**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра дискретной математики и алгоритмики**

АЗЯВЧИКОВ
Алексей Павлович

# МЕТОДЫ РЕШЕНИЯ ЗАДАЧИ О НАЗНАЧЕНИЯХ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ

Дипломная работа

Научный руководитель:
старший преподаватель
Г.П. Волчкова

Допущена к защите

«____» _____ 2025 г.

Заведующий кафедрой дискретной математики и алгоритмики,
доктор физико-математических наук, профессор В.М. Котов

Минск, 2025

# ОГЛАВЛЕНИЕ

# РЕФЕРАТ

Дипломная работа, 48 с., 33 таблицы, 4 иллюстрации, 3 приложения, 6 источников.

Ключевые слова: ОНЛАЙН-АЛГОРИТМЫ, ЗАДАЧА О НАЗНАЧЕНИЯХ, ОНЛАЙН-ПЛАНИРОВЩИКИ, ЭВРИСТИЧЕСКИЕ АЛГОРИТМЫ, C++.

Объект исследования: онлайн-алгоритмы о назначениях.

Цель работы: разработка автоматической системы распределения задач между работниками в режиме онлайн и покрытие её тестами.

Результат: было разработано четыре алгоритма, решающих задачу о назначениях в режиме реального времени, проведено тестирование их корректной работы и оценено качество полученных решений.

Область применения: сервисы заказа такси, распределение задач в облачных вычислениях, повышение эффективности логистики в цепях поставок, сервисы доставок.

# РЭФЕРАТ

Дыпломная праца, 48 с., 33 табліцы, 4 ілюстрацыі, 3 дадаткі, 6 крыніц.

Ключавыя словы: АНЛАЙН-АЛГАРЫТМЫ, ЗАДАЧА А ПРЫЗНАЧЭННЯХ, АНЛАЙН-ПЛАНІРОЎШЧЫКІ, ЭВРЫСТЫЧНЫЯ АЛГАРЫТМЫ, C++.

Аб'ект даследавання: анлайн-алгарытмы прызначэнняў.

Мэта работы: распрацоўка аўтаматычнай сістэмы размеркавання заданняў паміж работнікамі ў рэжыме анлайн і пакрыццё яе тэстамі.

Вынік: распрацаваны чатыры алгарытмы, якія вырашаюць задачу прызначэнняў у рэжыме рэальнага часу, праведзена тэставанне іх карэктнай працы і ацэнена якасць атрыманых рашэнняў.

Вобласць ужывання: сэрвісы замовы таксі, размеркаванне задач у воблачных вылічэннях, павышэнне эфектыўнасці лагістыкі ў ланцужках паставак, сэрвісы дастаўкі.

# ABSTRACT

Diploma thesis, 48 pages, 33 tables, 4 figures, 3 appendices, 6 references.

Keywords: ONLINE ALGORITHMS, ASSIGNMENT PROBLEM, ONLINE SCHEDULERS, HEURISTIC ALGORITHMS, C++.

Object of research: online assignment algorithms.

Objective: development of an automated task allocation system for workers in real time and comprehensive testing of the system.

Result: Four algorithms solving the assignment problem in real time were developed. Their correct operation was tested, and the quality of the obtained solutions was evaluated.

Application area: taxi ordering services, task allocation in cloud computing, supply chain logistics optimization, delivery services.

# ВВЕДЕНИЕ

В современном мире, характеризующемся динамично меняющимися условиями и высокой конкуренцией, эффективное управление ресурсами является важным фактором успеха любой организации. В этом контексте особенно актуальной становится задача оптимального распределения задач между сотрудниками, позволяющая максимизировать производительность труда наемных рабочих. Традиционные методы планирования и распределения задач, основанные на статическом анализе и ручном управлении, часто оказываются неэффективными в условиях постоянно поступающих новых задач и изменяющейся доступности работников. Автоматизированные же методы планирования позволяют не только повысить гибкость управления, но и значительно сократить издержки, связанные с простоями и неравномерной загрузкой персонала. Проведение исследований в этой области открывает новые возможности для совершенствования бизнес-процессов на промышленных предприятиях, в сфере услуг, а также в организациях с высокой степенью неопределённости и изменчивости задач.

Целью данной работы является разработка автоматической системы распределения задач между работниками в режиме онлайн, создание для неё тестов и оценка временны́х характеристик основных операций, поддерживаемых планировщиком.

Задачами исследования в рамках данной работы являются: разработка алгоритмов решения задачи о назначениях в режиме реального времени, реализация автоматической системы моделирования пользовательской нагрузки, тестирование корректности разработанных алгоритмов и оценка их качества с использованием системы моделирования пользовательской нагрузки.

# ГЛАВА 1
# ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И ПОНЯТИЯ

## 1.1 Основные понятия

**Определение 1.** *Задача* – единица работы, требующая исполнения. Характеризуется типом, вероятностью появления и временем исполнения.

**Определение 2.** *Работник* – исполнитель задач. Характеризуется: типом, временем прихода на работу и временем ухода с работы. Тип работника определяет набор его навыков, то есть набор типов задач, которые работник способен выполнять.

**Определение 3.** *Онлайн-алгоритм* – алгоритм, обрабатывающий входные данные (поступающие задачи и приходящие работники) последовательно, без знания будущих данных [1].

**Определение 4.** *Оффлайн-алгоритм* – алгоритм, имеющий доступ ко всем входным данным (всем задачам и работникам) заранее, до начала процесса распределения [2].

## 1.2 Постановка задачи

Пусть $Tasks = \{t_1, t_2, \ldots, t_n\}$ – множество задач, поступающих последовательно. Каждая задача $t_i$ характеризуется своим типом $type(t_i) \in Types$, где $Types$ – множество всех возможных типов задач, и временем появления. Время выполнения каждой задачи одинаково для любого работника, способного выполнять данный тип задачи. Время выполнения задач разных типов одинаково.

Пусть $Workers = \{w_1, w_2, \ldots, w_m\}$ – множество работников. Для каждого работника $w_j$ определено множество $WorkerAbilities(w_j) \subseteq Types$ – типы задач, которые он способен выполнять, $T_{in}$ – время прихода на работу, $T_{out}$ – время ухода с работы. Для всех работников разница между $T_{out}$ и $T_{in}$ одинакова и в дальнейшем называется продолжительностью рабочего дня.

Тогда требуется в момент появления каждой задачи немедленно назначить её одному из доступных работников, такому, чтобы тип задачи соответствовал одной из компетенций работника. Не допускается назначать работнику задачу, если ранее ему уже была назначена задача, которую тот не завершил (т.е. прошло меньше времени, чем время выполнения задачи), либо если рабочий день сотрудника завершился.

Задачей данной работы является реализация онлайн-планировщика на языке C++, который распределяет задачи по доступным работникам, максимизировав количество выполненных задач.

Другими словами, планировщик должен поддерживать следующие виды операций:

1. $AddWorkerTypeInfo(WorkerType, WorkerAbilities)$: получить и обработать информацию о новом для планировщика типе работника, представленную в виде списка работ, которые он может выполнять ($WorkerAbilities$). Функция не добавляет нового работника, а лишь уточняет информацию о новом типе.

2. $AddWorker(Worker)$: обработать событие появления работника (гарантируется, что информация о типе работника уже была предоставлена планировщику).

3. $AddTaskTypeInfo(TaskType, TaskTypeInfo)$: получить и обработать информацию о новом типе задач.

4. $AddTask(Task, OnScheduledCallback)$: обработать событие появления новой задачи и вызвать переданную функцию $OnScheduledCallback$ в момент, когда задаче будет назначен работник (гарантируется, что информация о типе задачи уже была предоставлена планировщику).

## 1.3 Области применения алгоритмов о назначениях

Онлайн-алгоритмы о назначениях имеют применение во многих областях, в качестве примеров приведем следующие области:

– сервисы заказа такси: например, такие сервисы, как *Uber* и *Lyft*, используют онлайн-алгоритмы, чтобы в реальном времени рассчитывать стоимость поездки в зависимости от количества свободных машин и числа людей, желающих уехать. Когда поступает запрос на поездку, система должна назначить водителя и определить цену, учитывая текущее местоположение водителей, среднее время ожидания и прогнозируемый спрос;

– сопоставление водителей с заказами в службах доставки еды: платформы доставки еды, такие как *Delivery Club* и Яндекс.Еда, используют онлайн алгоритмы для назначения курьеров заказам. Алгоритмы стараются подобрать самого близкого и свободного курьера к ресторану, чтобы еда доехала быстро, а клиенты не ждали, при этом стараясь, чтобы у курьера не было лишнего простоя;

– распределение задач в облачных вычислениях: облачные провайдеры используют онлайн алгоритмы для распределения вычислительных ресурсов между различными задачами. Когда поступает новый запрос на вычисления,

система должна решить, на каком сервере его выполнить, учитывая доступные ресурсы, требования задачи и целевые показатели производительности;

– повышение эффективности логистики в цепях поставок: компании, занимающиеся логистикой, используют онлайн алгоритмы для управления транспортными потоками и распределения грузов. Как только появляется новый заказ на перевозку, система выбирает подходящую машину, прокладывает маршрут и проверяет ограничения по вместимости и срокам доставки;

– подбор персонала на краткосрочные проекты: платформы для поиска фрилансеров и вре́менных сотрудников могут использовать онлайн алгоритмы для сопоставления специалистов с краткосрочными проектами. Когда появляется новый проект, платформа подбирает специалистов с требуемыми навыками, учитывая их занятость и желаемую оплату труда;

– управление бронированием: онлайн-платформы для бронирования отелей или аренды жилья используют онлайн-алгоритмы для назначения комнат или квартир новым клиентам. Система в реальном времени принимает решения, какое из доступных мест лучше всего подходит для каждого бронирования, учитывая предпочтения гостя, сроки проживания, правила минимального/максимального пребывания и текущую загрузку;

– показ рекламы на сайтах: Когда пользователь заходит на сайт, рекламные платформы (такие как *Google Ad Manager* или Яндекс.Директ) определяют, какую рекламу показать этому человеку, основываясь на информации о предпочтениях пользователя, последних действиях и покупках в интернете, а также в зависимости от сайта.

## 1.4 Выводы

Таким образом, определён объект исследования в данной работе — это задача о назначениях в режиме реального времени. Кроме того, определён интерфейс онлайн-алгоритма, который должен будет решать поставленную задачу. К тому же, описаны области применения онлайн-алгоритмов о назначениях.

# ГЛАВА 2
# РЕАЛИЗАЦИИ ОНЛАЙН-АЛГОРИТМА

## 2.1 Первый вариант реализации онлайн-алгоритма

### 2.1.1 Описание алгоритма

Пусть дана последовательность событий, состоящая из появления задач и работников. И пусть также в момент появления каждого работника уже известна информация о его типе, то есть известно, какие задачи способен выполнять работник, и какова вероятность появления этих задач. В рамках данной реализации полагается, что вероятность появления задач неизменна со временем.

Рассматриваемый алгоритм строится на выборе в каждый момент самого полезного работника, где метрика полезности определяется как суммарная вероятность появления типов задач, способных быть выполненными данным работником. Для того, чтобы быстро определять, какой тип сотрудников выбрать на новом шаге, сохраняется отсортированный по суммарной вероятности появления типов задач контейнер типов работников, где ключом является тип работника, а значением – суммарная вероятность выполняемых им типов задач. Для этого использовалась структура *std::map*. При этом, информация о вероятностях появления задач хранится в хеш-таблице по ключу типа задачи. Чтобы определять, остались ли сотрудники соответствующего типа или же все ушли с работы, так как закончился их рабочий день, сохраняется для каждого типа список сотрудников, отсортированный по времени прихода на работу, из которого и берется сотрудник при выборе конкретного типа. Для этих нужд использовался контейнер *std::priority_ queue*, в котором на вершине кучи хранится сотрудник, прибывший на работу раньше всех. В момент появления новой задачи совершается проход по отсортированным типам до момента, пока не встречается тип, для которого имеются незанятые работники. В случае, если такого типа не нашлось, задача остается не назначенной, а алгоритм при этом продолжает работу. Если же такой тип нашелся, то выбранного работника следует переместить в список занятых работников, чтобы по выполнению выданной ему работы, снова вернуть его в список незанятых сотрудников. Для этих нужд также использовался контейнер *std::priority_ queue*.

### 2.1.2 Временная сложность

Здесь и далее вводятся обозначения: $n = |Workers|$ – общее число пришедших работников, $m = |Tasks|$ – общее число пришедших задач, $W$ – количество различных типов работников, $T = |Types|$ – количество различных типов задач.

Далее приведено перечисление основных методов алгоритма и оценок их временно́й сложности:

– метод *AddWorkerTypeInfo* работает за $O(T+log(W))$, так как требуется просуммировать вероятности типов задач за $O(T)$ и вставить информацию о новом типе в отсортированный контейнер, что работает по свойствам *std::map* за $O(log(W))$ [3].

– метод *AddWorker* работает за $O(log(n))$, так как требуется добавить нового работника в список работников такого же типа, хранящийся в куче, что происходит по свойствам *std::priority_queue* за $O(log(n))$.

– метод *AddtaskTypeInfo* работает за $O(1)$, так как требуется вставить информацию о новом типе в хеш-таблицу, что работает по свойствам *std::unordered_map* за $O(1)$ [3].

– метод *AddTask* работает за $O(W{\cdot}log(W){\cdot}log(n))$, так как в худшем случае требуется пройтись по всему набору отсортированных типов работников (что по свойствам *std::map* работает за $O(W{\cdot}log(W))$), удалив хотя бы один элемент из кучи (хранилища отсортированных по времени прибытия на работу сотрудников), что работает по свойствам *std::priority_queue* за $O(log(n))$.

### 2.1.3 Затраты памяти

Затраты памяти составляют $O(n+W{\cdot}T)$, так как требуется сохранить информацию обо всех незанятых работниках ($O(n)$), информацию о компетенциях всех типов работников ($O(W{\cdot}T)$) и информацию о «полезности»каждого типа работника ($O(W)$).

### 2.1.4 Работа алгоритма в условиях недостатка информации о задачах

В случае, если данные о вероятности появления определенного типа задач не были предоставлены, или если данные могут содержать ошибки, в алгоритм можно добавить логику пересчета вероятности появления типов задач, исходя из исторических данных. Для этого изначально полагается, что все типы задач равновероятны, а затем информация о вероятности появления некоторого типа обновляется по формуле $\frac{1+seenCount}{typesCount+totalTasksCount}$, где

$seenCount$ — это количество задач выбранного типа, уже обработанных алгоритмом, $typesCount$ обозначает количество различных типов задач, а $totalTasksCount$ — общее количество задач, уже обработанных алгоритмом. Теперь, чтобы поддерживать корректную информацию о полезности различных типов работников, требуется хранить счетчики числа обработанных типов задач для каждого типа задачи в хеш-таблице и, обновляя их при обработке новой задачи, обновлять также информацию о полезности всех типов работников.

Такой подход замедлит функцию добавления задачи, добавив в асимптотику слагаемое $O(W \cdot T)$, так как потребуется для каждого типа работников пересчитать его полезность, итоговая сложность: $O(W \cdot log(W) \cdot log(n) + W \cdot T)$. Асимптотики времени работы оставшихся методов и затрат памяти не изменятся.

## 2.2 Второй вариант реализации онлайн-алгоритма

### 2.2.1 Описание алгоритма

Пусть дана последовательность событий, состоящая из появления задач и работников. И пусть также в момент появления каждого работника уже известна информация о его типе, то есть известно, какие задачи способен выполнять работник, и какова вероятность появления этих задач. В рамках данной реализации полагается, что вероятность появления задач неизменна со временем.

Главным отличием данной реализации от предыдущей будет попытка учесть количество работников для соответствующего типа работника, чтобы избегать ситуации, когда были использованы все работники, соответствующие типу с наименьшей суммарной вероятностью появления задач, способных быть выполненными этим типом, из-за чего некоторые маловероятные задачи лишаются исполнителей. Для этого сортируются типы работников (для которых существуют незанятые работники) по суммарной вероятности появления задач этого типа, делённой на количество работников данного типа. То есть каждому типу работников будет поставлена в соответствие величина «полезности» $\frac{\sum_{j=1}^{T} p_j}{\sum_{i=1}^{W} I_{ij}}$, где $p_j$ – вероятность появления $j$-й задачи, а $I_{ij}$ – индикатор события «работник с типом $i$ может выполнить задачу типа $j$».

Рассматриваемый алгоритм строится на выборе в каждый момент времени самого полезного работника. Для того, чтобы определять, какой тип сотрудников выбрать на новом шаге, сохраняются полезности всех типов работников в хеш-таблице, где ключом является тип работника, а значением – суммарная вероятность выполняемых им типов задач. Для этого использова-

лась структура *std::unordered_map*. Кроме того, чтобы определять, остались ли сотрудники соответствующего типа или же все ушли с работы (так как закончился их рабочий день), для каждого типа задач хранится список сотрудников, отсортированный по времени прихода на работу, из которого и берётся сотрудник при выборе конкретного типа. Для этих нужд используется контейнер *std::unordered_map*, ключами которого являются типы сотрудников, а значениями — списки незанятых сотрудников в виде контейнера *std::priority_queue*, в котором на вершине кучи хранится сотрудник, прибывший раньше всех. Теперь, чтобы выбрать работника в момент появления новой задачи, требуется пройтись по всем типам работников, выбрав из них тип с наибольшей суммарной вероятностью появления выполняемых им задач, делённой на количество оставшихся работников. Для этого рассматриваются только типы, для которых имеются незанятые сотрудники. В случае, если такого типа не нашлось, задача остаётся не назначенной, а алгоритм при этом продолжает работу. Если же такой тип нашёлся, то выбранного работника следует переместить в список занятых работников, чтобы по выполнению выданной ему работы снова вернуть его в список незанятых сотрудников. Для этих нужд также используется контейнер *std::priority_queue*.

Методология работы в условиях отсутствия или недостоверности информации о вероятностях появления различных типов задач остается аналогичной предыдущему алгоритму.

### 2.2.2 Временная сложность

Далее приведено перечисление основных методов алгоритма и асимптотик их работы:

– метод *AddWorkerTypeInfo* работает за $O(T)$, так как требуется вычислить «полезность»работника (что совершается за $O(T)$, так как требуется просуммировать вероятности тех типов задач, которые он способен выполнять) и вставить информацию о новом типе в хеш-таблицу, что работает по свойствам *std::unordered_map* за $O(1)$ [3];

– метод *AddWorker* работает за $O(log(n))$, так как требуется положить нового работника в список работников такого же типа, хранящийся в куче, что происходит по свойствам *std::priority_queue* за $O(log(n))$. Список работников заданного типа хранится в хеш-таблице (*std::unordered_map*) с асимптотикой доступа $O(1)$, поэтому итоговая асимптотика – $O(log(n))$;

– метод *AddtaskTypeInfo* работает за $O(1)$, так как требуется вставить информацию о новом типе в хеш-таблицу, что работает по свойствам *std::unordered_map* за $O(1)$ [3];

– метод *AddTask* работает за $O(W \cdot log(n))$, так как в худшем случае требуется пройтись по всему набору типов работников (проход по

*std::unordered_map* работает за $O(W)$), удалив в худшем случае из каждой очереди (хранилища отсортированных по времени прибытия на работу сотрудников) один элемент, положив впоследствии большинство элементов, за исключением быть может одного, на место, что работает по свойствам *std::priority_queue* за $O(log(n))$.

### 2.2.3 Затраты памяти

Затраты памяти составляют $O(n+W \cdot T)$, так как требуется сохранить информацию обо всех незанятых работниках ($O(n)$), информацию о компетенциях всех типов работников ($O(W \cdot T)$) и информацию о «полезности»каждого типа работника ($O(W)$).

## 2.3 Третий вариант реализации онлайн-алгоритма

### 2.3.1 Описание алгоритма

В отличие от предыдущих реализаций, в рамках данной реализации полагается, что вероятность появления определенного типа задач переменна со временем, что у каждого типа задач есть пиковый час, когда задач данного типа приходит больше всего, а также существует час, когда задач данного типа приходит меньше всего. Кроме того, полагается, что пиковые и минимальные часы нагрузки одинаковы и не меняются изо дня в день, а отношение числа задач определенного типа в пиковый и минимальный час одинаково для любой пары дней.

Теперь для выбора очередного работника для выполнения новой задачи следует воспользоваться метрикой полезности, заданной для каждого работника (пришедшего на работу в момент времени $T_{in}$ и ушедшего с работы в момент $T_{out}$), как: $\sum_{taskType}^{workerAbilities} (\sum_{t=T_{in}}^{T_{in}} (Probability(taskType, t))$, где $workerAbilities$ обозначает набор компетенций сотрудника, то есть набор типов задач, способных быть решёнными данным сотрудником, $Probability(taskType, t)$ – вероятность появления задачи типа $taskType$ в момент времени $t$, вычисленная из предположения, что вероятность появления задач линейно возрастает от минимального часа к пиковому и линейно убывает от пикового часа к минимальному. Таким образом рассчитывается «полезность»сотрудников, исходя из их реального графика с учетом предположения о появлении задач разных типов в рамках этого графика. В рамках такой модели каждый раз выбирается сотрудник с минимальной полезностью.

Для того, чтобы быстро находить сотрудника с минимальной полезностью, способного выполнять пришедшую задачу, для каждого типа со-

трудников хранится отсортированный по полезностям список сотрудников данного типа в виде приоритетной очереди (использовался контейнер *std::priority_ queue*). Тогда, имея новую задачу некоторого типа, совершается проход по типам работников, способных выполнить данную задачу, и выбор из них сотрудника с минимальной полезностью. В случае, если такого типа не нашлось, задача остается не назначенной, а алгоритм при этом продолжает работу. Если же такой тип нашелся, то выбранного работника следует переместить в список занятых работников, чтобы по выполнению выданной ему работы снова вернуть его в список незанятых сотрудников. Для этих нужд использовался контейнер *std::priority_ queue*.

### 2.3.2 Временная сложность

Далее приведено перечисление основных методов алгоритма и асимптотик их работы:

– метод *AddWorkerTypeInfo* работает за $O(1)$, так как требуется вставить информацию о типе в хеш-таблицу, что работает по свойствам *std::unordered_ map* за $O(log(1))$ [3];

– метод *AddWorker* работает за $O(T{\cdot}DaySize{\cdot}log(n))$ (где *DaySize* – длительность в часах рабочего дня сотрудников), так как требуется вычислить «полезность»работника, просуммировав для каждого типа задач (которых всего $O(T)$) вероятности появления задач данного типа во все часы работы сотрудника (работает за $O(DaySize)$), впоследствие вставив эту информацию в приоритетную очередь, что работает за $O(log(n))$. В отличие от предыдущих алгоритмов, в данном алгоритме полезность вычисляется для каждого сотрудника в отдельности, что приводит к ухудшению асимптотик;

– метод *AddtaskTypeInfo* работает за $O(1)$, так как требуется вставить информацию о новом типе в хеш-таблицу, что работает по свойствам *std::unordered_ map* за $O(1)$ [3];

– метод *AddTask* работает за $O(W{\cdot}log(n))$, так как в худшем случае требуется пройтись по всему набору типов работников (проход по *std::unordered_ map* работает за $O(W)$), удалив в худшем случае из каждой очереди (хранилища отсортированных по полезности сотрудников) один элемент, положив впоследствии большинство элементов, за исключением быть может одного, на место, что работает по свойствам *std::priority_ queue* за $O(log(n))$.

### 2.3.3 Затраты памяти

Затраты памяти составляют $O(n + W{\cdot}T)$, так как требуется сохранить информацию обо всех незанятых работниках и их полезности ($O(n)$) и ин-

формацию о компетенциях всех типов работников ($O(W \cdot T)$).

### 2.3.4 Работа алгоритма в условиях недостатка информации о задачах

В случае, если данные о вероятности появления определенного типа задач не были предоставлены, или если данные могут содержать ошибки, в алгоритм можно добавить логику пересчета вероятности появления типов задач, исходя из исторических данных. Для этого изначально полагается, что типы задач не имеют пиковых и минимальных часов, когда экземпляров задач данного типа приходит больше или меньше соответственно, а затем, по прошествии одного дня, определяется, в какой час задач приходило больше всего, а в какой меньше и какова была разница показов в эти часы. Впоследствии эти данные используются для расчета полезности сотрудников, уточняя их по итогам каждого нового дня, беря среднее значение числа пришедших задач выбранного типа из всех минимальных часов за минимум и среднее значение числа пришедших задач выбранного типа из всех максимальных часов за максимум. Такой подход не замедлит асимптотики времени работы методов алгоритма, затраты памяти также не изменятся.

## 2.4 Четвертый вариант реализации онлайн-алгоритма

### 2.4.1 Описание алгоритма

Как и в предыдущей реализации, в рамках данной реализации полагается, что вероятность появления определенного типа задач переменна со временем. Предположения о распределении вероятности во времени остаются справедливыми. Отличием текущей реализации от предыдущей будет попытка учесть количество незанятых работников каждого типа.

Теперь для выбора очередного работника для выполнения новой задачи использовалась метрика полезности, заданная для каждого работника (пришедшего на работу в момент времени $T_{in}$ и ушедшего с работы в момент $T_{out}$), как: $(\sum_{taskType}^{workerAbilities}(\sum_{t=T_{in}}^{T_{in}}(Probability(taskType, t)))/workersCount$, где $workerAbilities$ — набор компетенций сотрудника, то есть набор типов задач, способных быть решёнными данным сотрудником, под величиной $Probability(taskType, t)$ понимается вероятность появления новой задачи типа
$taskType$ в момент времени $t$, а $workersCount$ представляет собой количество работников данного типа. Выбирать в такой модели следует сотрудника с минимальной полезностью.

Для того, чтобы быстро находить сотрудника с минимальной полезностью, способного выполнять пришедшую задачу, следует, как и ранее, для каждого типа сотрудников хранить отсортированный по полезностям, умноженным на количество работников (чтобы не пересчитывать каждый раз полезности при изменении количества сотрудников данного типа), список сотрудников данного типа в виде приоритетной очереди (для этого использовался контейнер *std::priority_ queue)*. Тогда, имея новую задачу некоторого типа, алгоритму потребуется пройтись по типам работников, способных выполнить данную задачу, и выбрать из них сотрудника с минимальной полезностью, делённой на количество сотрудников данного типа. В случае, если такого типа не нашлось, задача остаётся не назначенной, а алгоритм при этом продолжает работу. Если же такой тип нашёлся, то выбранного работника следует переместить в список занятых работников, чтобы по выполнению выданной ему работы снова вернуть его в список незанятых сотрудников. Для этих нужд использовался контейнер *std::priority_ queue*.

Работа алгоритма в условиях недостатка информации о задачах остается аналогичной предыдущей реализации.

### 2.4.2 Временная сложность

Далее приведено перечисление основных методов алгоритма и асимптотик их работы:

– метод *AddWorkerTypeInfo* работает за $O(1)$, так как требуется вставить информацию о типе в хеш-таблицу, что работает по свойствам *std::unordered_ map* за $O(log(1))$ [3];

– метод *AddWorker* работает за $O(T \cdot DaySize)$ (где *DaySize* – длительность в часах рабочего дня сотрудников), так как требуется вычислить «полезность»работника, просуммировав для каждого типа задач (которых всего $O(T)$) вероятности появления задач данного типа во все часы работы сотрудника (работает за $O(DaySize)$), впоследствие вставив эту информацию в хеш-таблицу, что работает по свойствам *std::unordered_ map* за $O(log(1))$. В отличие от предыдущих алгоритмов, в данном алгоритме полезность вычисляется для каждого сотрудника в отдельности, что приводит к ухудшению асимптотик;

– метод *AddtaskTypeInfo* работает за $O(1)$, так как требуется вставить информацию о новом типе в хеш-таблицу, что работает по свойствам *std::unordered_ map* за $O(1)$ [3];

– метод *AddTask* работает за $O(W \cdot log(n))$, так как в худшем случае требуется пройтись по всему набору типов работников (проход по *std::unordered_ map* работает за $O(W)$), удалив в худшем случае из каждой

очереди (хранилища отсортированных по полезности сотрудников) один элемент, положив впоследствии большинство элементов, за исключением быть может одного, на место, что работает по свойствам *std::priority_queue* за $O(log(n))$.

### 2.4.3 Затраты памяти

Затраты памяти составляют $O(n + W{\cdot}T)$, так как требуется сохранить информацию обо всех незанятых работниках и их полезности ($O(n)$) и информацию о компетенциях всех типов работников ($O(W{\cdot}T)$).

## 2.5 Выводы

В данной главе было рассмотрено четыре реализации онлайн-планировщика, основывающиеся на равномерном и неравномерном представлении о распределении задач во времени. Для каждой из реализаций была описана своя эвристика выбора сотрудника. Кроме того, были описаны затраты по времени и памяти описанных решений.

# ГЛАВА 3
# МОДЕЛИРОВАНИЕ РЕАЛЬНОЙ НАГРУЗКИ

## 3.1 Описание тестовых моделей

В данной главе будут описаны методы моделирования реальной пользовательской нагрузки на алгоритм в виде приходящих задач и работников. Для генерации тестов будут использованы два подхода: генерация задач с равномерной вероятностью появления их во времени и генерация задач с неравномерной вероятностью появления. Во втором случае моделирование будет происходить в рамках одной недели по часам, что означает, что и работники, и задачи будут приходить только в начале каждого часа, а время выполнения задач будет кратно одному часу. Особенности распределения вероятностей задач во времени будут описаны ниже.

Для генерации компетенций работников в тестах будет использован гиперпараметр $P$, обозначающий вероятность появления у некоторого типа работника способности к выполнению выбранной задачи. Этот параметр имеет смысл процента заполненности матрицы компетенций, получаемой, если поставить в соответствие каждому типу работника $j$ столбец, а каждому типу задачи $i$ строку, где на пересечении будет стоять 0 или 1 в зависимости от того, способен ли $j$ тип выполнять $i$ тип задач (0 – нет, 1 – да). Т.е. для заданного $P$ мы будем генерировать компетенции работников (несколько вариантов), а для каждого варианта таких компетенций будем генерировать тестовый сценарий (по заданному числу задач), по которому будем считать метрику среднего числа выполненных задач.

## 3.2 Генерация последовательности задач Венгерским алгоритмом

### 3.2.1 Венгерский алгоритм

Алгоритм Куна-Манкреса, также известный как венгерский алгоритм [4], [5], предназначен для решения задачи о назначениях. В данном алгоритме предполагается, что каждый работник $i$ может выполнять любую работу $j$ с определенной стоимостью $C_{ij}$. Для моделирования невозможности выполнить работу этим стоимостям будут присваиваться достаточно большие значения, чтобы при существовании корректного решения получить меньшую суммарную стоимость работ.

Алгоритм основан на построении совершенного паросочетания минимального веса в двудольном графе $G = (U, V, E)$, где $U$ — множество работников,

$V$ — множество работ, а $E$ — множество рёбер с весами $C_{ij}$.

Ключевую роль играют потенциалы вершин: $u_i$ для работников $i \in U$ и $v_j$ для работ $j \in V$. Ребро $(i, j)$ называется допустимым, если $C_{ij} = u_i + v_j$. Алгоритм итеративно увеличивает размер паросочетания, используя только допустимые ребра.

На первом шаге алгоритма происходит инициализация:

– устанавливаются потенциалы работников: $u_i = 0$ для всех $i \in U$;

– устанавливаются потенциалы работ: $v_j = \min_{i \in U} C_{ij}$ для всех $j \in V$;

– инициализируется паросочетание $M = \emptyset$.

На втором шаге алгоритма происходит поиск увеличивающего пути: Пока $|M| < n$:

– выбирается свободный работник $i \in U$ (не входящий в $M$);

– запускается поиск в ширину по допустимым ребрам, начиная с работника $i$, помечая посещенные вершины. Для этого используется очередь $Q$ и массив *visited*;

– если найден свободный узел работы $j \in V$ (не входящий в $M$), то найден увеличивающий путь. В этом случае реконструируется увеличивающий путь от $j$ до $i$ (обычно с помощью массива предков), расширяется паросочетание $M$ путем инвертирования ребер на найденном пути (добавляются ребра, которых не было в $M$, и удаляются ребра, которые были) и совершается переход к следующей итерации основного цикла (шаг 2);

– если увеличивающий путь не найден, то вычисляется $\Delta = \min\{C_{ij} - u_i - v_j\}$ для всех посещенных работников $i$ и непосещенных работ $j$. Затем обновляются потенциалы: $u_i = u_i + \Delta$ для всех посещенных работников $i$; $v_j = v_j - \Delta$ для всех работ $j$, достижимых по чередующейся цепи из допустимых ребер, начиная со свободного работника $i$ (это можно эффективно реализовать, используя массив предков, построенный во время поиска в ширину).

Результат: Паросочетание $M$ содержит оптимальное назначение. Стоимость назначения равна $\sum_{(i,j) \in M} C_{ij}$.

Классическая реализация алгоритма Куна-Манкреса имеет сложность $O(n^3)$.

### 3.2.2 Генерация тестов на основе оффлайн-алгоритма

В рамках данного подхода будет генерироваться последовательность задач с равномерной вероятностью появления их во времени.

Интерфейс тестового генератора должен быть таким, чтобы для заданного числа задач с заданными вероятностями появления каждой из задач генерировалась последовательность задач заданного размера, удовлетворяющая заданным вероятностям с небольшой погрешностью. Для этого сначала генерируется набор задач по заданным вероятностям, а затем предпринимается попытка сгенерировать для этого набора работников, способных выполнить имеющуюся последовательность. Для генерации набора работников, способных выполнить имеющуюся последовательность, используются идеи бинарного поиска по ответу, и последовательность работников генерируется до тех пор, пока не станет возможным выполнение поставленных задач. Поиск решения методом бинарного поиска по ответу здесь возможен, так как понятно, что если некоторое количество работников способно выполнить последовательность задач, то, добавив к ним произвольное количество работников, последовательность задач также будет выполнена. Поэтому для того, чтобы применить бинарный поиск по ответу, требуется, добавляя и убирая лишних работников из последовательности исполнителей задач, найти минимальное число работников, всё ещё способных выполнить данную последовательность задач. Определить, возможно ли выполнение поставленных задач, помогает вышеописанный венгерский алгоритм. Имея последовательность работников, из которой можно назначить биективно работников задачам, остаётся лишь убрать из неё незадействованных работников. Теперь, имея назначения сотрудников на задачи, остаётся назначить время прихода и ухода на работу сотрудников таким, чтобы оно покрывало приходящую задачу, но не позволяло выполнить хотя бы ещё одну задачу. Для этого работнику можно присвоить $T_{in}$ времени появления задачи, а $T_{out} = T_{in} + TaskTime$, где $TaskTime$ – время выполнения задачи.

### 3.2.3 Временная сложность

Временна́я сложность такого подхода к генерации составит $O((T{\cdot}m)^3{\cdot}log(T{\cdot}m))$, так как потребуется бинарным поиском найти последовательность работников, способных выполнить заданную последовательность задач, которая в худшем случае может иметь размер $O(T{\cdot}m)$ (за счёт большого числа незадействованных работников), применяя на каждой итерации бин-поиска Венгерский алгоритм за $O((T{\cdot}m)^3)$. Теперь из того факта, что итераций бинарного поиска будет $O(log(T{\cdot}m))$ (так как последовательность не может быть длиной больше $O(T{\cdot}m)$) и из того, что на каждой из них потребуется применить Венгерский алгоритм, получаем асимптотику времени работы $O((T{\cdot}m)^3{\cdot}log(T{\cdot}m))$.

Данная оценка делает невозможной генерацию больших тестов вышеописанным методом, поэтому ниже в данной работе будет рассмотрен ускоренный тестовый генератор.

## 3.3 Генерация последовательности задач за линейное от числа задач время

### 3.3.1 Описание алгоритма

Ниже описывается метод ускорения по сравнению с предыдущим подходом генерации последовательности задач с постоянными вероятностями появления их во времени.

Ранее генерация последовательности задач опиралась по сути на идею равновероятной генерации исполнителей для всего набора задач, для чего генерировалась их (исполнителей) последовательность. В рассматриваемом алгоритме генерация исполнителей задач принимается равновероятной только в рамках одного типа задач. Это означает, что, как и ранее, генерируется последовательность задач различных типов в соответствии с вероятностями этих типов, а затем для каждой задачи в отдельности генерируется исполнитель, выбирая равновероятно тип сотрудника из типов, способных выполнить текущую задачу. Это значит, что в результате генерации сразу получается выполнимая последовательность задач и отпадает необходимость бинарного поиска по ответу с запуском медленного Венгерского алгоритма. Рабочий день сотрудника в таком подходе определяется аналогично предыдущему алгоритму генерации тестов.

### 3.3.2 Временная сложность

Временна́я сложность такого подхода к генерации составит $O(T \cdot W + m)$, так как для начала потребуется сгенерировать для каждого типа задачи (которых всего $O(T)$) набор его исполнителей, которых может быть $O(W)$, а затем для каждой задачи из сгенерированной последовательности задач размера $O(m)$ выбрать исполнителя.

## 3.4 Генерация последовательности задач с неравномерной вероятностью появления во времени

### 3.4.1 Описание используемой модели появления задач

В рамках данных генераций последовательности задач генерируются следующим образом: полагается, что в рамках каждого дня у каждой задачи есть пиковый час, когда экземпляров этой задачи приходит больше всего,

и час минимальной нагрузки, когда экземпляров меньше всего. Кроме того, полагается, что пиковые часы каждой задачи одинаковы в рамках одной недели и характеризуются пиковым часовым коэффициентом, который показывает соотношение числа задач, пришедших в пиковый час, к числу задач, пришедших в час минимальной нагрузки. Как и сами пиковые часы, пиковые коэффициенты полагаются неизменными для всех дней недели и являются параметром генератора. Ко всему прочему, полагается, что у задачи есть пиковые дни, аналогично пиковым часам, характеризующиеся пиковыми коэффициентами, определяющими соотношение суммарного числа задач, пришедшего в пиковый день и в обычный.

В рамках такой модели пиковые дни являются отражением праздников, когда пользователи некоторой платформы, решающей задачу о назначениях, приходят чаще, чем обычно.

### 3.4.2 Алгоритм генерации задач с неравномерной вероятностью появления во времени

Теперь для того, чтобы по заданному набору типов задач, с определёнными для каждого типа задач параметрами распределения (минимальный час, пиковый час, пиковый часовой коэффициент, пиковые дни и пиковые дневные коэффициенты) построить распределение этих задач во времени, используется следующий алгоритм:

– распределение для каждой задачи строится отдельно, зная, исходя из вероятности появления задачи, количество её экземпляров, которые должны появиться за всё время;

– для каждого типа задач вычисляется массив почасовых коэффициентов, отражающий отношение числа показов для любой выбранной пары часов (как отношение соответствующих коэффициентов этих часов). Для этого каждому часу ставится в соответствие коэффициент:

  – 1, если час не является пиковым и соответствующий день также не является пиковым;

  – «пиковый часовой коэффициент», если час является пиковым, но соответствующий день не является пиковым;

  – «пиковый дневной коэффициент», если час не является пиковым, но соответствующий день является пиковым;

  – «пиковый часовой коэффициент»· «пиковый дневной коэффициент», если час является пиковым и соответствующий день также является пиковым.

– построив массив почасовых коэффициентов, строится массив процента числа задач, появившихся в заданный час, как коэффициент данного часа, делённый на сумму почасовых коэффициентов. При этом для решения проблемы округления число задач в заданный час вычисляется через суммарное число задач, пришедших к заданному часу (что есть сумма процента числа задач для часов, не превосходящих заданный) за вычетом суммарного числа задач, пришедших к предыдущему часу. Так избегается ситуация, когда небольшое число задач не распределилось полностью по часам из-за небольших коэффициентов в некоторых часах и погрешностей округления.

Вышеописанный алгоритм работает за время $O(PeriodSize \cdot T + m)$, где $PeriodSize$ означает длительность периода планирования в часах, так как для каждого типа задачи строится массив почасовых коэффициентов за $O(PeriodSize)$, а затем генерируется $O(m)$ экземпляров задач.

### 3.4.3 Алгоритм генерации исполнителей задач

Теперь возникает задача найма работников для заданной последовательности задач. Требуется сгенерировать последовательность работников, способных выполнить последовательность задач, с соблюдением рабочих графиков сотрудников и, что самое важное, максимально эффективно тратя время нанятых работников, чтобы сгенерированная последовательность далее, поданная в качестве теста, в алгоритм, максимально приближала идеальное решение, не давая алгоритму лишних сотрудников, что могло бы улучшить его (алгоритма) метрики качества. Для определения качества сгенерированных тестов здесь и далее вводится параметр $PlanOptimality$, означающий отношение числа задач, выполненных последовательностью работников, к числу задач, теоретически способных быть выполненными заданным набором сотрудников (то есть к числу задач, которое сотрудник теоретически способен выполнить в рабочий день, умноженному на число сотрудников).

Для генерации используется следующий алгоритм:

– заводится массив счетчиков для каждого типа задач по каждому часу, где хранится число задач, пришедших в этот час и ещё не имеющих исполнителя;

– перебираются в порядке возрастания все часы, для которых существуют задачи без исполнителя, генерируя при этом исполнителя для каждой из этих задач (равновероятно для каждого типа задач из списка работников, способных выполнять данный тип задач), пока задач без исполнителя в данном часу не останется (то есть пока счетчик данного часа не занулится);

– имея сгенерированный набор новых исполнителей, назначаются времена выхода каждого из них на работу либо текущим часом, либо предыдущим

и далее в оставшееся свободное от выполнения уже назначенной задачи время, каждому исполнителю производится попытка дать ещё некоторое количество задач, найденных проходом по массиву счетчиков числа невыполненных задач следующих часов. При этом ставится цель назначать ровно столько задач, сколько способен выполнить сотрудник за рабочий день, при условии наличия задач соответствующих типов.

На практике, при генерации реальных последовательностей задач, оказалось, что точность такого подхода ($PlanOptimality$) составляет на всех сгенерированных в дальнейшем тестах не менее 98%, то есть генерируемые последовательности работников назначают работников не сильно хуже оптимального решения, неэффективно использовав лишь 2% от общего рабочего времени.

Вышеописанный алгоритм работает за время $O(m \cdot DaySize)$, где $DaySize$ означает длительность в часах рабочего дня сотрудников, так как для каждого типа задачи генерируется исполнитель, для которого впоследствии перебором оставшихся часов рабочего дня находятся, возможно, ещё некоторые задачи.

## 3.5 Выводы

В данной главе были описаны алгоритмы генерации тестовых данных. Для случая с равномерным распределением задач во времени было описано два алгоритма генерации тестовых сценариев, а для случая неравномерного распределения была описана математическая модель появления задач во времени и алгоритм, её реализующий, а также алгоритм, назначающий работников в соответствии с заданным распределением. Кроме того, были описаны затраты по времени описанных решений.

# ГЛАВА 4
# ТЕСТИРОВАНИЕ РАЗРАБОТАННЫХ АЛГОРИТМОВ

## 4.1 Тестирование корректности работы разработанных алгоритмов

Тестирование корректности работы реализованных алгоритмов проводилось с помощью библиотеки *gtest* и адресного санитайзера для *C++* [6]. Типы тестов описаны ниже. Для автоматической генерации тестовых сценариев использовались генераторы, описанные в главе 3 – о моделировании реальной нагрузки. Во всех нижеописанных тестах присутствует часть сценариев, написанных вручную, и часть, сгенерированных рандомизированно.

Для тестирования корректности работы алгоритма использовались следующие тестовые сценарии:

– тесты соответствия навыков: данные тесты определяют корректность работы алгоритма по соответствию навыков назначаемых сотрудников и типу задачи. Не допускается назначать сотрудника на задачу, для которой у него недостаточно компетенций;

– тесты незанятости работников: проверяют в момент назначения исполнителя задачи, не является ли назначенный сотрудник исполнителем ещё некоторой задачи в данный момент. Для этого сотрудникам присваиваются уникальные идентификаторы, для которых поддерживается список занятых работников. В момент, когда возникает необходимость проверить занятость работника, достаточно лишь выявить наличие идентификатора назначенного работника в списке занятых работников. Для хранения такого списка использовалась структура *std::unordered_map*, позволяющая быстро ($O(1)$) искать идентификаторы;

– тесты соблюдения рабочих графиков: данные тесты определяют корректность работы алгоритма по сопоставлению времени назначения сотрудника на задачу и часов работы сотрудника (не допускается назначать задачу сотруднику ещё не пришедшему на работу или уже ушедшего с работы). Кроме того, проверяется, что во все часы, необходимые для выполнения задачи, сотрудник находится на работе;

– простейшие тесты логики: проверяют, что в случае биективного соответствия типов сотрудников типам задач, алгоритм распределяет 100% пришедших задач.

## 4.2 Вычислительный эксперимент

### 4.2.1 Тесты на последовательности задач с равномерной вероятностью появления

Здесь и далее использованы тесты для периода планирования, размером в одну неделю, с длительностью рабочего дня - 8 часов и временем выполнения одной задачи - 3 часа. Для тестирования использовалась модель с 20 различными типами сотрудников и 20 различными типами задач.

В приведенных ниже тестах проверялось качество алгоритмов на простой модели, в рамках которой задачи обладают неизменной вероятностью появления со временем. Здесь и далее представлены таблицы с результатами точности алгоритмов (то есть процентом назначенных задач) в зависимости от вероятности появления навыка ($P$) и от реально сгенерированных компетенций сотрудников (параметр *Setup*).

Ниже в таблицах 4.1 и 4.2 приведены результаты тестов для достаточно больших $P$, из которых видно, что алгоритмы хорошо справляются с назначением работников при больших значениях $P$ (при увеличении этого параметра метрики только улучшаются), что связано с большой взаимозаменяемостью сотрудников.

Таблица 4.1 — Сравнительная точность алгоритмов при $P$=60%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99740 | 0.98569 | 0.99695 | 0.98660 |
| Setup-2 | 0.98510 | 0.96899 | 0.97895 | 0.97343 |
| Setup-3 | 0.99707 | 0.98771 | 0.99985 | 0.98603 |
| Setup-4 | 0.99725 | 0.98264 | 0.99984 | 0.97722 |
| Setup-5 | 0.99766 | 0.98810 | 0.99981 | 0.98240 |
| Setup-6 | 0.99658 | 0.98414 | 0.99320 | 0.98016 |
| Setup-7 | 0.89079 | 0.97154 | 0.97944 | 0.98468 |
| Setup-8 | 0.99726 | 0.98201 | 0.99980 | 0.97872 |
| Setup-9 | 0.99757 | 0.98951 | 0.99990 | 0.98314 |
| Setup-10 | 0.99743 | 0.98907 | 0.99981 | 0.98233 |

Таблица 4.2 — Сравнительная точность алгоритмов при $P$=50%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99731 | 0.98853 | 0.99282 | 0.98057 |
| Setup-2 | 0.99802 | 0.96983 | 0.98936 | 0.97838 |
| Setup-3 | 0.99716 | 0.97910 | 0.99833 | 0.97294 |
| Setup-4 | 0.99723 | 0.98924 | 0.99983 | 0.97564 |
| Setup-5 | 0.99770 | 0.98132 | 0.99982 | 0.98321 |
| Setup-6 | 0.98693 | 0.99085 | 0.99477 | 0.98166 |
| Setup-7 | 0.87843 | 0.97974 | 0.97571 | 0.96721 |
| Setup-8 | 0.99252 | 0.99143 | 0.99976 | 0.97892 |
| Setup-9 | 0.99738 | 0.98212 | 0.99975 | 0.97332 |
| Setup-10 | 0.99769 | 0.96168 | 0.97991 | 0.98555 |

Ниже в таблицах 4.3 и 4.4 приведены результаты тестов для $P$=40% и $P$=30%, из которых видно, что алгоритмы хуже справляются с назначением работников, так как работники становятся менее универсальными (чем при больших $P$) и возникает больше ошибок с определением подходящей им задачи. При этом первый алгоритм начинает в среднем показывать результаты хуже, чем остальные алгоритмы.

Таблица 4.3 — Сравнительная точность алгоритмов при $P$=40%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.95584 | 0.98979 | 0.98782 | 0.97572 |
| Setup-2 | 0.88844 | 0.96076 | 0.97986 | 0.98494 |
| Setup-3 | 0.89041 | 0.97512 | 0.98011 | 0.97331 |
| Setup-4 | 0.93067 | 0.97787 | 0.98075 | 0.96158 |
| Setup-5 | 0.93908 | 0.97533 | 0.99005 | 0.97342 |
| Setup-6 | 0.98520 | 0.97605 | 0.99132 | 0.96997 |
| Setup-7 | 0.99744 | 0.96489 | 0.99188 | 0.97646 |
| Setup-8 | 0.99759 | 0.99185 | 0.99976 | 0.97421 |
| Setup-9 | 0.99758 | 0.97785 | 0.99978 | 0.97280 |
| Setup-10 | 0.99141 | 0.97068 | 0.99817 | 0.97387 |

Таблица 4.4 — Сравнительная точность алгоритмов при $P$=30%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.94955 | 0.97321 | 0.99211 | 0.96622 |
| Setup-2 | 0.90368 | 0.97081 | 0.97462 | 0.97303 |
| Setup-3 | 0.94484 | 0.97247 | 0.98127 | 0.97008 |
| Setup-4 | 0.99721 | 0.97438 | 0.99464 | 0.96901 |
| Setup-5 | 0.99835 | 0.97273 | 0.98522 | 0.96134 |
| Setup-6 | 0.87127 | 0.96762 | 0.98030 | 0.97447 |
| Setup-7 | 0.91110 | 0.93983 | 0.97918 | 0.97956 |
| Setup-8 | 0.96754 | 0.97200 | 0.98039 | 0.96339 |
| Setup-9 | 0.86206 | 0.95982 | 0.96851 | 0.96484 |
| Setup-10 | 0.88797 | 0.96643 | 0.98029 | 0.96209 |

Ниже в таблицах 4.5 и 4.6 приведены результаты для $P$=20% и $P$=10%, из которых видно, что алгоритмы хорошо справляются с назначением задач при достаточно малых параметрах $P$ (при уменьшении этого параметра меньше 10% метрики продолжат улучшаться), что связано с тем, что при малом количестве навыков значительно сужается неопределенность в момент назначения, так как лишь небольшое количество работников способно выполнить появившийся тип задач.

Таблица 4.5 — Сравнительная точность алгоритмов при $P$=20%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99794 | 0.97322 | 0.99972 | 0.95933 |
| Setup-2 | 0.99036 | 0.98467 | 0.99296 | 0.96197 |
| Setup-3 | 0.99856 | 0.96835 | 0.99963 | 0.96095 |
| Setup-4 | 0.99871 | 0.97320 | 0.99839 | 0.97464 |
| Setup-5 | 0.89719 | 0.98013 | 0.96936 | 0.96423 |
| Setup-6 | 0.99798 | 0.98261 | 0.99976 | 0.97696 |
| Setup-7 | 0.94341 | 0.95386 | 0.96570 | 0.96599 |
| Setup-8 | 0.92120 | 0.94976 | 0.98211 | 0.96804 |
| Setup-9 | 0.98390 | 0.97920 | 0.98101 | 0.96804 |
| Setup-10 | 0.97368 | 0.97064 | 0.99535 | 0.96967 |

Таблица 4.6 — Сравнительная точность алгоритмов при $P$=10%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 1.00000 | 0.99203 | 0.99994 | 0.98712 |
| Setup-2 | 0.83976 | 0.97350 | 0.98420 | 0.98158 |
| Setup-3 | 0.99857 | 0.96924 | 0.99971 | 0.96253 |
| Setup-4 | 0.99950 | 1.00000 | 1.00000 | 1.00000 |
| Setup-5 | 0.99929 | 0.99471 | 0.99994 | 0.98833 |
| Setup-6 | 0.99884 | 0.99671 | 0.99996 | 0.98848 |
| Setup-7 | 0.91095 | 0.97518 | 0.98436 | 0.96180 |
| Setup-8 | 0.99930 | 0.99600 | 0.99993 | 0.99097 |
| Setup-9 | 0.99998 | 1.00000 | 1.00000 | 1.00000 |
| Setup-10 | 0.99955 | 0.97959 | 0.99992 | 0.99991 |

### 4.2.2 Тесты на последовательности задач с неравномерной вероятностью появления

В первом сценарии распределения задач (изображенном на рисунке 4.1) во времени мы смоделируем последовательности из типов задач, обладающих близкими пиковыми и минимальными часами и не имеющих пиковых дней.

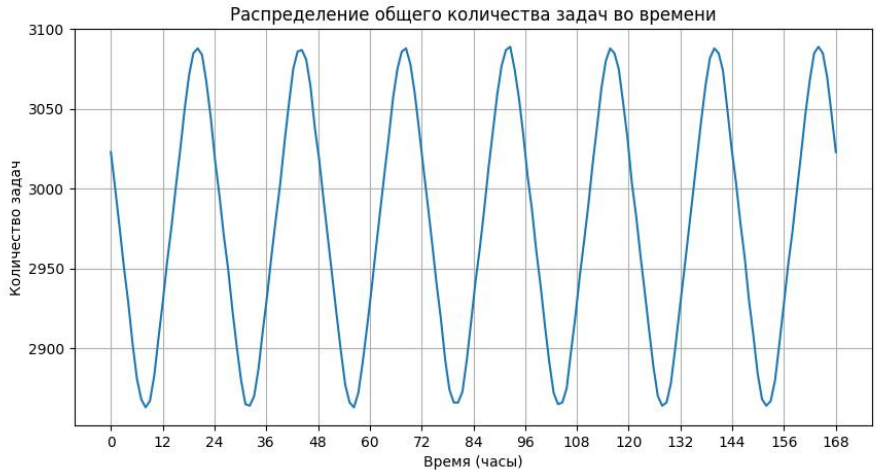


Рисунок 4.1 — Первый сценарий распределения задач во времени

Ниже в таблицах 4.7 и 4.8 приведены результаты тестирования алгоритмов при $P$=60% и $P$=50%, из которых видно, что алгоритмы, как и ранее, хорошо справляются с назначением работников при больших значениях $P$, что связано с большой взаимозаменяемостью сотрудников.

Таблица 4.7 — Сравнительная точность алгоритмов при $P$=60%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99696 | 0.98401 | 0.99169 | 0.97521 |
| Setup-2 | 0.98959 | 0.99032 | 0.99153 | 0.98075 |
| Setup-3 | 0.99730 | 0.99110 | 0.99162 | 0.97876 |
| Setup-4 | 0.99708 | 0.99176 | 0.99420 | 0.98145 |
| Setup-5 | 0.96629 | 0.98732 | 0.98586 | 0.98074 |
| Setup-6 | 0.96243 | 0.98379 | 0.98851 | 0.97709 |
| Setup-7 | 0.99730 | 0.98050 | 0.98931 | 0.97341 |
| Setup-8 | 0.99751 | 0.99657 | 0.99726 | 0.98632 |
| Setup-9 | 0.96448 | 0.97373 | 0.98605 | 0.97090 |
| Setup-10 | 0.99621 | 0.98684 | 0.99259 | 0.98104 |

Таблица 4.8 — Сравнительная точность алгоритмов при $P$=50%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98124 | 0.98709 | 0.98971 | 0.97712 |
| Setup-2 | 0.98882 | 0.98364 | 0.99202 | 0.97777 |
| Setup-3 | 0.98703 | 0.97866 | 0.98798 | 0.97131 |
| Setup-4 | 0.98741 | 0.98788 | 0.99183 | 0.97613 |
| Setup-5 | 0.98283 | 0.99112 | 0.99164 | 0.97896 |
| Setup-6 | 0.98495 | 0.98029 | 0.98620 | 0.97328 |
| Setup-7 | 0.98715 | 0.98403 | 0.98676 | 0.97936 |
| Setup-8 | 0.98676 | 0.98620 | 0.99184 | 0.98015 |
| Setup-9 | 0.98471 | 0.98708 | 0.99031 | 0.97854 |
| Setup-10 | 0.96818 | 0.99107 | 0.99249 | 0.97949 |

Ниже в таблицах 4.9 и 4.10 приведены результаты тестирования алгоритмов при $P$=40% и $P$=30%, из которых видно, что алгоритмы хуже распределяют задачи при меньших $P$, при этом в среднем третий и четвертый алгоритмы показывают себя лучше в силу более точной математической модели появления задач (она лучше соответствует тестам).

Таблица 4.9 — Сравнительная точность алгоритмов при $P$=40%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98696 | 0.97960 | 0.99374 | 0.98206 |
| Setup-2 | 0.98615 | 0.98295 | 0.98566 | 0.96908 |
| Setup-3 | 0.98738 | 0.97871 | 0.98668 | 0.97167 |
| Setup-4 | 0.97417 | 0.97676 | 0.98708 | 0.96949 |
| Setup-5 | 0.98722 | 0.98028 | 0.99177 | 0.96696 |
| Setup-6 | 0.98613 | 0.97571 | 0.98358 | 0.96749 |
| Setup-7 | 0.98746 | 0.98971 | 0.99216 | 0.97757 |
| Setup-8 | 0.98700 | 0.96883 | 0.98461 | 0.96876 |
| Setup-9 | 0.98748 | 0.98443 | 0.98860 | 0.96932 |
| Setup-10 | 0.98737 | 0.98493 | 0.98785 | 0.97429 |

Таблица 4.10 — Сравнительная точность алгоритмов при $P$=30%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.94711 | 0.98231 | 0.98711 | 0.97565 |
| Setup-2 | 0.90124 | 0.97065 | 0.97874 | 0.97539 |
| Setup-3 | 0.94926 | 0.96200 | 0.97472 | 0.96708 |
| Setup-4 | 0.91554 | 0.96003 | 0.97860 | 0.97415 |
| Setup-5 | 0.96380 | 0.98449 | 0.98802 | 0.97624 |
| Setup-6 | 0.98750 | 0.96951 | 0.98471 | 0.96643 |
| Setup-7 | 0.93304 | 0.98293 | 0.98793 | 0.97015 |
| Setup-8 | 0.98687 | 0.96520 | 0.98831 | 0.96552 |
| Setup-9 | 0.98323 | 0.97483 | 0.98345 | 0.97117 |
| Setup-10 | 0.98708 | 0.97704 | 0.98087 | 0.96872 |

Ниже в таблицах 4.11 и 4.12 приведены результаты тестирования алгоритмов при $P$=20% и $P$=10%, из которых видно, что алгоритмы, как и ранее в случае равномерной вероятности, хорошо справляются с распределением задач при малых $P$, несмотря на изменившийся тип теста.

Таблица 4.11 — Сравнительная точность алгоритмов при $P$=20%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.92030 | 0.97573 | 0.98169 | 0.96535 |
| Setup-2 | 0.93030 | 0.97135 | 0.98973 | 0.96561 |
| Setup-3 | 0.96576 | 0.96295 | 0.98480 | 0.96609 |
| Setup-4 | 0.99333 | 0.94489 | 0.99056 | 0.96163 |
| Setup-5 | 0.99633 | 0.98055 | 0.99279 | 0.98015 |
| Setup-6 | 0.93347 | 0.89502 | 0.95921 | 0.96398 |
| Setup-7 | 0.89437 | 0.96309 | 0.96845 | 0.96870 |
| Setup-8 | 0.99858 | 0.97815 | 0.99336 | 0.97173 |
| Setup-9 | 0.99796 | 0.97805 | 0.98978 | 0.96701 |
| Setup-10 | 0.99831 | 0.98688 | 0.99252 | 0.97712 |

Таблица 4.12 — Сравнительная точность алгоритмов при $P$=10%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99746 | 0.93888 | 0.97951 | 0.95489 |
| Setup-2 | 0.97080 | 0.97887 | 0.98045 | 0.97465 |
| Setup-3 | 0.99366 | 0.97730 | 0.98223 | 0.97173 |
| Setup-4 | 0.99909 | 0.97983 | 0.99525 | 0.97841 |
| Setup-5 | 0.99743 | 0.96282 | 0.98534 | 0.96597 |
| Setup-6 | 0.96667 | 0.97472 | 0.98311 | 0.97227 |
| Setup-7 | 0.87356 | 0.97202 | 0.98634 | 0.96493 |
| Setup-8 | 0.90405 | 0.98737 | 0.98788 | 0.97516 |
| Setup-9 | 0.93359 | 0.96117 | 0.97985 | 0.96946 |
| Setup-10 | 0.99822 | 0.97680 | 0.98684 | 0.98317 |

Из приведённых результатов можно сделать вывод, что алгоритмы хорошо справляются с назначением задач при больших ($\geq 50\%$) и достаточно малых ($\leq 20\%$) параметрах $P$, а в среднем лучше себя ведёт третья реализация.

Второй сценарий распределения задач во времени (изображенный на рисунке 4.2) будет моделировать наличие у типов задач пиковых дней (при выборе первого и второго дней недели, как пиковых для всех типов задач), при этом оставив близкими у разных типов задач пиковые и минимальные часы.

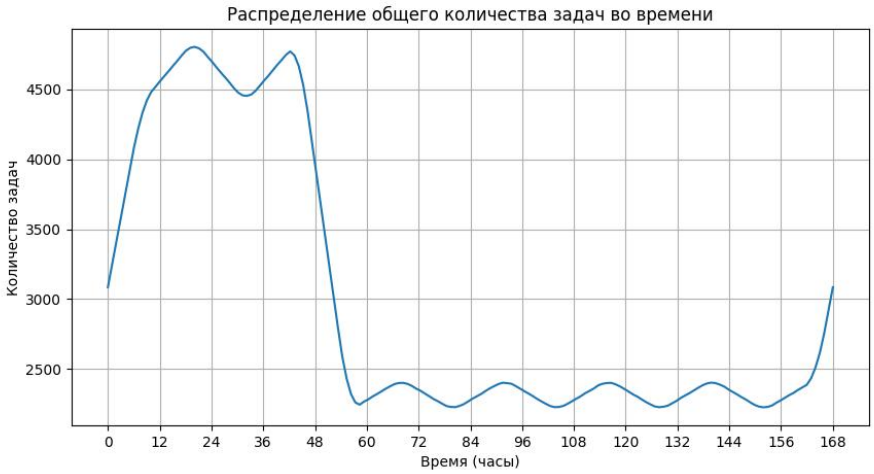


Рисунок 4.2 — Второй сценарий распределения задач во времени

Ниже в таблицах 4.13 и 4.14 приведены результаты тестирования алгоритмов при $P$=60% и $P$=50%, из которых видно, что третий и четвертый алгоритмы достаточно хорошо справляются с назначением задач по сравнению с первым и вторым, так как последние обладают менее точной математической моделью появления задач. Однако по сравнению с первым тестовым сценарием метрики ухудшаются, так как в математической модели алгоритмов отсутствует информация о пиковых днях.

Таблица 4.13 — Сравнительная точность алгоритмов при $P$=60%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98711 | 0.99193 | 0.99371 | 0.98233 |
| Setup-2 | 0.89497 | 0.97391 | 0.98181 | 0.97236 |
| Setup-3 | 0.99704 | 0.97319 | 0.98730 | 0.97146 |
| Setup-4 | 0.99721 | 0.98427 | 0.98941 | 0.97836 |
| Setup-5 | 0.93536 | 0.98529 | 0.98473 | 0.97879 |
| Setup-6 | 0.99722 | 0.99351 | 0.99359 | 0.98335 |
| Setup-7 | 0.98742 | 0.97986 | 0.99063 | 0.97198 |
| Setup-8 | 0.98722 | 0.98515 | 0.98896 | 0.97147 |
| Setup-9 | 0.99731 | 0.98560 | 0.98959 | 0.97498 |
| Setup-10 | 0.99719 | 0.97582 | 0.98668 | 0.97560 |

Таблица 4.14 — Сравнительная точность алгоритмов при $P$=50%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.93975 | 0.98540 | 0.98697 | 0.97354 |
| Setup-2 | 0.99716 | 0.99156 | 0.99401 | 0.97911 |
| Setup-3 | 0.98478 | 0.96427 | 0.97870 | 0.96785 |
| Setup-4 | 0.99708 | 0.96689 | 0.97751 | 0.97177 |
| Setup-5 | 0.93043 | 0.96422 | 0.97955 | 0.97193 |
| Setup-6 | 0.97330 | 0.97696 | 0.98071 | 0.96875 |
| Setup-7 | 0.98695 | 0.97979 | 0.98833 | 0.97605 |
| Setup-8 | 0.98450 | 0.98467 | 0.98928 | 0.97797 |
| Setup-9 | 0.98723 | 0.98234 | 0.98651 | 0.96648 |
| Setup-10 | 0.98435 | 0.98539 | 0.98570 | 0.97523 |

Ниже в таблицах 4.15 и 4.16 приведены результаты тестирования алгоритмов при $P$=40% и $P$=30%, из которых видно, что хоть метрики всех алгоритмов ухудшились (как и ранее, это связано с меньшей взаимозаменяемостью сотрудников), но все же третий и четвертый алгоритмы более стабильно с точки зрения качества назначают сотрудников на задачи, что по-прежнему связано с более точной математической моделью.

Таблица 4.15 — Сравнительная точность алгоритмов при $P$=40%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98957 | 0.97867 | 0.98130 | 0.97040 |
| Setup-2 | 0.90239 | 0.97470 | 0.98112 | 0.96604 |
| Setup-3 | 0.99752 | 0.98526 | 0.99262 | 0.97703 |
| Setup-4 | 0.96228 | 0.99263 | 0.99195 | 0.97605 |
| Setup-5 | 0.90540 | 0.98137 | 0.97851 | 0.97393 |
| Setup-6 | 0.98722 | 0.99315 | 0.99550 | 0.98346 |
| Setup-7 | 0.98703 | 0.98832 | 0.98778 | 0.97435 |
| Setup-8 | 0.85327 | 0.96721 | 0.98012 | 0.97123 |
| Setup-9 | 0.98221 | 0.97452 | 0.98168 | 0.97328 |
| Setup-10 | 0.91661 | 0.96938 | 0.98388 | 0.97357 |

Таблица 4.16 — Сравнительная точность алгоритмов при $P$=30%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98775 | 0.96548 | 0.97843 | 0.96781 |
| Setup-2 | 0.88110 | 0.96545 | 0.97428 | 0.97211 |
| Setup-3 | 0.90641 | 0.97320 | 0.98274 | 0.97031 |
| Setup-4 | 0.99708 | 0.96823 | 0.98414 | 0.96722 |
| Setup-5 | 0.89987 | 0.96751 | 0.97120 | 0.96801 |
| Setup-6 | 0.99677 | 0.97648 | 0.98801 | 0.95994 |
| Setup-7 | 0.95299 | 0.98146 | 0.98714 | 0.97653 |
| Setup-8 | 0.99643 | 0.98136 | 0.98967 | 0.97027 |
| Setup-9 | 0.91869 | 0.97235 | 0.97966 | 0.96800 |
| Setup-10 | 0.95888 | 0.95411 | 0.98316 | 0.96377 |

Ниже в таблицах 4.17 и 4.18 приведены результаты тестирования алгоритмов при $P$=20% и $P$=10%, из которых видно, что при уменьшившейся неопределенности назначения сотрудников (так как лишь немногие из них подходят для новоприходящих задач) снова начинают хорошо себя показывать простые (первый и второй) алгоритмы. При этом метрики всех алгоритмов улучшаются (при дальнейшем уменьшении параметра $P$, метрики продолжат улучшаться).

Таблица 4.17 — Сравнительная точность алгоритмов при $P$=20%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.95872 | 0.98148 | 0.98348 | 0.96468 |
| Setup-2 | 0.91305 | 0.97720 | 0.98346 | 0.97589 |
| Setup-3 | 0.99750 | 0.98037 | 0.99292 | 0.97715 |
| Setup-4 | 0.99673 | 0.92686 | 0.97238 | 0.96690 |
| Setup-5 | 0.92274 | 0.96984 | 0.96842 | 0.96852 |
| Setup-6 | 0.87225 | 0.97597 | 0.98233 | 0.97812 |
| Setup-7 | 0.99847 | 0.98849 | 0.99693 | 0.98775 |
| Setup-8 | 0.98198 | 0.97443 | 0.98675 | 0.96561 |
| Setup-9 | 0.99764 | 0.98960 | 0.99680 | 0.97581 |
| Setup-10 | 0.92306 | 0.98995 | 0.98314 | 0.97867 |

Таблица 4.18 — Сравнительная точность алгоритмов при $P$=10%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.94945 | 0.98614 | 0.98937 | 0.98498 |
| Setup-2 | 0.99962 | 0.97574 | 0.99485 | 0.95912 |
| Setup-3 | 0.93811 | 0.98152 | 0.98893 | 0.96491 |
| Setup-4 | 0.99861 | 0.97105 | 0.99232 | 0.97904 |
| Setup-5 | 0.99927 | 1.00000 | 1.00000 | 1.00000 |
| Setup-6 | 0.99865 | 0.99755 | 0.99751 | 0.99374 |
| Setup-7 | 0.94869 | 0.97624 | 0.98179 | 0.97086 |
| Setup-8 | 0.99970 | 0.96258 | 0.99496 | 0.97902 |
| Setup-9 | 0.99988 | 0.98506 | 0.99731 | 0.98991 |
| Setup-10 | 0.93125 | 0.97559 | 0.99289 | 0.97676 |

Третий сценарий распределения задач во времени (изображенный на рисунке 4.3) будет моделировать наличие у типов задач пиковых дней с различными пиковыми коэффициентами для каждого типа задач, при этом оставив близкими у разных типов задач пиковые и минимальные часы.

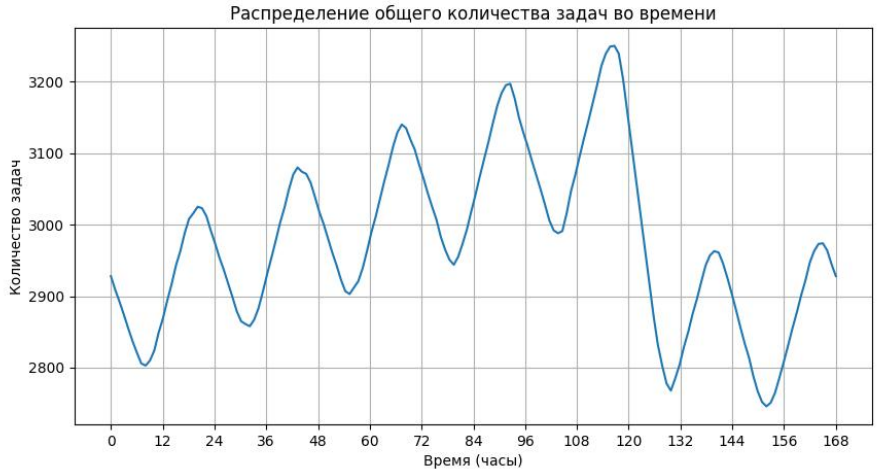


Рисунок 4.3 — Третий сценарий распределения задач во времени

Ниже в таблицах 4.19 и 4.20 приведены результаты тестирования алгоритмов при $P$=60% и $P$=50%, из которых видно, что алгоритмы достаточно хорошо себя показывают при больших значениях $P$, однако большое количество пиковых дней ухудшает метрики всех алгоритмов по сравнению со вторым сценарием, в силу отсутствия в моделях знаний о пиковых днях.

Таблица 4.19 — Сравнительная точность алгоритмов при $P$=60%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98701 | 0.99376 | 0.99484 | 0.98276 |
| Setup-2 | 0.98725 | 0.99138 | 0.99566 | 0.98228 |
| Setup-3 | 0.99697 | 0.97934 | 0.99356 | 0.97354 |
| Setup-4 | 0.98696 | 0.98514 | 0.99010 | 0.97864 |
| Setup-5 | 0.98654 | 0.98304 | 0.98859 | 0.97374 |
| Setup-6 | 0.98729 | 0.99239 | 0.99085 | 0.98348 |
| Setup-7 | 0.99723 | 0.99631 | 0.99527 | 0.98821 |
| Setup-8 | 0.99728 | 0.98770 | 0.98962 | 0.98083 |
| Setup-9 | 0.98719 | 0.98261 | 0.98938 | 0.97625 |
| Setup-10 | 0.99308 | 0.98150 | 0.98468 | 0.97875 |

Таблица 4.20 — Сравнительная точность алгоритмов при $P$=50%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98686 | 0.98262 | 0.99079 | 0.97531 |
| Setup-2 | 0.98592 | 0.98558 | 0.98637 | 0.97131 |
| Setup-3 | 0.98738 | 0.98302 | 0.98984 | 0.97971 |
| Setup-4 | 0.92608 | 0.98342 | 0.98203 | 0.97116 |
| Setup-5 | 0.98721 | 0.99259 | 0.99530 | 0.98329 |
| Setup-6 | 0.99711 | 0.99072 | 0.99459 | 0.97788 |
| Setup-7 | 0.98742 | 0.97774 | 0.98812 | 0.97696 |
| Setup-8 | 0.98708 | 0.97979 | 0.99033 | 0.97015 |
| Setup-9 | 0.98560 | 0.99242 | 0.99178 | 0.98089 |
| Setup-10 | 0.92990 | 0.99250 | 0.98273 | 0.97852 |

Ниже в таблицах 4.21 и 4.22 приведены результаты тестирования алгоритмов при $P$=40% и $P$=30%, из которых видно, что метрики алгоритмов ухудшаются при меньших $P$, однако лучше всего себя показывают вторая и третья реализации.

Таблица 4.21 — Сравнительная точность алгоритмов при $P$=40%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98392 | 0.97110 | 0.98458 | 0.97101 |
| Setup-2 | 0.92207 | 0.98565 | 0.98811 | 0.96978 |
| Setup-3 | 0.98790 | 0.98673 | 0.99469 | 0.96583 |
| Setup-4 | 0.93110 | 0.97172 | 0.98087 | 0.97358 |
| Setup-5 | 0.99699 | 0.99101 | 0.99226 | 0.97240 |
| Setup-6 | 0.97574 | 0.98521 | 0.98801 | 0.97156 |
| Setup-7 | 0.93929 | 0.97331 | 0.98424 | 0.96869 |
| Setup-8 | 0.98739 | 0.98096 | 0.98768 | 0.97258 |
| Setup-9 | 0.91985 | 0.99168 | 0.98363 | 0.98132 |
| Setup-10 | 0.94946 | 0.96854 | 0.97489 | 0.96686 |

Таблица 4.22 — Сравнительная точность алгоритмов при $P$=30%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99762 | 0.99545 | 0.99242 | 0.99004 |
| Setup-2 | 0.98340 | 0.98605 | 0.98821 | 0.97615 |
| Setup-3 | 0.93003 | 0.96639 | 0.98728 | 0.97500 |
| Setup-4 | 0.99691 | 0.97708 | 0.99120 | 0.96853 |
| Setup-5 | 0.90497 | 0.95759 | 0.97752 | 0.97034 |
| Setup-6 | 0.99760 | 0.95397 | 0.97735 | 0.96059 |
| Setup-7 | 0.93203 | 0.97589 | 0.98692 | 0.96967 |
| Setup-8 | 0.98778 | 0.97299 | 0.97723 | 0.96790 |
| Setup-9 | 0.98731 | 0.98461 | 0.99403 | 0.97482 |
| Setup-10 | 0.98584 | 0.96671 | 0.98177 | 0.97084 |

Ниже в таблицах 4.23 и 4.24 приведены результаты тестирования алгоритмов при $P$=20% и $P$=10%, из которых видно, что метрики алгоритмов улучшаются, что, как и ранее, связано с меньшей неопределённостью в момент назначения сотрудника.

Таблица 4.23 — Сравнительная точность алгоритмов при $P$=20%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99824 | 0.99248 | 0.99829 | 0.98661 |
| Setup-2 | 0.89348 | 0.93581 | 0.96774 | 0.97561 |
| Setup-3 | 0.95631 | 0.96416 | 0.97879 | 0.97072 |
| Setup-4 | 0.99488 | 0.98209 | 0.99514 | 0.97067 |
| Setup-5 | 0.99846 | 0.97630 | 0.99466 | 0.97679 |
| Setup-6 | 0.99757 | 0.95347 | 0.97507 | 0.96373 |
| Setup-7 | 0.99808 | 0.96817 | 0.98469 | 0.95610 |
| Setup-8 | 0.91860 | 0.95927 | 0.97203 | 0.97151 |
| Setup-9 | 0.99733 | 0.99364 | 0.99281 | 0.98057 |
| Setup-10 | 0.97253 | 0.96817 | 0.98119 | 0.96030 |

Таблица 4.24 — Сравнительная точность алгоритмов при $P$=10%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99976 | 0.99153 | 0.99616 | 0.98669 |
| Setup-2 | 0.94303 | 0.98247 | 0.99376 | 0.97430 |
| Setup-3 | 0.95112 | 0.98748 | 0.99410 | 0.98447 |
| Setup-4 | 0.99878 | 0.98850 | 0.99730 | 0.98935 |
| Setup-5 | 0.99900 | 0.99081 | 0.99534 | 0.98039 |
| Setup-6 | 0.89487 | 0.98531 | 0.98225 | 0.97036 |
| Setup-7 | 0.99900 | 0.96767 | 0.99145 | 0.98230 |
| Setup-8 | 0.95987 | 0.98094 | 0.99092 | 0.98487 |
| Setup-9 | 0.99840 | 0.98377 | 0.99626 | 0.98181 |
| Setup-10 | 0.99918 | 0.97990 | 0.98627 | 0.98151 |

Четвёртый сценарий распределения задач (изображённый на рисунке 4.4) моделирует наличие пиковых дней и отличающихся пиковых и минимальных часов для различных типов задач.

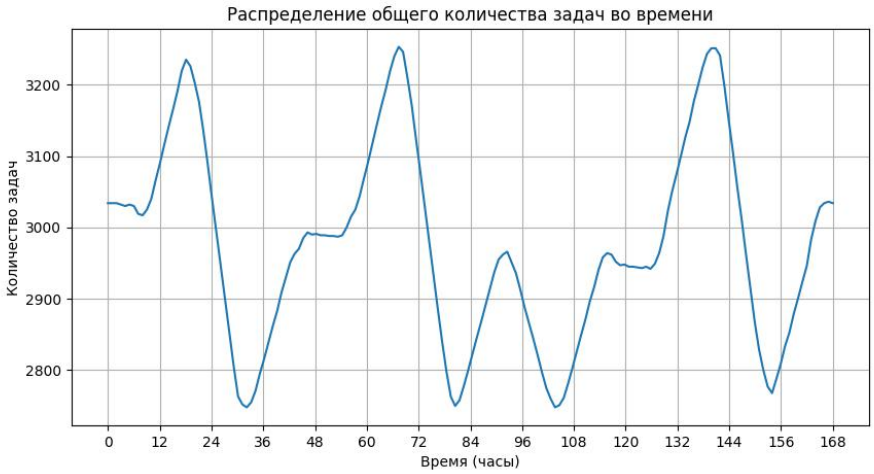


Рисунок 4.4 — Четвёртый сценарий распределения задач во времени

Ниже в таблицах 4.25 и 4.26 приведены результаты тестирования алгоритмов при $P$=60% и $P$=50%, из которых видно, что метрики алгоритмов достаточно хороши, что связано, как и ранее, с тем, что сотрудники хорошо взаимозаменяемы. Наличие пиковых дней, как и ранее, ухудшает метрики алгоритмов.

Таблица 4.25 — Сравнительная точность алгоритмов при $P$=60%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98774 | 0.98480 | 0.98594 | 0.97152 |
| Setup-2 | 0.99715 | 0.99165 | 0.99444 | 0.97825 |
| Setup-3 | 0.97694 | 0.99168 | 0.98772 | 0.98176 |
| Setup-4 | 0.98699 | 0.97910 | 0.98530 | 0.98085 |
| Setup-5 | 0.99737 | 0.98331 | 0.99072 | 0.97565 |
| Setup-6 | 0.98111 | 0.98879 | 0.99023 | 0.98440 |
| Setup-7 | 0.99564 | 0.97337 | 0.98513 | 0.96478 |
| Setup-8 | 0.93224 | 0.99009 | 0.98798 | 0.97934 |
| Setup-9 | 0.98345 | 0.98861 | 0.98906 | 0.97868 |
| Setup-10 | 0.99698 | 0.99402 | 0.99588 | 0.98360 |

Таблица 4.26 — Сравнительная точность алгоритмов при $P$=50%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99688 | 0.99472 | 0.99334 | 0.98668 |
| Setup-2 | 0.98641 | 0.99154 | 0.99144 | 0.98369 |
| Setup-3 | 0.99680 | 0.98086 | 0.98809 | 0.97426 |
| Setup-4 | 0.98677 | 0.98832 | 0.99146 | 0.97895 |
| Setup-5 | 0.99738 | 0.98891 | 0.99379 | 0.97962 |
| Setup-6 | 0.99784 | 0.96708 | 0.97905 | 0.96738 |
| Setup-7 | 0.89335 | 0.96119 | 0.97267 | 0.97645 |
| Setup-8 | 0.98735 | 0.98672 | 0.99203 | 0.97473 |
| Setup-9 | 0.98308 | 0.97454 | 0.98021 | 0.97589 |
| Setup-10 | 0.99767 | 0.98145 | 0.99348 | 0.97849 |

Ниже в таблицах 4.27 и 4.28 приведены результаты тестирования алгоритмов при $P$=40% и $P$=30%, из которых видно, что лучшие результаты в условиях максимальной неопределенности того, какого сотрудника назначить, показывают второй и третий алгоритмы. Однако, как и ранее, наличие пиковых дней ухудшает метрики по сравнению с первым тестовым сценарием.

Таблица 4.27 — Сравнительная точность алгоритмов при $P$=40%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.98795 | 0.96590 | 0.98827 | 0.96526 |
| Setup-2 | 0.88129 | 0.98175 | 0.97914 | 0.96644 |
| Setup-3 | 0.97484 | 0.98298 | 0.98804 | 0.97779 |
| Setup-4 | 0.90717 | 0.97736 | 0.98087 | 0.96965 |
| Setup-5 | 0.95360 | 0.98056 | 0.98241 | 0.96868 |
| Setup-6 | 0.98122 | 0.98341 | 0.98518 | 0.97621 |
| Setup-7 | 0.96425 | 0.98470 | 0.98518 | 0.97493 |
| Setup-8 | 0.92047 | 0.96739 | 0.97522 | 0.97372 |
| Setup-9 | 0.98706 | 0.98471 | 0.99235 | 0.97317 |
| Setup-10 | 0.98734 | 0.98650 | 0.99543 | 0.98108 |

Таблица 4.28 — Сравнительная точность алгоритмов при $P$=30%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.96573 | 0.99060 | 0.98847 | 0.97286 |
| Setup-2 | 0.99711 | 0.98521 | 0.99331 | 0.97206 |
| Setup-3 | 0.94287 | 0.97839 | 0.98443 | 0.96191 |
| Setup-4 | 0.92971 | 0.97054 | 0.97996 | 0.96177 |
| Setup-5 | 0.90151 | 0.95523 | 0.96354 | 0.96124 |
| Setup-6 | 0.94904 | 0.97926 | 0.98738 | 0.96580 |
| Setup-7 | 0.98230 | 0.97705 | 0.98752 | 0.97123 |
| Setup-8 | 0.99770 | 0.97274 | 0.98673 | 0.96523 |
| Setup-9 | 0.96029 | 0.97974 | 0.98720 | 0.96614 |
| Setup-10 | 0.95702 | 0.95936 | 0.98321 | 0.96806 |

Ниже в таблицах 4.29 и 4.30 приведены результаты тестирования алгоритмов при $P$=20% и $P$=10%, из которых видно, что метрики алгоритмов улучшаются, что, как и ранее, связано с меньшей неопределённостью в момент назначения сотрудника.

Таблица 4.29 — Сравнительная точность алгоритмов при $P$=20%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99700 | 0.94598 | 0.98097 | 0.95809 |
| Setup-2 | 0.86801 | 0.97571 | 0.97679 | 0.97940 |
| Setup-3 | 0.97384 | 0.96774 | 0.97842 | 0.96896 |
| Setup-4 | 0.99813 | 0.96821 | 0.98770 | 0.96481 |
| Setup-5 | 0.92984 | 0.98663 | 0.98437 | 0.97505 |
| Setup-6 | 0.92127 | 0.97537 | 0.98359 | 0.96788 |
| Setup-7 | 0.94984 | 0.99334 | 0.99329 | 0.98561 |
| Setup-8 | 0.99791 | 0.97497 | 0.98346 | 0.97106 |
| Setup-9 | 0.96990 | 0.95913 | 0.97343 | 0.95460 |
| Setup-10 | 0.99856 | 0.99086 | 0.99404 | 0.98100 |

Таблица 4.30 — Сравнительная точность алгоритмов при $P$=10%

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| Setup-1 | 0.99949 | 0.97317 | 0.99162 | 0.98677 |
| Setup-2 | 0.99906 | 0.99001 | 0.99393 | 0.97865 |
| Setup-3 | 0.99961 | 0.99630 | 0.99686 | 0.98933 |
| Setup-4 | 0.99914 | 0.98750 | 0.99366 | 0.97450 |
| Setup-5 | 0.90666 | 0.97997 | 0.98173 | 0.97720 |
| Setup-6 | 0.99894 | 0.98392 | 0.99155 | 0.97573 |
| Setup-7 | 0.99862 | 0.95763 | 0.99135 | 0.99109 |
| Setup-8 | 0.94941 | 0.97123 | 0.98123 | 0.98772 |
| Setup-9 | 0.99976 | 1.00000 | 1.00000 | 1.00000 |
| Setup-10 | 0.99953 | 0.98706 | 0.99504 | 0.98486 |

## 4.3 Тестирование скорости работы

Ниже приведены замеры скорости работы алгоритмов при разных параметрах $W$, $T$, $m$ и при фиксированном параметре $P$=50%.

Для ускорения сбора метрик использовались параллельные запуски алгоритма сразу на нескольких тестах с использованием встроенных средств *C++* (сущность *std::thread* и встроенные примитивы синхронизации).

Ниже в таблице 4.31 приведены результаты тестирования алгоритмов при $W$=100 и $T$=10, из которых видно, что второму алгоритму по сравнению с первым требуется больше времени на обработку последовательности задач, так как требуется постоянно обновлять данные о полезности каждого типа сотрудников. Третий и четвёртый алгоритм работают в среднем медленнее первого и второго, так как обладают более сложными математическими моделями, что замедляет обработку запросов.

Таблица 4.31 — Сравнительное время работы алгоритмов в миллисекундах при $W$=100, $T$=10

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| m = 500’000 | 8881 | 11696 | 9128 | 9320 |
| m = 100’000 | 1721 | 2175 | 1775 | 1761 |
| m = 50’000 | 807 | 990 | 828 | 823 |
| m = 10’000 | 143 | 154 | 151 | 149 |

Ниже в таблице 4.32 приведены результаты тестирования алгоритмов при $W$=10 и $T$=100, из которых видно, что при увеличившемся количестве типов задач третьему и четвертому алгоритмам по сравнению с первым и вторым требуется больше времени на обработку последовательности задач, так как вычисление полезности сотрудника в этих алгоритмах дольше и имеет большую скрытую константу асимптотики, чем в первой и второй реализациях.

Таблица 4.32 — Сравнительное время работы алгоритмов в миллисекундах при $W$=10, $T$=100

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| m = 500'000 | 3536 | 4323 | 11655 | 11751 |
| m = 100'000 | 660 | 789 | 2279 | 2287 |
| m = 50'000 | 330 | 386 | 1126 | 1122 |
| m = 10'000 | 63 | 69 | 215 | 212 |

Ниже в таблице 4.33 приведены результаты тестирования алгоритмов при $W$=100 и $T$=100, из которых видно, что алгоритмы достаточно быстро обрабатывают большие объёмы поступающих задач, подтверждая теоретические асимптотики.

Таблица 4.33 — Сравнительное время работы алгоритмов в миллисекундах при $W$=100, $T$=100

| | Algo-1 | Algo-2 | Algo-3 | Algo-4 |
|---|---|---|---|---|
| m = 500'000 | 10021 | 12521 | 17067 | 16955 |
| m = 100'000 | 2124 | 2654 | 3549 | 3537 |
| m = 50'000 | 959 | 1123 | 1661 | 1630 |
| m = 10'000 | 160 | 174 | 303 | 299 |

## 4.4 Выводы

Из вышеописанных тестов качества работы алгоритмов можно сделать вывод, что алгоритмы хорошо справляются с назначением задач при достаточно малых ($\leq 20\%$) параметрах $P$, что связано с тем, что при малом количестве навыков значительно сужается неопределенность в момент назначения, и при достаточно больших ($\geq 50\%$) $P$, что связано с хорошей взаимозаменяемостью сотрудников. Однако при $50\% > P > 20\%$ и в условиях наличия пиковых дней у задач более стабильно работают вторая и третья реализации.

Из тестов времени работы алгоритмов можно сделать вывод, что в среднем быстрее работают первая и вторая реализации, так как обладают более простыми математическими моделями, что ускоряет обработку запросов.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения работы были получены следующие результаты:

– разработаны четыре реализации требуемого алгоритма;

– протестирована корректность работы разработанных алгоритмов;

– разработана система моделирования процесса появления задач и работников;

– разработана система тестирования;

– проведены оценки качественных характеристик разработанных алгоритмов;

– реализованные алгоритмы позволяют распределять на сгенерированных тестовых сценариях не менее 85% выданных задач.

Программный код реализованных структур и тесты к ним выложены на github: https://github.com/AzyavchikovAlex/Coursework_7sem, https://github.com/AzyavchikovAlex/diploma.

В дальнейшем развитие дипломной работы может быть связано с усложнением математической модели задачи на случай более сложных распределений задач во времени и совершенствованием алгоритмов для неё, либо с разработкой прикладного программного обеспечения, позволяющего решать конкретные бизнес-задачи.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ


1. Introduction to algorithms / T. H. Cormen [and others (4th ed.)]; MIT Press and McGraw-Hill. ISBN 0-262-04630-X. – United States, 2022. – 1312 pp.
2. Vazirani, V.V. Approximation algorithms / V.V. Vazirani. – corrected second printing – Berlin Heidelberg: Springer-Verlag (Springer Science & Business Media), 2003 – 380 pp.
3. C++ documentation [Electronic resource]. – Carbon. – Mode of access: https://en.cppreference.com/w/. – Date of access: 05.05.2025.
4. Ahuja, R.K. Network flows: theory, algorithms, and applications / R.K. Ahuja, T.L. Magnanti, J.B. Orlin; production editor Merrill Peterson. – New Iersey: Prentice-Hall Inc., 1993. – 863 pp.
5. H. Christos, H. Combinatorial optimization: Algorithms and complexity / H. Christos, S. Kenneth. – Mineola, New York: Dover Publications Inc, 1998. – 496 pp.
6. MemorySanitizer documentation [Electronic resource]. – Clang 21.0.0git documentation. – Mode of access: https://clang.llvm.org/docs/MemorySanitizer.html. – Date of access: 05.05.2025.

# Интерфейс планировщика

```cpp
using TWorkerType = size_t;
using TTaskType = size_t;

struct TTaskTypeInfo {
    std::unordered_set<TWorkerType> Executors;
    std::unordered_set<size_t> PeakDays;
    TMultiplierType PeakDaysMultiplier;
    size_t PeakHour;
    size_t MinHour;
    TMultiplierType PeakHourMultiplier;
    TProbability Probability;
};

struct TWorkerTypeInfo {
    std::unordered_set<TTaskType> Abilities;
};


class IScheduler {
public:
    IScheduler() = default;

    virtual ~IScheduler() = default;

    virtual void AddWorkerTypeInfo(TWorkerType type,
                                   TWorkerTypeInfo info) = 0;

    virtual void AddTaskTypeInfo(TTaskType type,
                                 TTaskTypeInfo info) = 0;

    virtual void AddWorker(TTimestamp timestamp, THiredWorker worker) = 0;

    virtual void AddTask(TTimestamp timestamp,
                         TTaskType taskType,
                         std::function<void(int64_t)> onScheduledCallback) =
    0;
};
```

# Структуры для хранения тестов

```
struct THiredWorker {
  TWorkerType WorkerType;
  int64_t WorkerId;
};

struct TWeeklyHirePlan {
  std::array<std::vector<THiredWorker>, 7 * 24> WeeklyHirePlan;
  long double PlanOptimality{1};
};

struct TWeeklySetup {
  std::array<std::vector<TTaskType>, 7 * 24> WeeklyTasksLoad;

  std::unordered_map<TWorkerType, TWorkerTypeInfo> WorkerTypesInfo;
  std::unordered_map<TTaskType, TTaskTypeInfo> TaskTypesInfo;

  size_t WorkerDaySize{8}; // in hours
  size_t TaskDuration{3};  // in hours
};

struct TTest {
  TWeeklySetup Setup;
  TWeeklyHirePlan HirePlan;
};
```