявки. По листам построенного дерева определяются общие штрафы при реализации конкретных расписаний, лист, у которого значение штрафа минимально, соответствует оптимальному решению.

Оптимальное расписание получается обратным ходом от листа к вершине дерева, при этом заявки добавляются в конец очереди для соответствующих им по узлу-состоянию специалистам, так как на более высоких уровнях дерева рассматривались те заявки, которые будет оптимально выполнить раньше.

Пример решения задачи:

Дано множество заявок {1, 2, 3, 4, 5}, характеристики которых представлены в таблице 1. Необходимо распределить эти заявки между тремя специалистами. На момент времени $t_0 = 0$ все специалисты свободны и могут немедленно приступить к выполнению переданных им задач.

Таблица 1 – Характеристики заявок

Документ	Штраф, p(i)	Время выполнения, $T_з(i)$
Заявка 1	4	5
Заявка 2	3	3
Заявка 3	7	8
Заявка 4	7	2

Первый шаг решения – это вычисление параметра m(i) и упорядочивание заявок по убыванию этого параметра. Тогда после расчета параметра m(i) и сортировки списка заявок по возрастанию этого параметра, они будут рассматриваться в таком же порядке, как показано в таблице 2.

Таблица 2 – Заявки, упорядоченные по убыванию параметра m(i)

Документ	Новый номер, i	Штраф, p(i)	Время выполнения, $T_з(i)$	Параметр m(i)
Заявка 4	1	7	2	3,5
Заявка 2	2	3	3	1
Заявка 3	3	7	8	0,875
Заявка 1	4	4	5	0,8

Далее строится дерево состояний, в котором рассматриваются все возможные варианты распределения заявок между сотрудниками. В данном случае нулевое состояние не учитываем, так как все сотрудники начинают одновременно. Заявку 4 сразу определим первому сотруднику. Тогда первое состояние будет содержать следующий набор параметров: T1 = 2, T2 = 0, T3 = 0, S*(1, 2, 0, 0) = 14. В данном случае если узел является левым потомком для другого узла, то это означает что текущая заявка определена первому специалисту, соответственно если узел является центральным потомком - второму специалисту, и если правым – третьему. Остальные состояния

заполняются сверху вниз в соответствии с уравнением 16. Итоговое дерево состояний представлено на рисунке 1.

По дереву видно, что минимальный штраф достигается в двух итоговых состояниях S*(4, 5, 8, 5) = S*(4, 5, 5, 8) = 105. Проходя по родительским узлам этих состояний получаются следующие расписания R_1 = {{1, 2}, {3}, {4}} и R_2 = {{1, 2}, {4}, {3}}, или переходя к изначальной нумерации R_1 = {{Заявка 4, Заявка 2}, {Заявка 3}, {Заявка 1}} и R_2 = {{Заявка 4, Заявка 2}, {Заявка 1}, {Заявка 3}} соответственно.

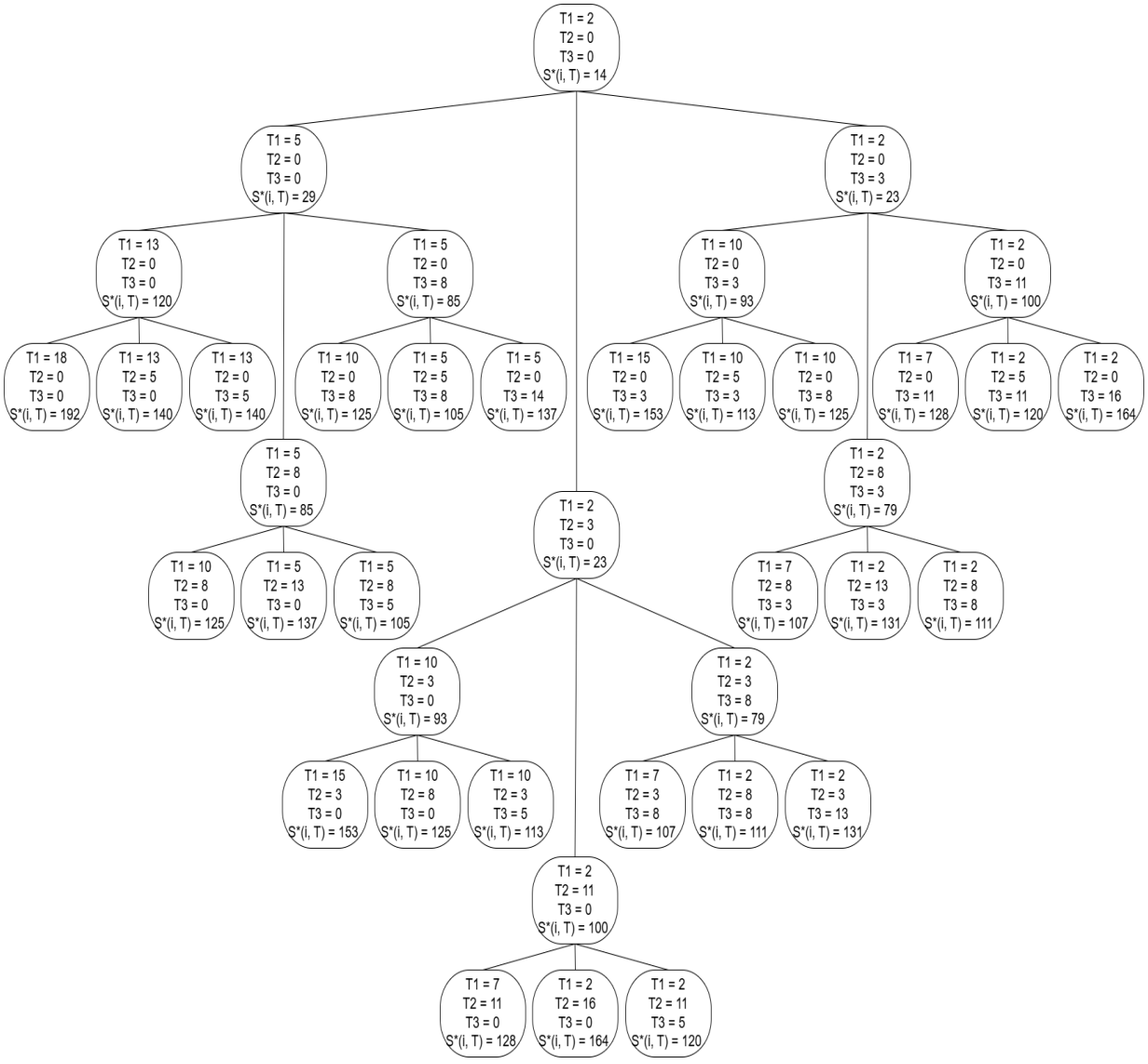


Рисунок 1 – Дерево состояний возможных распределений заявок

В данном примере видно, что деревья образованные средним и правым потомками верхнего узла дают одинаковые результаты. Это говорит о том, что случаи передачи заявки на выполнение свободным специалистам (и специалистам с одинаковой занятостью) можно не рассматривать отдельно, так как дальнейшие результаты у них будут одинаковые. Тогда если в текущем состоянии имеем несколько специалистов с одинаковой занятостью, то рассматриваем только ситуацию передачи следующей заявки только одному из них – специалисту с меньшим номером.

На рисунке 2 представлено сокращенное дерево, построенное с использованием описанного выше нового условия.

Такое решение является достаточно большим даже при небольших количествах заявок и специалистов, например, у дерева для решения задачи с десятью заявками и тремя специалистами будет иметь пятьдесят девять тысяч сорок девять листов. Очевидно, что количество необходимых вычислений с увеличением числа заявок или специалистов растет экспоненциально, поэтому имеет смысл использование приближенных и эвристических алгоритмов.

В дальнейшем, при программной реализации алгоритма, чтобы привести задачу к полиномиальной сложности, можно рассмотреть следующий эвристический алгоритм решения:

а) заявки, упорядоченные по параметру m(i), разбиваются на группы, например, по 10 штук, с сохранением их последовательности, в последней группе их может быть меньше;

б) с помощью точного алгоритма решается задача составления оптимального расписания для первой группы заявок;

в) оптимальное состояние, которое будет решением для текущей группы заявок берется за начальное, и задача решается точным алгоритмом с новым набором заявок;

г) пункт «в» необходимо повторять пока не будут распределены все заявки;

д) все полученные расписания объединяются в порядке их получения.

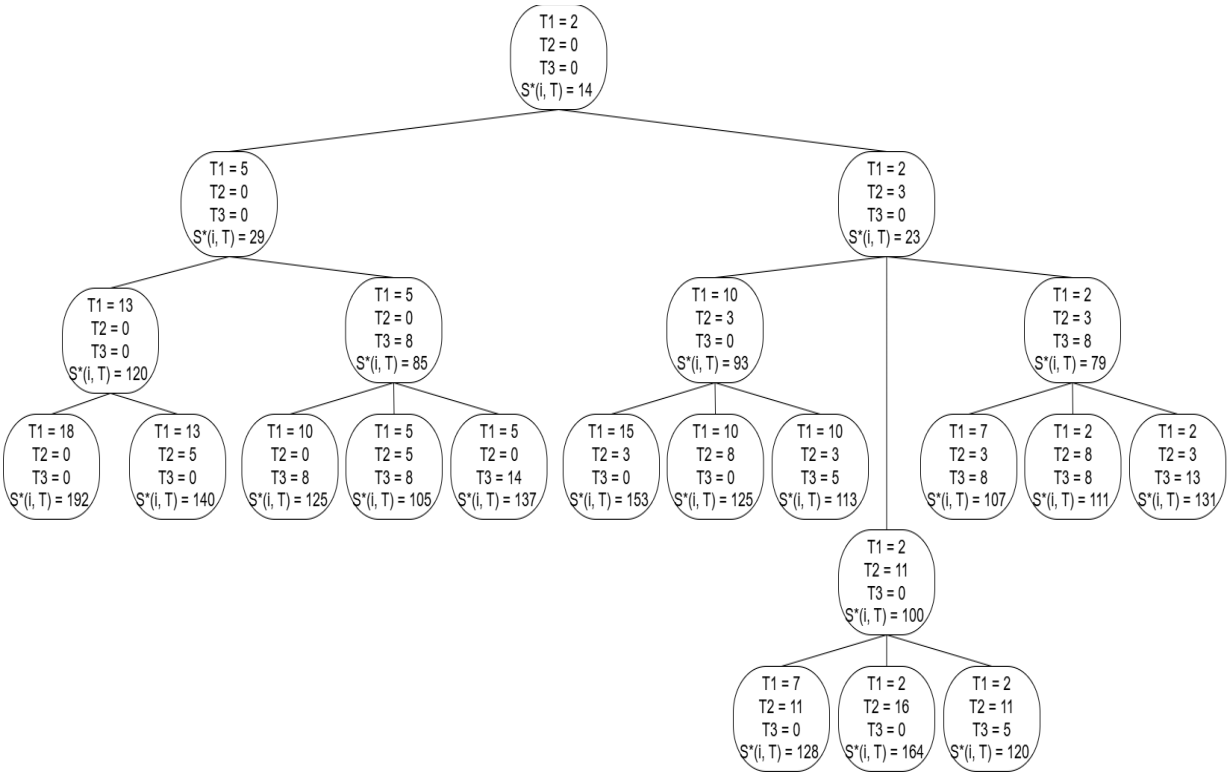


Рисунок 2 – Сокращенное дерево состояний

Вышеописанный метод является неточным и может иметь погрешность больше чем у первого из описанных способов решений, где каждая новая заявка передавалась специалисту, который освободится раньше.

Таким образом были рассмотрены основные методы решения, на основе которых можно реализовать программные алгоритмы для решения задачи синтеза расписаний обслуживания точным и эвристическими методами.

2.3 Описание структуры данных тестовой системы

Перед проектированием алгоритма и его программной реализацией необходимо сначала разработать тестовую систему в которой он будет работать. Тестовая система по своей структуре должна быть максимально похожа на самые распространенные системы оперативного учёта. Для работы алгоритма будут взяты только необходимые для его работы элементы реальных систем. В качестве реальных систем рассматривались системы оперативного учёта на платформе 1С, такие как «Управление торговлей» и «Управление нашей фирмой».

Под оперативным учётом подразумевается процесс сбора, обработки и анализа информации о текущем состоянии и движении ресурсов, материалов, товаров и других активов в организации.

Система оперативного учёта – это комплекс взаимосвязанных методов, средств и технологий, предназначенных для организации и ведения оперативного учёта в компании.

Структура данных тестовой системы описана с помощью диаграммы сущность-связь, представленной на рисунке 3. В ней отражены все объекты и связи между ними необходимые для решения задачи синтеза расписания обслуживания множества заявок. В структуру данных тестовой системы входят следующие сущности:

- «Заявки» – таблица, содержащая основные данные о заявках;
- «Сотрудники» – таблица, в которой хранятся данные о сотрудниках, которые будут выполнять входящие заявки;
- «Услуги» – таблица со всеми возможными услугами организации и оценкой их времени выполнения;
- «УслугиВЗаказе» – вспомогательная таблица, необходимая для реализации связи «многие-ко-многим» между таблицами «Заявки» и «Услуги»;

– «ПриоритетыЗаявок» – таблица, хранящая значение штрафа в единицу времени, который является одним из параметров заявки;
– «СтатусыВыполненияЗаявок» – таблица, содержащая предопределенные данные для обозначения статусов заявки, например, «Не распределена», «В работе» или «Выполнена».

Таблицы «ПриоритетыЗаявок» и «СтатусыВыполненияЗаявок» связаны с таблицей «Заявки» связью «один-ко-многим», при этом статус и приоритет должны быть обязательно заполнены. «Сотрудники» и «Заявки» имеют аналогичную связь, но в данном случае заявка может не иметь исполнителя, что будет означать, что заявка должна быть определена в работу одному из сотрудников, то есть может пойти на вход алгоритма составления расписания.

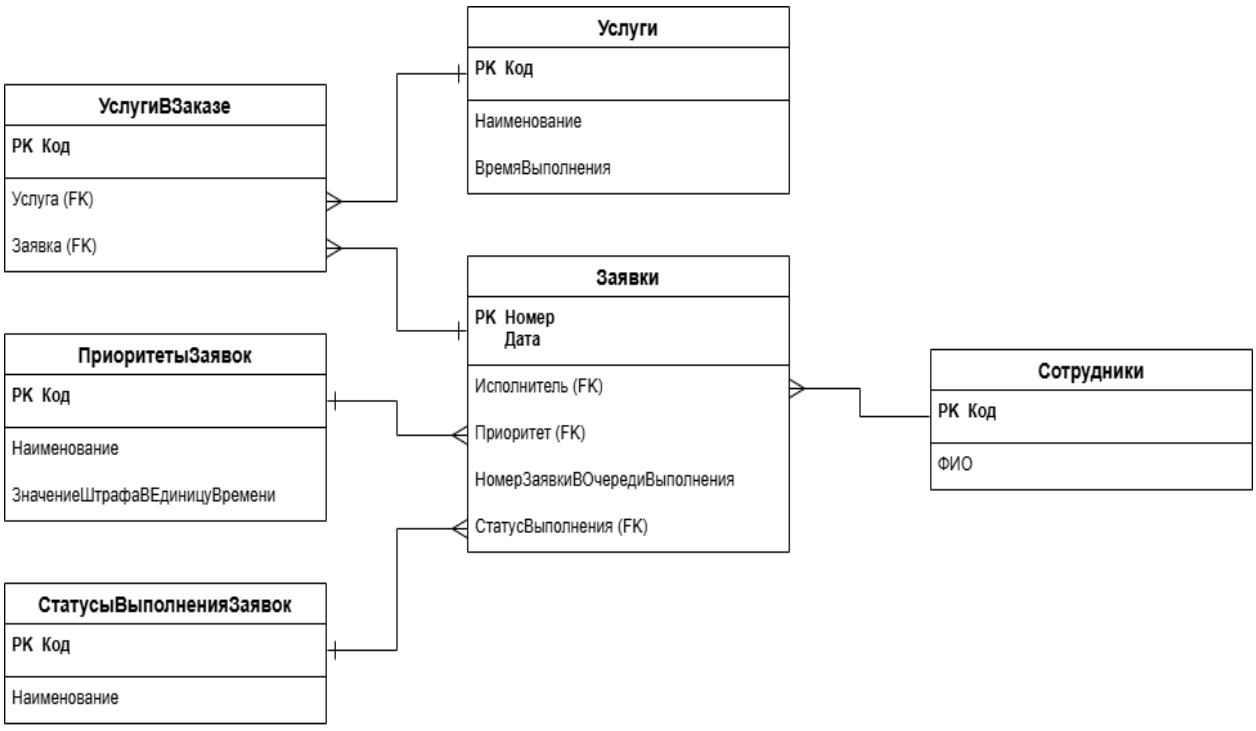


Рисунок 3 – Диаграмма сущность-связь для тестовой системы

Таким образом для получения времени выполнения заявки необходимо с помощью вспомогательной таблицы «УслугиВЗаказе» найти суммарное время выполнения всех услуг, содержащихся в конкретном заказе. А штраф в единицу времени пребывания в системе можно получить исходя из приоритета заявки.

Для обозначения места в очереди выполнения для конкретного сотрудника используется поле «НомерЗаявкиВОчередиВыполнения» таблицы «Заявки». У всех заявок, имеющих одного исполнителя эти номера будут уникальными, для двух заявок с разными исполнителями номер может быть одинаковым.

Для определения занятости сотрудника на данный момент возможно найти суммарное время выполнения всех заявок, в которых он назначен исполнителем, и которые имеют статус выполнения «В работе». Так как невозможно узнать какая часть задачи, выполняемой сотрудником в данный момент, уже сделана, лучшим решением будет брать время выполнения этой задачи полностью.

Тогда входными данными для решения задачи могут быть любые заявки, у которых не заполнен исполнитель, и которые имеют статус выполнения «Не распределена», и любые сотрудники определенные в таблице «Сотрудники».

Таким образом была описана структура данных системы, в которой возможна реализация решения поставленной задачи синтеза расписания обслуживания множества заявок.

2.4 Описание алгоритма работы программы

Получив описанными выше способами данные в виде списка заявок и сотрудников можно переходить к решению задачи.

На рисунке 4 представлена блок-схема алгоритм решения задачи в общем виде.

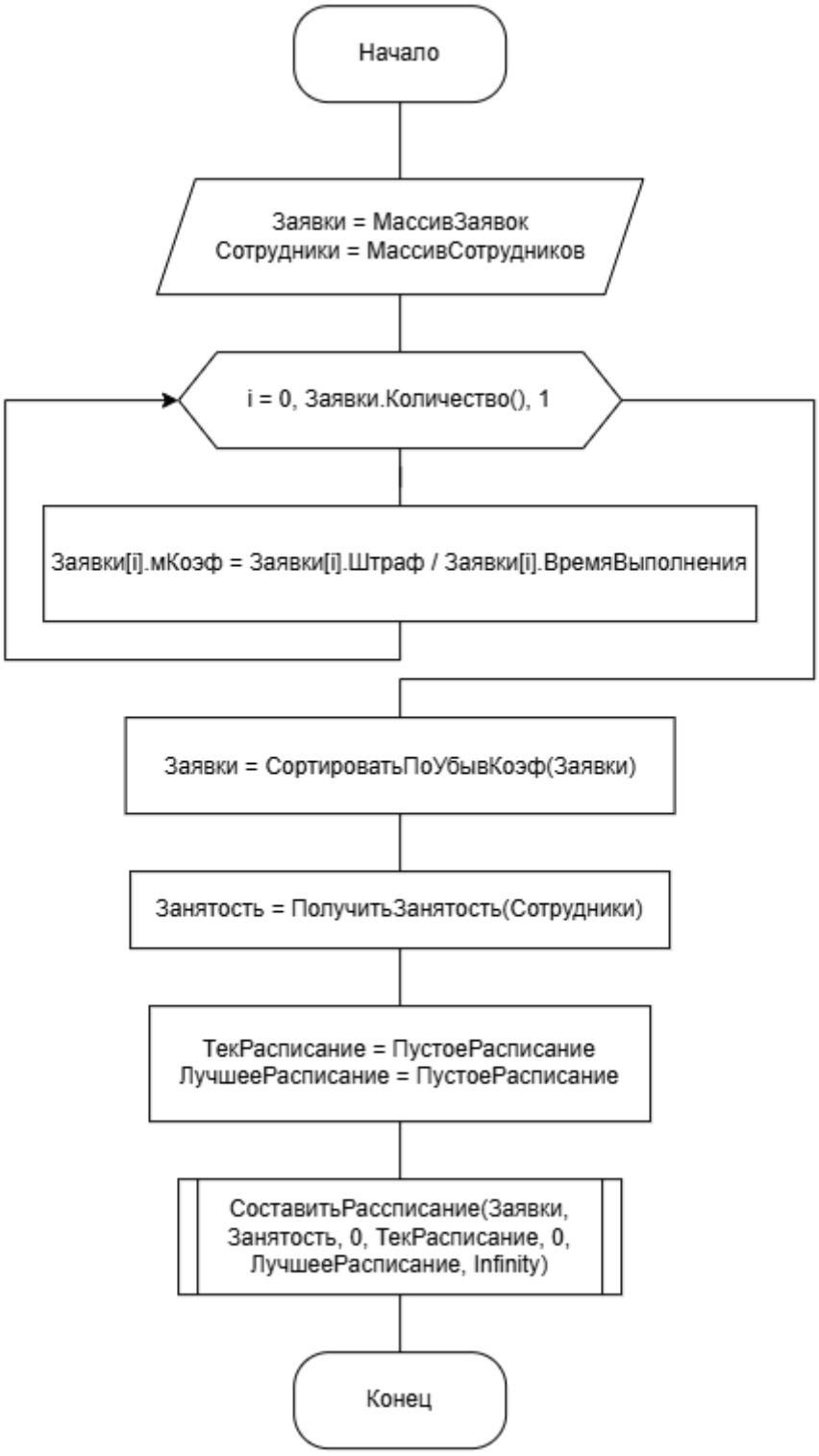


Рисунок 4 – Блок-схема алгоритма синтеза расписания обслуживания

Сначала для каждой заявки рассчитывается параметр m(i). По этому параметру заявки сортируются в порядке убывания. Для сортировки удобно использовать любой из уже реализованных алгоритмов стандартными библиотеками или функциями используемого языка. Чаще всего это алгоритм быстрой сортировки или алгоритм Хоара.

Алгоритм Хоара можно описать следующим образом:

а) из массива выбирается опорный элемент, например, последний;

б) массив разделяется на две части в следующем порядке: сначала идут элементы, большие или равные опорному, а затем элементы, меньшие опорного;

в) для полученных частей массива, длинна которых больше одного элемента, этот же алгоритм применяется рекурсивно.

Для заполнения расписания будет использоваться структура данных, представляющая собой соответствие сотрудника и массива переданных ему заявок.

Далее по описанной ранее структуре данных для каждого сотрудника из полученного на входе списка вычисляется начальная занятость. Эта занятость формируется простой выборкой данных: выбирается время выполнения каждой заявки со статусом в работе и исполнителем которых является сотрудник из списка, после чего выбранные данные группируются по сотрудникам с суммированием времени выполнения.

Для распределения заявок между специалистами используется рекурсивная функция, описанная блок-схемой, представленной на рисунке 5.

На вход при первом вызове функция для распределения заявок получает:

– «Заявки» – массив заявок, со всеми их характеристиками;

– «Занятость» – массив, содержащий начальные занятости сотрудников;

– «Итерация» – целое число, обозначающее какую по счету заявку рассматривает функция в текущем вызове, при первом вызове равное нулю;
– «ТекРасписание» – текущее расписание в которое по ходу работы алгоритма добавляются рассматриваемые заявки;
– «ТекШтраф» – число, равное штрафу при реализации текущего расписания, при первом вызове равен нулю;
– «ЛучшееРасписание» – расписание, включающее в себя все заявки из списка и имеющее минимальный из найденных штраф при реализации;
– «ЛучшийШтраф» – число, равное штрафу при реализации лучшего расписания, при первом вызове равен бесконечности.

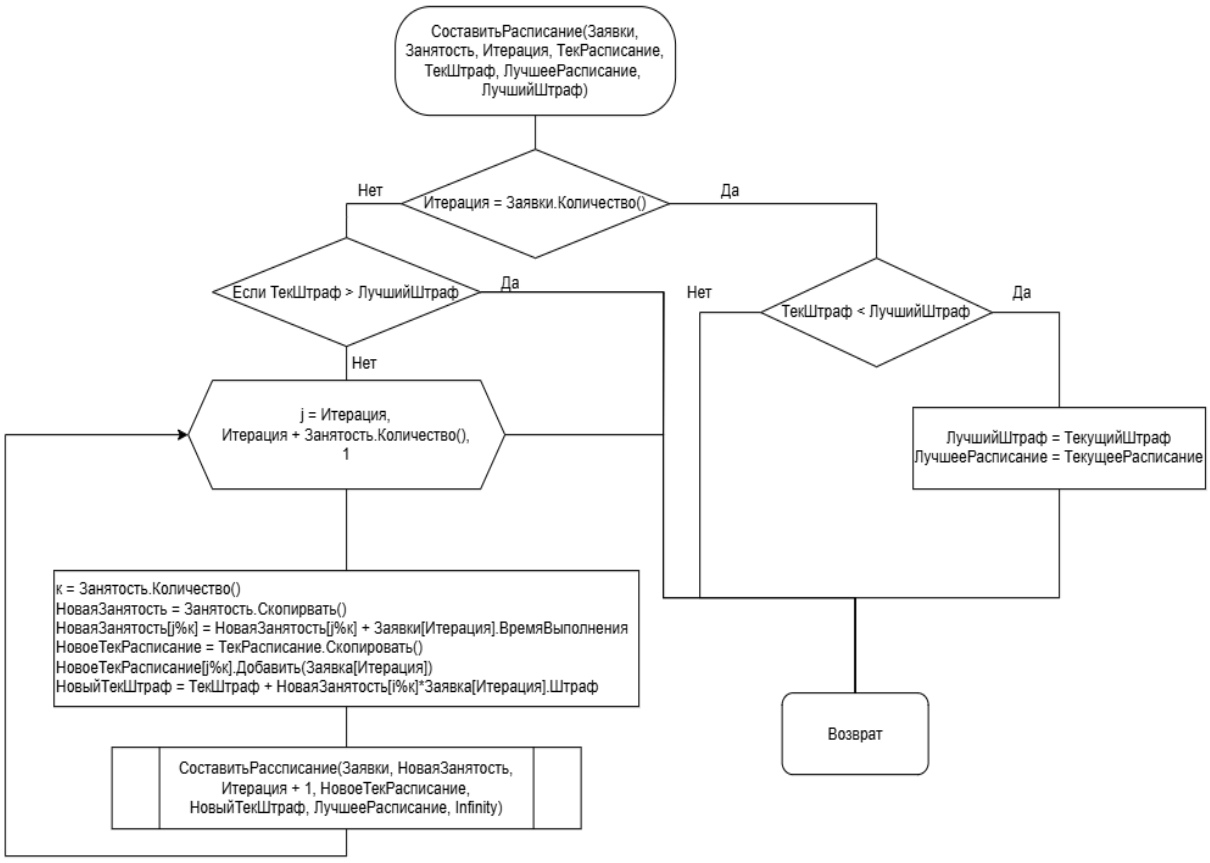


Рисунок 5 – Блок-схема рекурсивной функции распределения заявок

При каждом вызове функции «СоставитьРасписание» выполняется следующий алгоритм: проверяется номер итерации: если итерация имеет значение большее числа заявок, то составление текущего расписания завершено и необходимо проверить является ли оно лучшим на данный момент и завершить выполнение функции. если текущий штраф больше лучшего найденного на данный момент, то выполнение функции также следует завершить. Если функция продолжает выполнение, то необходимо рассмотреть передачу заявки, соответствующей текущей итерации, каждому из сотрудников: создаются новое текущее расписание и новый массив занятости. В расписании сотруднику в список добавляется заявка, а в массив занятости сотруднику прибавляется время выполнения этой заявки. По новым данным рассчитывается новый штраф, и функция вызывается рекурсивно для рассмотрения заявки, соответствующей следующей итерации.

На каждой последующей итерации рекурсивного алгоритма сначала рассматриваются сотрудники, которым было передано меньшее число заявок. То есть на первой итерации цикл начинается с первого сотрудника и заканчивается последним. На второй итерации цикл следует начать со второго сотрудника, а закончить первым и так далее. Когда номер итерации станет больше чем количество сотрудников, цикл снова начнется с первого. При таком способе распределения, когда будет составлено первое условно оптимальное расписание, оно не будет заведомо худшим. Соответственно относительно него можно будет быстрее отсеивать неоптимальные расписания.

Таким образом был описан алгоритм решения поставленной задачи методом динамического программирования в общем виде.

3 Программная реализация

3.1 Выбор технических средств

В качестве средств для разработки описанной выше системы и реализации алгоритма решения задачи синтеза расписания обслуживания множества заявок выбрана технологическая платформа 1С Предприятие 8.3. 1С:Предприятие 8.3 – это технологическая платформа для разработки корпоративных информационных систем, предназначенная для автоматизации различных бизнес-процессов. Система включает в себя средства для разработки прикладных решений, управления данными и автоматизации задач, таких как бухгалтерия, управление персоналом, логистика, производство, CRM.

Технологическая платформа 1С:Предприятие 8.3 представляет собой интегрированную среду разработки и исполнения бизнес-приложений. Она включает в себя: систему управления данными, средства разработки пользовательских интерфейсов и инструменты для обработки данных и реализации бизнес-логики.

Для разработки на платформе используется язык 1С, который представляет собой высокоуровневый язык, поддерживающий как объектно-ориентированное, так и процедурное программирование. Этот язык позволяет реализовывать сложную логику и обеспечивать гибкость в работе с данными.

1С:Предприятие использует многослойную архитектуру, где данные хранятся в реляционных базах данных.

Платформа предлагает широкие возможности для интеграции с внешними системами через стандартные или самостоятельно разработанные интерфейсы. Это позволяет использовать систему в сочетании с другими

программными продуктами, такими как ERP-системы, CRM и другие корпоративные решения.

В таблице 3 приведено сравнение 1С:Предприятие 8.3 с другими популярными платформами для разработки корпоративных информационных систем, такими как Microsoft Dynamics 365, SAP ERP и Oracle ERP.

Таблица 3 – Сравнение платформ для разработки

Платформа	1С:Предприятие 8.3	Microsoft Dynamics 365	SAP ERP	Oracle ERP
Язык разработки	1С (собственный язык)	X++ (собственный язык)	ABAP	PL/SQL
Поддержка БД	MS SQL, PostgreSQL, файловая база данных (возможно использование с другими реляционными СУБД)	MS SQL Server, Azure (системы разработанные Microsoft)	SAP HANA	Oracle DB, MySQL
Цена и доступность	относительно дешевый для среднего и малого бизнеса	высокая стоимость для малых и средних компаний	высокая стоимость, ориентирован на крупные компании	высокая стоимость, часто используется крупным бизнесом

Выбранная для разработки технологическая платформа имеет высокую степень адаптации к российскому законодательству, что делает ее привлекательной для отечественных компаний. В отличие от международных решений, таких как SAP или Oracle, 1С обладает встроенными механизмами для учёта специфики российского налогообложения и отчетности.

Для задачи синтеза расписания обслуживания множества заявок с участием нескольких сотрудников, 1С:Предприятие 8.3 является удобным инструментом благодаря возможностям:

– работать с большими объемами данных, так как платформа предназначена для эффективной обработки больших объемов данных

и предоставляет инструменты для работы с базой данных и оптимизации запросов;

- реализовать логику синтеза расписания с использованием встроенных средств разработки, что позволяет обеспечить автоматическую проверку условий и корректировку расписания;
- настроить интерфейс для пользователя и интегрировать систему с другими инструментами для обмена данными.

Исходя из описанных выше преимуществ платформы 1С:Предприятие 8.3 можно сделать вывод, что 1С:Предприятие 8.3 является мощной и гибкой платформой для решения широкого спектра задач в области автоматизации бизнес-процессов. В контексте задачи синтеза расписания обслуживания заявок, она предоставляет все необходимые инструменты для реализации сложных логик, работы с данными и взаимодействия с пользователем. Сравнение с аналогами показывает, что 1С идеально подходит для применения в России благодаря своей адаптации под местные требования, доступности и гибкости, в отличие от более дорогих и сложных решений таких как SAP или Oracle.

3.2 Разработка программы

На основе ER-Диаграммы, представленной на рисунке 3 и описанной во втором разделе, была разработана структура из справочников, документов и перечислений, которые представляют собой объекты платформы 1С – справочники, документы и перечисления, описывающие нужные таблицы.

Справочник – это объект метаданных, который используется для хранения информации о типовых данных, применяемых в системе. Обычно справочники содержат списки, например, контрагентов, товаров, услуг, и позволяют организовать данные в иерархическом виде.

Документ – это объект метаданных, который фиксирует определенные события или операции в системе. Документы часто используются для учета, например, продаж, закупок, платежей, и содержат информацию о времени, участниках и данных операции.

Перечисление – это также объект метаданных, реализующий фиксированный набор значений. Перечисления обеспечивают возможность выбора из предопределенного списка данных, например, статусов, типов документов и так далее.

Все созданные объекты представлены на рисунке 6.

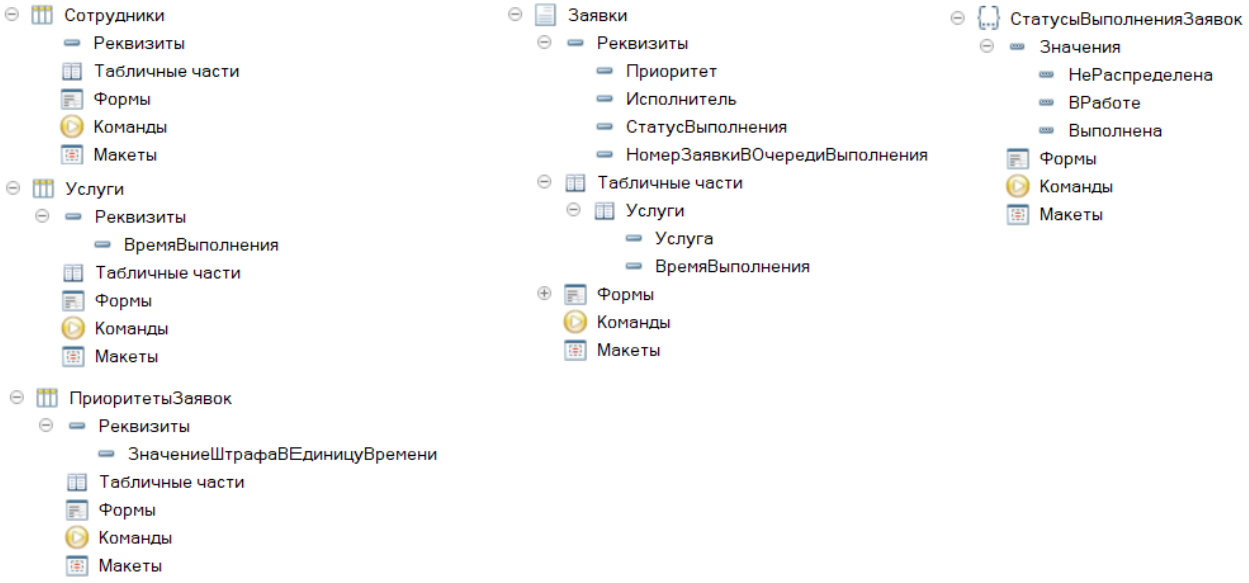


Рисунок 6 – Созданные в 1С объекты

Таблицы сотрудников, услуг и видов приоритетов заявок представлены в виде справочников, так как они отражают постоянную информацию и должны быть редактируемыми. Заявка является документом, так как отражает событие – выполнение услуги. Услуги в заявке добавлены в виде табличной части документа. И статусы выполнения заявок являются

перечислением, так как значения таблицы предопределены и их редактирование не имеет смысла.

Для решения задачи синтеза расписания обслуживания множества заявок разработана обработка. Обработка в 1С – это прикладной объект конфигурации, предназначенный для выполнения специализированных задач, которые не покрываются стандартными возможностями системы.

Обработка состоит из двух форм интерфейса: для вывода списков выбора заявок для распределения и специалистов, которым будут переданы заявки на выполнение и для вывода результата работы алгоритма – расписаний специалистов.

На рисунке 7 показан листинг кода на языке 1С, соответствующего реализации алгоритма, описанного в блок-схеме из рисунка 4, которая была спроектирована в рамках второго раздела.

Здесь, как и было описано, на вход алгоритм получает две таблицы значений: заявок и специалистов. При этом для таблицы заявок на данном шаге уже вычислены коэффициенты m, и строки таблицы отсортированы по убыванию этого коэффициента.

Далее по данным информационной базы для каждого сотрудника вычисляется суммарное время заявок, находящихся на выполнении на данный момент, и номер последней заявки в очереди на выполнение, необходимый для того чтобы по ходу распределения для новых заявок присвоить новые номера.

После создаются два пустых расписания, содержащих только список сотрудников, для дальнейшей записи частичных и условно оптимальных решений. Расписание имеет тип «Дерево значений». Дерево значений – это структура, представляющая собой иерархическую модель, используемую для организации и хранения значений определённого типа.

```
&НаСервере
Процедура НачатьСоставлениеРассписания(ТЗСотрудники, ТЗЗаявки)
    Запрос = Новый Запрос("ВЫБРАТЬ
    |   СУММА(ВЫБОР
    |           КОГДА ЗаявкиУслуги.Ссылка.СтатусВыполнения = &СтатусВыполнения
    |               ТОГДА ЗаявкиУслуги.ВремяВыполнения
    |           ИНАЧЕ 0
    |       КОНЕЦ) КАК ВремяВыполнения,
    |   ЗаявкиУслуги.Ссылка.Исполнитель КАК Исполнитель,
    |   МАКСИМУМ(ЗаявкиУслуги.Ссылка.НомерЗаявкиВОчередиВыполнения) КАК НомерЗаявкиВОчередиВыполнения
    |ИЗ
    |   Документ.Заявки.Услуги КАК ЗаявкиУслуги
    |ГДЕ
    |   ЗаявкиУслуги.Ссылка.Исполнитель В ИЕРАРХИИ(&Исполнитель)
    |
    |СГРУППИРОВАТЬ ПО
    |   ЗаявкиУслуги.Ссылка.Исполнитель");
    Запрос.УстановитьПараметр("Исполнитель", ТЗСотрудники.ВыгрузитьКолонку("Сотрудник"));
    Запрос.УстановитьПараметр("СтатусВыполнения", Перечисления.СтатусыВыполненияЗаявок.ВРаботе);
    ЧасыРаботыСотр = Запрос.Выполнить().Выбрать();
    Пока ЧасыРаботыСотр.Следующий() Цикл
        Для Каждого Стр Из ТЗСотрудники Цикл
            Если ЧасыРаботыСотр.Исполнитель = Стр.Сотрудник Тогда
                Стр.Занятость = ЧасыРаботыСотр.ВремяВыполнения;
                Стр.НомерПоследнейЗаявки = ЧасыРаботыСотр.НомерЗаявкиВОчередиВыполнения;
            КонецЕсли;
        КонецЦикла;
    КонецЦикла;
    ТекРасписание = Новый ДеревоЗначений(); //Шаблон для расписания
    ТекРасписание.Колонки.Добавить("Сотрудник", Новый ОписаниеТипов("СправочникСсылка.Сотрудники"));
    ТекРасписание.Колонки.Добавить("Заявка", Новый ОписаниеТипов("ДокументСсылка.Заявки"));
    ТекРасписание.Колонки.Добавить("НомерВыполнения", Новый ОписаниеТипов("Число"));
    Для Каждого Сот Из ТЗСотрудники Цикл
        НС = ТекРасписание.Строки.Добавить();
        НС.Сотрудник = Сот.Сотрудник;
    КонецЦикла;
    ЛучшееРасписание = ТекРасписание.Скопировать();
    СоставитьРассписание(ТЗЗаявки, ТЗСотрудники, 0, ТекРасписание, 0, ЛучшееРасписание, 9999999999);
    ЗначениеВРеквизитФормы(ЛучшееРасписание, "Расписание");
КонецПроцедуры
```

Рисунок 7 – Реализация алгоритма синтеза расписаний

После подготовки всех данных вызывается рекурсивная функция для распределения заявок между специалистами, которая была реализована на основе второй блок-схемы алгоритма (рисунок 5), которая была спроектирована во втором разделе работы. Функция рекурсивного распределения заявок представлена на рисунке 8. Здесь реализованы все те же действий, которые уже были описаны в соответствующей этой функции блок-схеме алгоритма: заявки берутся по одной и поочередно передаются разным специалистам, а после обработки последней заявки из списка

происходит проверка оптимальности полученного на данном шаге расписания.

```
&НаСервере
Процедура СоставитьРассписание(Заявки, Занятость, Итерация,
                               ТекРасписание, ТекШтраф,
                               ЛучшееРасписание, ЛучшийШтраф)
    Если Итерация = Заявки.Количество() Тогда
        Если ТекШтраф < ЛучшийШтраф Тогда
            ЛучшееРасписание = ТекРасписание.Скопировать();
            ЛучшийШтраф = ТекШтраф;
        КонецЕсли;
        Возврат;
    КонецЕсли;
    Если ТекШтраф > ЛучшийШтраф Тогда
        Возврат;
    КонецЕсли;
    РассмотренныеЗанятости = Новый Соответствие();
    Для Инд = Итерация По Итерация + Занятость.Количество() - 1 Цикл
        Если РассмотренныеЗанятости.Получить(Занятость[Инд % Занятость.Количество()].Занятость) = Неопределено Тогда
            РассмотренныеЗанятости.Вставить(Занятость[Инд % Занятость.Количество()].Занятость, 1);
        Иначе
            Продолжить;
        КонецЕсли;
        НоваяЗанятость = Занятость.Скопировать();
        НоваяЗанятость[Инд % Занятость.Количество()].Занятость = НоваяЗанятость[Инд % Занятость.Количество()].Занятость
                               + Заявки[Итерация].ВремяВыполнения;
        НоваяЗанятость[Инд % Занятость.Количество()].НомерПоследнейЗаявки =
                               НоваяЗанятость[Инд % Занятость.Количество()].НомерПоследнейЗаявки + 1;
        НовыйТекШтраф = ТекШтраф +
                               (НоваяЗанятость[Инд % Занятость.Количество()].Занятость *
                               Заявки[Итерация].Штраф);
        НовоеТекРасписание = ТекРасписание.Скопировать();
        ЗаявкаВРасписании = НовоеТекРасписание.Строки[Инд % Занятость.Количество()].Строки.Добавить();
        ЗаявкаВРасписании.Заявка = Заявки[Итерация].Заявка;
        ЗаявкаВРасписании.НомерВыполнения = НоваяЗанятость[Инд % Занятость.Количество()].НомерПоследнейЗаявки;
        СоставитьРассписание(Заявки, НоваяЗанятость, Итерация+1,
                             НовоеТекРасписание, НовыйТекШтраф,
                             ЛучшееРасписание, ЛучшийШтраф);
    КонецЦикла;

КонецПроцедуры
```

Рисунок 8 – Реализация рекурсивной функции распределения заявок

После выполнения рекурсивного алгоритма программой будет получено оптимальное расписание, при котором штраф становится минимальным, и само значение штрафа, при реализации оптимального расписания. Это расписание будет выведено как рекомендация к распределению в виде таблицы, состоящей из двух уровней иерархии: на первом будут выведены специалисты, а на втором их заявки.

3.3 Тестирование программного продукта

Для проверки работоспособности программы можно использовать данные для решенной во втором разделе задачи, где заявки из таблицы 1 были распределены между тремя специалистами. Для этого в базу данных предварительно вводятся данные о заявках и специалистах. Тогда форма выбора заявок и специалистов выглядит как показано на рисунке 9.

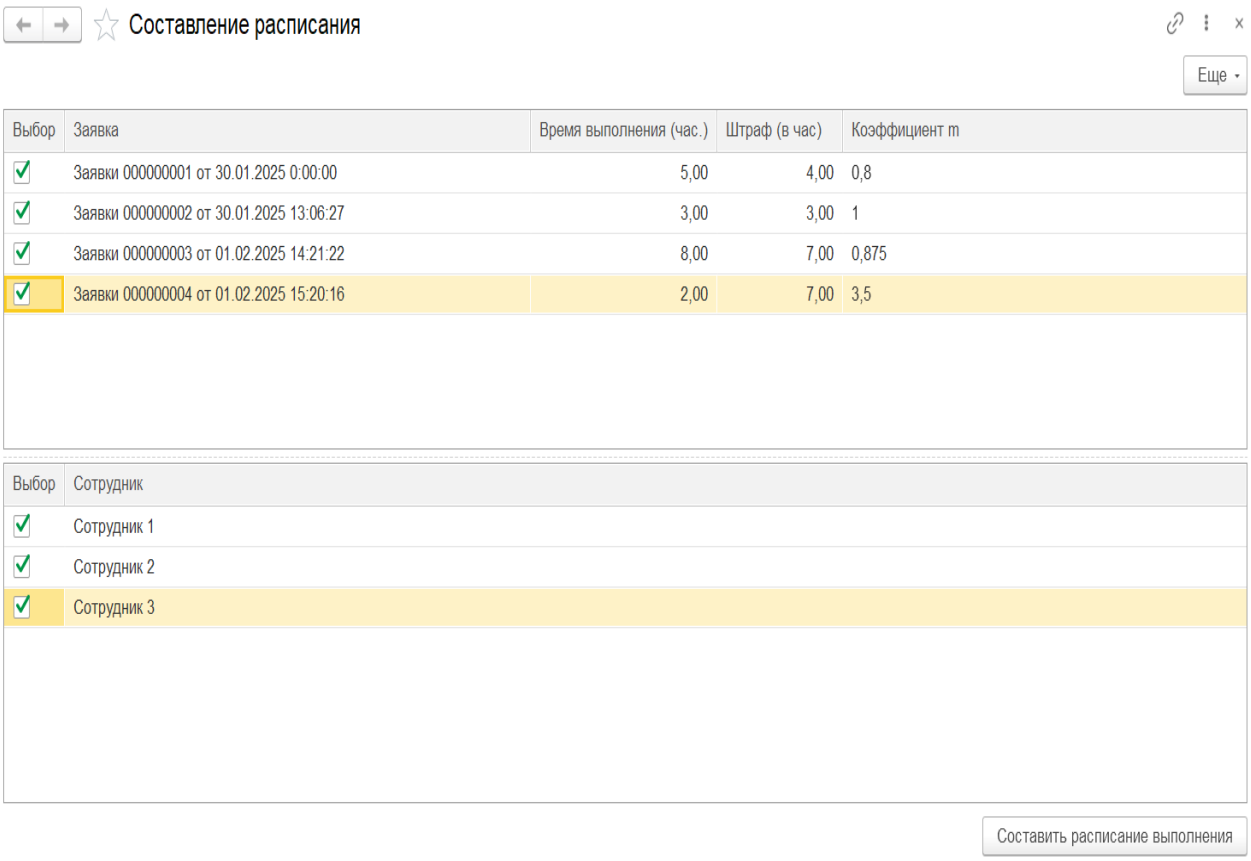


Рисунок 9 – Форма выбора

По введенным данным формируется расписание в виде таблицы, оно представлено на рисунке 10.

Результат работы программы полностью совпадает с полученным во втором разделе решением. Таким образом программа работает корректно и достаточно эффективно для задач с небольшими размерностями.

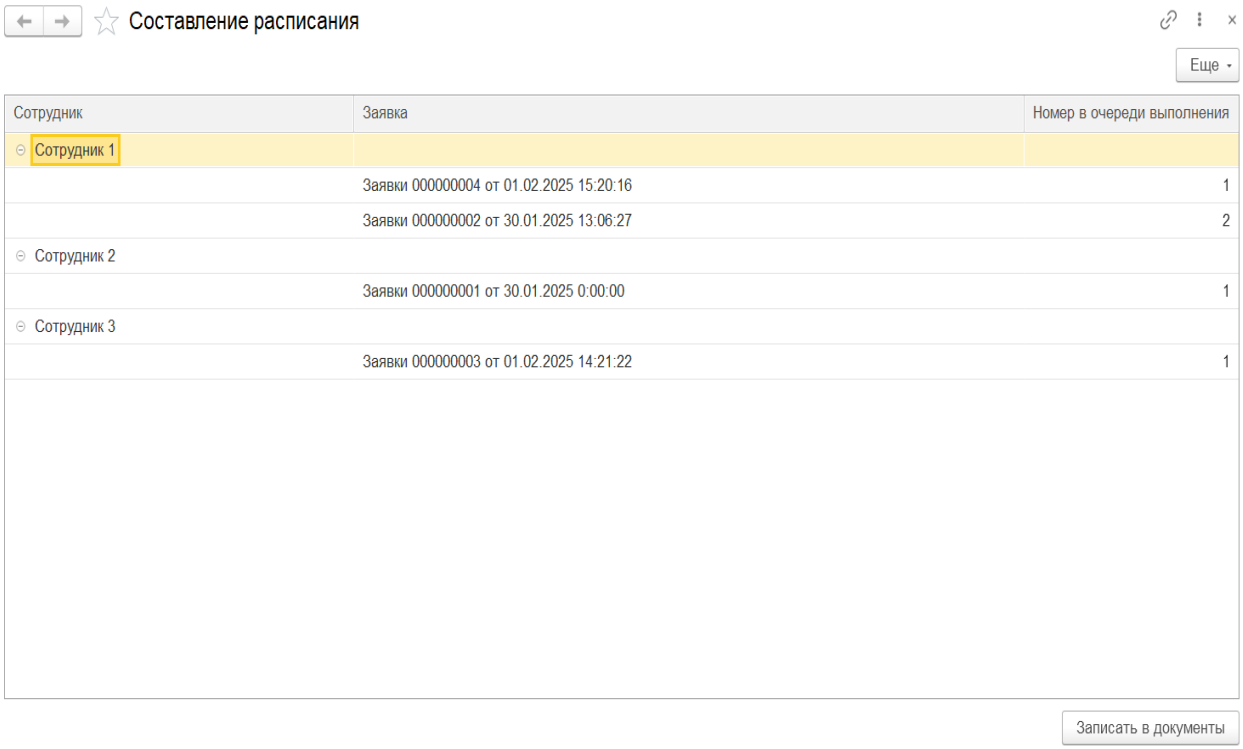
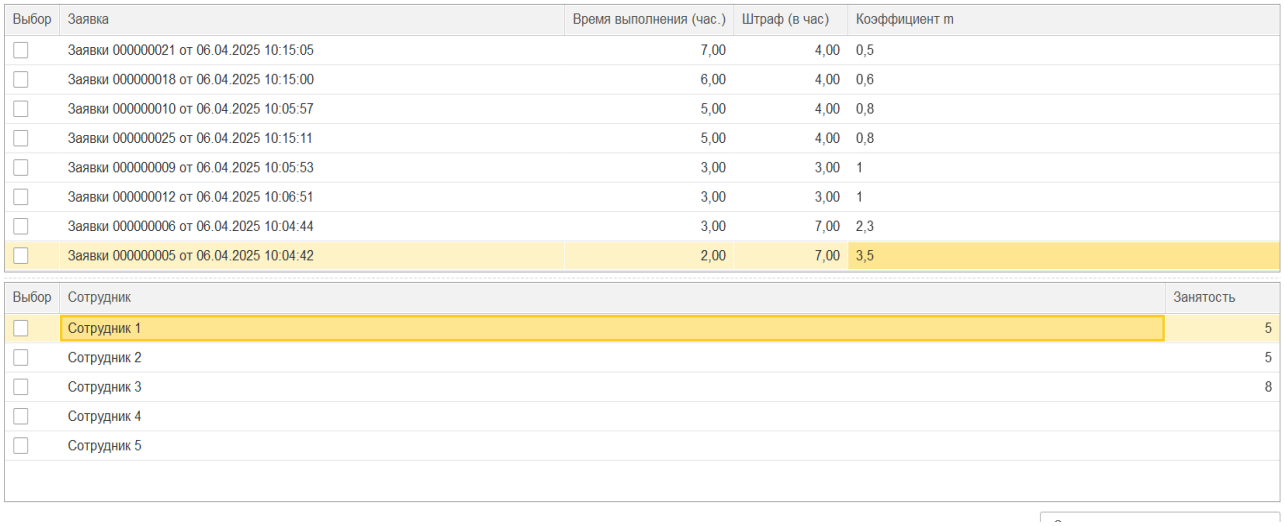


Рисунок 10 – Результат работы программы

Тестирование программы на больших размерностях показывает, что почти моментально можно получить решение задачи при распределении до двадцати заявок между двумя специалистами или до двенадцати заявок между пятью специалистами, чего в большинстве случаев на практике достаточно.

Но при необходимости решения задачи большей размерности, как и было описано ранее, можно решать задачу по частям: заранее отсортировать таблицу параметру m и при каждом решении выбирать только ту часть заявок для распределения, у которых коэффициенты m больше.

Для решения задач большого количества заявок можно доработать пользовательский интерфейс следующим образом: выборка заявок на первой форме должна быть сразу отсортирована, а выбор пользователем количества заявок ограничить до оптимального количества, которое будет зависеть от количества выбранных специалистов. В случаях, когда необходимо выбрать большое количество специалистов возможно также предусмотреть ограничение по количеству выбранных на первой форме специалистов. Тогда пользователь будет иметь возможность выполнять алгоритм по несколько раз, выбирая для распределения только самые важные задачи и наименее загруженных специалистов. Доработанная форма выбора заявок и специалистов представлена на рисунке 11.



Выбор	Заявка	Время выполнения (час.)	Штраф (в час)	Коэффициент m
☐	Заявки 000000021 от 06.04.2025 10:15:05	7,00	4,00	0,5
☐	Заявки 000000018 от 06.04.2025 10:15:00	6,00	4,00	0,6
☐	Заявки 000000010 от 06.04.2025 10:05:57	5,00	4,00	0,8
☐	Заявки 000000025 от 06.04.2025 10:15:11	5,00	4,00	0,8
☐	Заявки 000000009 от 06.04.2025 10:05:53	3,00	3,00	1
☐	Заявки 000000012 от 06.04.2025 10:06:51	3,00	3,00	1
☐	Заявки 000000006 от 06.04.2025 10:04:44	3,00	7,00	2,3
☐	Заявки 000000005 от 06.04.2025 10:04:42	2,00	7,00	3,5

Выбор	Сотрудник	Занятость
☐	Сотрудник 1	5
☐	Сотрудник 2	5
☐	Сотрудник 3	8
☐	Сотрудник 4	
☐	Сотрудник 5	

Рисунок 11 – Доработанная форма выбора заявок и специалистов

Таким образом была разработана программа для решения задачи синтеза расписания обслуживания множества заявок.

Заключение

В рамках выпускной квалификационной работы рассмотрены ключевые аспекты синтеза расписаний обслуживания множества заявок, исследованы проблемы, стоящие перед современными организациями. Введение в тему подчеркнуло актуальность оптимизации процессов в условиях ограниченных ресурсов и возрастающих требований со стороны клиентов.

В первом разделе выпускной квалификационной работы рассматривается общая характеристика задачи синтеза расписания; выполнен анализ существующих проблем в области создания расписаний; определены требования к разработанной в рамках работы системе. Далее проведена постановка задачи синтеза расписания, где обозначены основные цели и направления работы. После определения характеристик задачи, в данном разделе представлен обзор различных методов решения, которые используются в практике.

Второй раздел посвящен разработке общего алгоритма решения задачи синтеза расписания. Сначала описан простейший случай – составление расписания для одного специалиста, чтобы наглядно продемонстрировать основные принципы работы алгоритма. Затем разобрана более сложная ситуация, когда необходимо учитывать множество специалистов и работать с большим количеством заявок. Раздел также предполагает описание структуры данных, используемых в тестовой системе и алгоритма синтеза расписаний.

Третий раздел посвящен программной реализации разработанных алгоритмов. Здесь описывается выбор технических средств, необходимых для реализации программы, а также процесс разработки самого программного обеспечения. Результаты работы программы представлены в виде форм разработанного пользовательского интерфейса с демонстрацией

успешного решения задачи синтеза расписания обслуживания множества заявок.

Таким образом, уделив внимание общему характеру задачи и представив основные методы ее решения, разработан алгоритм, обеспечивающий оптимальное расписание для работы специалистов, обслуживающих множество поступивших заявок.

Данная работа подчеркивает важность правильного планирования и систематизации процессов, что является основой для успешного функционирования бизнеса в любой сфере. Результаты данного исследования могут послужить базой для дальнейших исследований в области синтеза расписаний. В будущем исследование в данной области могут быть продолжены, фокусируясь на разработке более сложных алгоритмов.

Список используемой литературы

1 Булатов А., Куликов А. Основы языка программирования 1С:Предприятие 8. Краткий курс. - М.: БХВ-Петербург, 2017. - 352 с.

2 Быстров А., Карташов Д., Зотов Н. 1С:Предприятие 8.3. Информационная база и программный код. - СПб.: БХВ-Петербург, 2016. - 320 с.

3 Визгунов Н.П. Динамическое программирование в экономических задачах с применением системы SciLab. - 1-е изд. - Нижний Новгород: Нижегородский государственный университет им. Н. И. Лобачевского, 2011

4 Коган Д. И. Задачи и методы конечномерной оптимизации Часть 3. Динамическое программирование и дискретная многокритериальная оптимизация / Новгород: Издательство Нижегородского госуниверситета, 2004 — 258 с.

5 Кристофидес Н. Теория графов. Алгоритмический подход / Н. Кристофидес. – М.: Мир, 1978. – 432 с.

6 Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. - М.: Вильямс, 2005. – 1296 с.

7 Норенков И.П. Основы автоматизированного проектирования / И.П. Норенков. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2009. – 440 с.

8 Попов Р.С. Разработка алгоритмов построения расписаний для систем массового обслуживания. - СПб.: Лань, 2017. – 256 с.

9 Смирнов П.И. Модели и алгоритмы управления ресурсами в системах обслуживания. - М.: Радио и связь, 2008. – 368 с.

10 Сосина Н.А. Исследование операций. Электронное учебное пособие. В 2-х частях. Часть 2/ Тольятти: ФГБОУ ВО «Тольяттинский государственный университет», 2023г. Электронный ресурс, 3,3 Мб. ISBN 978-5-8259-1045-1. С. 7-67.

11 Танаев , В. С. Введение в теорию расписаний / В. С. Танаев, В. В. Шкурба. - М.: Наука, 1975. - 256 с.

12 Федоров Е.Н. Алгоритмы и методы оптимизации расписаний в транспортных системах. - М.: Транспорт, 2003. - 304 с.

13 Филатова В.О., 1С:Предприятие 8.3. Бухгалтерия предприятия, Управление торговлей, Управление персоналом / В.О Филатова. - 3-е. - Санкт-Петербург : Питер, 2014. - 240 с.

14 Фофанов С.А. Оптимизация алгоритма календарного планирования с помощью динамического программирования / Фофанов С.А. [Электронный ресурс] // Тезисы докладов L конференции. Естественные и технические науки (2024 г.) : [сайт]. - URL: https://osnk-sr.ru/theses/index.html - с. 390 (дата обращения: 05.02.2025).

15 Bertsekas, D. P. (2017), Dynamic Programming and Optimal Control (4th ed.), Athena Scientific.

16 Manuel Laguna, Johan Marklund Business process modeling, simulation and design: textbook. New York: CRC Press, 2019. 526 с.

17 Progress in inventory research. //Proc. of the 4-th Internat. Symp Budapest: Akad. Kiado, 1989. - 446 p.

18 Winston W.L. Introduction to Mathematical Programming: Applications and Algorithms. Boston (Mass.): PWS–KENT Publ., 1991. - 794 p.

19 Xiaoxi Chen. A Comparison of Greedy Algorithm and Dynamic Programming Algorithm (2023 год) // Researchgate URL: https://www.researchgate.net/publication/372342141_A_segment-wise_dynamic_programming_algorithm_for_BSDEs (дата обращения: 03.12.2024).

20 Yunong Zhang. A survey of dynamic programming algorithms (2023 год) // Researchgate URL: https://www.researchgate.net/publication/377936794_A_survey_of_dynamic_programming_algorithms (дата обращения: 03.12.2024).

21 Yuntao Bai, Di Liu, Jili Ma Centralized scheduling, decentralized scheduling or demand scheduling? How to more effectively allocate and recycle shared takeout lunch boxes / Yuntao Bai, Di Liu, Jili Ma (2025 год) // Researchgate - URL: https://www.researchgate.net/publication/389554925_Centralized_scheduling_decentralized_scheduling_or_demand_scheduling_How_to_more_effectively_allocate_and_recycle_shared_takeout_lunch_boxes (дата обращения: 30.05.2025).