|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт информационных технологий |
| Кафедра корпоративных информационных систем  **КУРСОВАЯ РАБОТА**  по дисциплине  Системы синхронизированного планирования ресурсов предприятия  **Тема курсовой работы**: Решение задачи о назначении с применением генетического алгоритма   |  |  |  | | --- | --- | --- | | Студент группы | ИКМО-03-23  Монаков Андрей Вячеславович | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) | |  |  |  | | Руководитель курсовой работы | профессор  Демидова Лилия Анатольевна | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись руководителя) | |  |  |  | | Работа представлена к защите | «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2025 г. |  | |  |  |  | | Допущен к защите | «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2025 г. |  | |

Москва 2025

****

**Оглавление**

[Введение 4](#_Toc192871535)

[1. Теоретическая часть 5](#_Toc192871536)

[1.1. Генетический алгоритм 5](#_Toc192871537)

[1.2. Сущность задачи о назначении 6](#_Toc192871538)

[2. Реализация задачи о назначении 8](#_Toc192871539)

[2.1. Реализации генетического алгоритма 8](#_Toc192871540)

[2.2. Полная реализация решения задачи о назначении 11](#_Toc192871541)

[Заключение 19](#_Toc192871542)

[Список источников 20](#_Toc192871543)

[Приложение А 21](#_Toc192871544)

# Введение

В текущую эпоху развития информационных технологий приходится сталкиваться со сбором, хранением и передачей огромных объемов данных, которые необходимо изучать, обрабатывать и применять выводы, полученные на их основе для дальнейшего развития. Чтобы данные задачи были выполнимы существует целый пласт алгоритмов по работе с данными, каждый из которых лучше подходит для разных случаев.

Одним из наиболее востребованных типов алгоритмов являются алгоритмы оптимизации данных. К ним относятся генетические алгоритмы и алгоритмы, основанные на поведении различных животных.

Целью данной работы является изучение задачи о назначении с применением генетического алгоритма. В рамках работы будут рассмотрены основные подходы к реализации алгоритма, будет исследована специфика предметной области и действующие в ней ограничения, а также представлена реализация алгоритма с решением поставленной в рамках темы задачи.

# 1. Теоретическая часть

## 1.1. Генетический алгоритм

Генетический алгоритм (GA) был предложен Джоном Холландом (J. Holland) в 1975 году. Он основан на принципах естественного отбора и генетики. В алгоритме используется популяция потенциальных решений, представленных в виде хромосом, которые эволюционируют по мере выполнения алгоритма. Каждая хромосома состоит из генов, представляющих различные параметры задачи [6].

Процесс работы генетического алгоритма начинается с инициализации начальной популяции случайным образом. Каждая хромосома оценивается с помощью фитнес-функции, которая измеряет качество решения. Алгоритм повторяет цикл выбора, скрещивания и мутации до достижения заданного критерия остановки [7].

Этапы генетического алгоритма:

1. Инициализация: Создание начальной популяции хромосом случайным образом.

2. Оценка: Вычисление фитнес-функции для каждой хромосомы.

3. Выбор: Отбор хромосом для создания новой популяции на основе их фитнеса. Часто используются методы рулеточного колеса или турнирного отбора.

4. Скрещивание (кроссинговер): Комбинирование пар хромосом для создания потомков. Обычно применяются одноточечный, двуточечный или униформный кроссинговеры.

5. Мутация: Внесение случайных изменений в гены хромосом для поддержания генетического разнообразия.

6. Замещение: Замена старой популяции новым поколением хромосом.

Генетический алгоритм формально описывается следующими уравнениями [7]:

Формула фитнес-функции:

(1.1)  
где x ‒ текущая хромосома, представляющая решение задачи.

Формула отбора:

(1.2)  
где P*i* ‒ вероятность отбора хромосомы x*i*, n ‒ размер популяции.

Формула кроссинговера:

*новыйродитель1родитель2*,(1.3)  
где alpha ‒ случайная величина в диапазоне [0, 1].

Формула мутации:

*новый* (1.4)  
где delta ‒ случайное отклонение.

Генетические алгоритмы находят широкое применение в задачах оптимизации и поисковых задачах, таких как планирование, распределение ресурсов, распознавание образов и др. Их эффективность обусловлена способностью эффективно исследовать большие и сложные пространства поиска, а также адаптироваться к динамическим условиям.

## 1.2. Сущность задачи о назначении

Задача о назначении представляет собой сложную задачу планирования времени и распределения ресурсов с учётом множества ограничений. Эта задача широко применяется в различных областях, таких как производство, логистика, образование и управление персоналом, где важно эффективно организовать выполнение работ для минимизации конфликтов и максимизации использования ресурсов.

Дано:

* Набор работ: Каждая работа имеет свои временные рамки (время начала и окончания), длительность, а также требования к ресурсам. Некоторые работы должны выполняться последовательно, что накладывает дополнительные ограничения на порядок их выполнения.
* Набор работников: Работники обладают определёнными навыками, которые позволяют им выполнять только некоторые работы качественно. Каждый работник имеет ограничения по рабочему времени (не более 8 часов в день) и должен иметь обеденный перерыв продолжительностью 30 минут.
* Ограничения:
  + Работы должны выполняться в заданные временные рамки.
  + Работник не может быть назначен на работу, которую он не может выполнить качественно.
  + Работы, зависящие друг от друга, должны выполняться в определённом порядке.
  + Рабочее время работника не должно превышать 8 часов в день, включая обеденный перерыв.

Необходимо:

* Составить оптимальное назначение для выполнения работ с учётом заданных ограничений.
* Минимизировать штрафы за нарушения ограничений, такие как:
  + Нарушение порядка выполнения работ.
  + Назначение работника на работу, которую он не может выполнить качественно.
  + Превышение рабочего времени.
  + Нарушение обеденного перерыва.
  + Повторение одной и той же работы одним и тем же работником
* Максимизировать эффективность использования рабочего времени и ресурсов.

Учёт времени, ресурсов, приоритетов и зависимостей:

* Время выполнения: Каждая работа имеет определённое время начала и окончания, которое должно быть строго соблюдено.
* Ресурсы: Каждая работа требует определённых навыков, которые должны быть доступны у назначенного работника.
* Зависимости: Некоторые работы зависят от выполнения других работ, что требует соблюдения порядка выполнения.

Таким образом, задача о назначении заключается в нахождении оптимального распределения времени и ресурсов для выполнения всех работ с учётом всех заданных ограничений и приоритетов. Решение этой задачи позволяет минимизировать штрафы за нарушения и обеспечить эффективное использование рабочего времени и ресурсов.

# 2. Реализация задачи о назначении

Чтобы реализовать решение данной задачи, необходимо применить генетический алгоритм и использовать систему штрафов из таблицы 1.

Таблица 1 – Система штрафов

|  |  |  |
| --- | --- | --- |
| **№** | Критерий | Штрафы, баллы |
| 1. | За нарушение порядка выполнения 2 последовательных работ. | 100 |
| 2. | За обратный порядок выполнения последовательных работ. | Число последовательных работ\*100 |
| 3. | За повторное выполнение работ, тем же работником. | 80 |
| 4. | За назначение работника на работу, которую он может выполнить некачественно | 80 |
| 5. | За рабочий день более 8 часов | 100 |
| 6. | За обеденный перерыв менее 30 минут | 100 |
| 7. | За смещение обеденного перерыва из интервала [12-00, 14-00]. | Число минут\*2 |

## **2.1. Реализации генетического алгоритма**

1. Инициализация популяции

Функция create\_schedule создает начальное назначение для каждого работника. Для каждой работы, которую работник может выполнить, случайным образом выбирается время начала в пределах допустимого интервала.

Входные данные:

* + Список работников (workers) с их навыками.
  + Список работ (works) с временными рамками и зависимостями.

Выходные данные:

* + Список назначений для каждого работника.

2. Оценка приспособленности

Функция evaluate\_schedule рассчитывает штрафы для назначения на основе нарушений ограничений:

* Нарушение порядка выполнения работ.
* Назначение на работу, которую работник не может выполнить качественно.
* Превышение рабочего времени (более 8 часов).
* Нарушение обеденного перерыва (менее 30 минут или вне интервала [12:00, 14:00]).
* Повторное выполнение работ, тем же работником.

Входные данные:

* + Назначение работников (schedule).

Выходные данные:

* + Суммарный штраф для назначения.

3. Селекция

Функция select\_best\_schedules выбирает лучшие назначения из популяции на основе штрафов. Используется сортировка по возрастанию штрафов.

Входные данные:

* + Популяция расписаний (population).
  + Количество лучших нащначений для выбора (num\_best).

Выходные данные:

* + Список лучших назначений.

4. Мутация

Функция mutate\_schedule вносит случайные изменения в расписание:

* Изменение времени начала задачи.
* Перемещение задачи в другое место в расписании.

Входные данные:

* + Расписание работников (schedule).

Выходные данные:

* + Мутированное расписание.

5. Создание новой популяции

Функция genetic\_algorithm реализует основной цикл ГА:

* Создание начальной популяции.
* Оценка приспособленности.
* Селекция лучших решений.
* Мутация и создание новой популяции.
* Повторение процесса для заданного числа поколений.

Входные данные:

* + Количество поколений (num\_generations).
  + Размер популяции (population\_size).

Выходные данные:

* + Лучшее расписание.
  + Список штрафов для каждого поколения.

6. Островная модель

Функция island\_model реализует островную модель ГА, где популяции разделены на несколько "островов". Между островами происходит миграция лучших решений, что позволяет улучшить качество оптимизации.

Входные данные:

* + Количество поколений (num\_generations).
  + Размер популяции (population\_size).
  + Количество островов (num\_islands).
  + Частота миграции (migration\_rate).

Выходные данные:

* + Лучшее расписание.
  + Список штрафов для каждого поколения.

## 2.2. Полная реализация решения задачи о назначении

Работы и работники задаются в виде списка словарей. Каждая работа и работник описываются своими характеристиками.

Листинг 1 - Инициализация данных

# Работа

works = [

{"name": "Подготовка деталей", "start\_time": 9, "end\_time": 12, "duration": 1, "dependencies": []},

{"name": "Сборка лампы", "start\_time": 11, "end\_time": 15, "duration": 3, "dependencies": ["Покраска корпуса"]},

# Другие работы...

]

# Работники

workers = [

{"name": "Иван", "skills": ["Подготовка деталей", "Сборка лампы", "Изготовление плафона", "Ремонт оборудования"], "max\_hours": 8},

# Другие работники...

]

Штрафы задаются как константы. Они используются для оценки качества расписания.

Листинг 2 - Штрафы

PENALTY\_ORDER\_VIOLATION = 100 # Нарушение порядка выполнения

PENALTY\_REVERSE\_ORDER = 100 # Обратный порядок выполнения

PENALTY\_LOW\_QUALITY = 80 # Назначение на работу, которую работник не может выполнить качественно

PENALTY\_OVERTIME = 100 # Рабочий день более 8 часов

PENALTY\_SHORT\_LUNCH = 100 # Обеденный перерыв менее 30 минут

PENALTY\_LUNCH\_TIME = 2 # Штраф за каждую минуту смещения обеденного перерыва

PENALTY\_REPEATED\_WORK = 80 # Штраф за повторное назначение одной и той же работы одному сотруднику

Функция create\_schedule создает начальное расписание для каждого работника. Для каждой работы, которую работник может выполнить, случайным образом выбирается время начала в пределах допустимого интервала.

Листинг 3 – Создание начального расписания

def create\_schedule():

    schedule = []

    for worker in workers:

        worker\_schedule = []

        for work in works:

            if work["name"] in worker["skills"]:

                start\_time = random.randint(work["start\_time"], work["end\_time"] - work["duration"])

                worker\_schedule.append((work["name"], start\_time, start\_time + work["duration"]))

        schedule.append(worker\_schedule)

    return schedule

Функция evaluate\_schedule оценивает расписание, рассчитывая штрафы за нарушения ограничений. Каждый тип нарушения проверяется отдельно.

Листинг 4 – Оценка расписания

def evaluate\_schedule(schedule):

    penalty = 0

    # Проверка порядка выполнения работ

    for work in works:

        for dependency in work["dependencies"]:

            for worker\_schedule in schedule:

                work\_start = None

                dependency\_start = None

                for task in worker\_schedule:

                    if task[0] == work["name"]:

                        work\_start = task[1]

                    if task[0] == dependency:

                        dependency\_start = task[1]

                if work\_start is not None and dependency\_start is not None:

                    if work\_start < dependency\_start:

                        penalty += PENALTY\_ORDER\_VIOLATION

                    elif work\_start > dependency\_start:

                        penalty += PENALTY\_REVERSE\_ORDER

    # Проверка качества выполнения работ

    for worker\_schedule in schedule:

        for task in worker\_schedule:

            if task[0] not in workers[schedule.index(worker\_schedule)]["skills"]:

                penalty += PENALTY\_LOW\_QUALITY

    # Проверка рабочего времени и обеденного перерыва

    for worker\_schedule in schedule:

        total\_hours = sum(task[2] - task[1] for task in worker\_schedule)

        if total\_hours > 8:

            penalty += PENALTY\_OVERTIME

        # Проверка обеденного перерыва (30 минут)

        lunch\_duration = 0

        for task in worker\_schedule:

            if 12 <= task[1] <= 14:

                lunch\_duration += task[2] - task[1]

        if lunch\_duration < 0.5:

            penalty += PENALTY\_SHORT\_LUNCH

        elif lunch\_duration > 0.5:

            penalty += PENALTY\_LUNCH\_TIME \* abs(lunch\_duration - 0.5)

    # Проверка повторного назначения одной и той же работы одному сотруднику

    for worker\_schedule in schedule:

        tasks = [task[0] for task in worker\_schedule]

        if len(tasks) != len(set(tasks)):

            penalty += PENALTY\_REPEATED\_WORK

    return penalty

Функция mutate\_schedule вносит случайные изменения в расписание.

Листинг 5 – Мутация расписания

def mutate\_schedule(schedule):

mutated\_schedule = [worker\_schedule.copy() for worker\_schedule in schedule]

worker\_idx = random.randint(0, len(mutated\_schedule) - 1)

worker\_schedule = mutated\_schedule[worker\_idx]

if len(worker\_schedule) > 0:

task\_idx = random.randint(0, len(worker\_schedule) - 1)

task = worker\_schedule[task\_idx]

new\_start\_time = random.randint(task[1] - 1, task[1] + 1)

new\_start\_time = max(works[task\_idx]["start\_time"], min(works[task\_idx]["end\_time"] - works[task\_idx]["duration"], new\_start\_time))

worker\_schedule[task\_idx] = (task[0], new\_start\_time, new\_start\_time + works[task\_idx]["duration"])

if len(worker\_schedule) > 1:

new\_position = random.randint(0, len(worker\_schedule) - 1)

worker\_schedule[task\_idx], worker\_schedule[new\_position] = worker\_schedule[new\_position], worker\_schedule[task\_idx]

return mutated\_schedule

Функция select\_best\_schedules выбирает лучшие расписания из популяции на основе штрафов

Листинг 6 – Селекция лучших расписаний

def select\_best\_schedules(population, num\_best=5):

evaluated\_population = [(schedule, evaluate\_schedule(schedule)) for schedule in population]

evaluated\_population.sort(key=lambda x: x[1])

return [schedule for schedule, \_ in evaluated\_population[:num\_best]]

Функция genetic\_algorithm реализует основной цикл генетического алгоритма.

Листинг 7 – Генетический алгоритм

def genetic\_algorithm(num\_generations=100, population\_size=20):

population = [create\_schedule() for \_ in range(population\_size)]

penalties = []

for \_ in range(num\_generations):

best\_schedules = select\_best\_schedules(population)

new\_population = best\_schedules.copy()

while len(new\_population) < population\_size:

parent = random.choice(best\_schedules)

offspring = mutate\_schedule(parent)

new\_population.append(offspring)

population = new\_population

penalties.append(evaluate\_schedule(best\_schedules[0]))

return population[0], penalties

Функция island\_model реализует островную модель генетического алгоритма, где популяции разделены на несколько "островов".

Листинг 8 – Островная моедель.

def island\_model(num\_generations=100, population\_size=20, num\_islands=5, migration\_rate=10):

islands = [[create\_schedule() for \_ in range(population\_size)] for \_ in range(num\_islands)]

penalties = []

for generation in range(num\_generations):

for i in range(num\_islands):

best\_schedules = select\_best\_schedules(islands[i])

new\_population = best\_schedules.copy()

while len(new\_population) < population\_size:

parent = random.choice(best\_schedules)

offspring = mutate\_schedule(parent)

new\_population.append(offspring)

islands[i] = new\_population

if generation % migration\_rate == 0:

best\_solutions = [select\_best\_schedules(island, num\_best=1)[0] for island in islands]

for i in range(num\_islands):

next\_island = (i + 1) % num\_islands

islands[next\_island].append(best\_solutions[i])

if len(islands[next\_island]) > population\_size:

islands[next\_island] = select\_best\_schedules(islands[next\_island], num\_best=population\_size)

penalties.append(evaluate\_schedule(select\_best\_schedules(islands[0], num\_best=1)[0]))

best\_schedules = [select\_best\_schedules(island, num\_best=1)[0] for island in islands]

best\_schedule = min(best\_schedules, key=lambda s: evaluate\_schedule(s))

return best\_schedule, penalties

Критерий Уилкоксона используется для сравнения двух связанных выборок (в данном случае — штрафов базовой и островной моделей). Он проверяет, есть ли статистически значимые различия между двумя наборами данных.

Листинг 9 – Критерий Уилкоксона

statistic, p\_value = Wilcoxon (penalties\_base, penalties\_island)

* Нулевая гипотеза (H₀): Нет различий между двумя моделями.
* Альтернативная гипотеза (H₁): Есть различия между двумя моделями.

Если p-value < 0.05, нулевая гипотеза отвергается, и различия считаются статистически значимыми.

Функции visualize\_schedule, run\_experiments и plot\_island\_penalties используются для визуализации расписания, сравнения моделей и анализа влияния числа островов на качество оптимизации.

Таким образом, решение задачи оптимизации функции посредством применения генетического алгоритма найдёт оптимальный набор входных параметров, при котором значение функции общего штрафа будет минимальным. Оптимальным набором, полученным в результате работы алгоритма, будет являться назначение работ при минимуме штрафов.

Для оценки эффективности работы генетического алгоритма в задаче поиска оптимального решения были замерены различные метрики, отражающие качество работы алгоритма в задаче. На рисунке 1 продемонстрировано время выполнения решения задачи двумя методами, количество штрафов, статистика Уилкоксона и p-value [8].

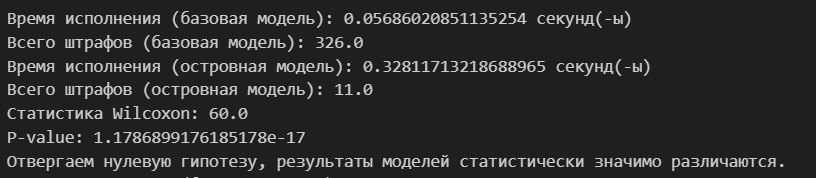


Рисунок 1 ‒ Метрики

Критерий Уилкоксона ‒ это непараметрический статистический тест, используемый для сравнения двух связанных выборок. Он проверяет гипотезу о том, что различия между парами выборок имеют симметричное распределение вокруг нуля.

Если вычисленное p-value меньше выбранного уровня значимости alpha, то нулевую гипотезу следует отвергнуть. Это означает, что существует статистически значимая разница между сравниваемыми выборками.

В приведенном примере, при alpha = 0.05, вычисленное p-value меньше 0.05. Поэтому нулевая гипотеза отвергается, и можно сделать вывод о наличии статистически значимой разницы между базовой моделью и островной моделью с 5 островами.

На рисунке 2 и 3 можно увидеть фрагмент результата решения задачи о назначении. В нем выведено лучшее расписание для базовой и островной модели с 5ю островами.

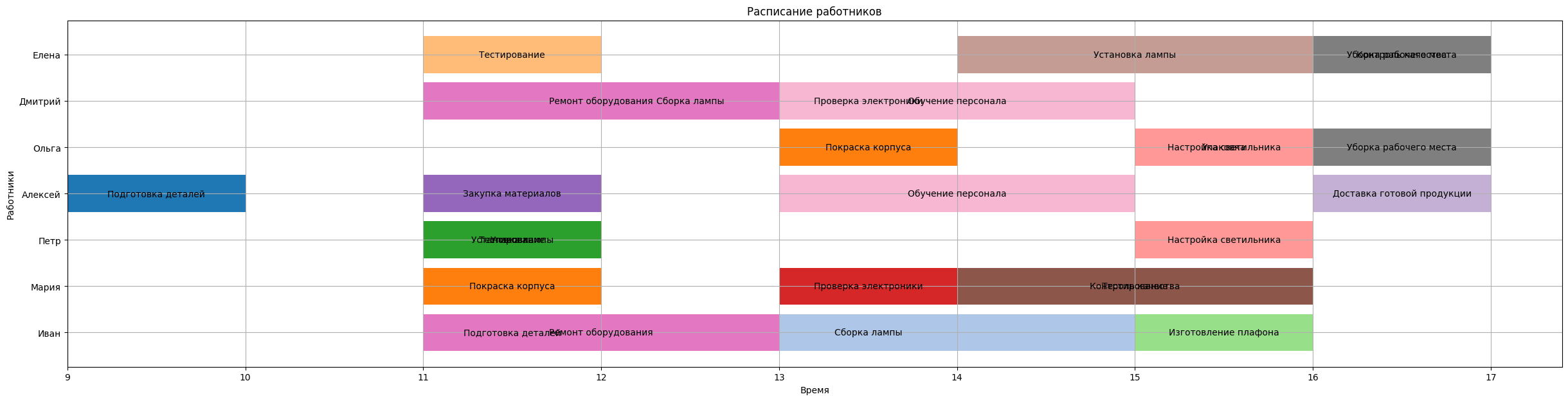


Рисунок 2 ‒ Лучшее назначение работников на рабочий день (базовая модель)

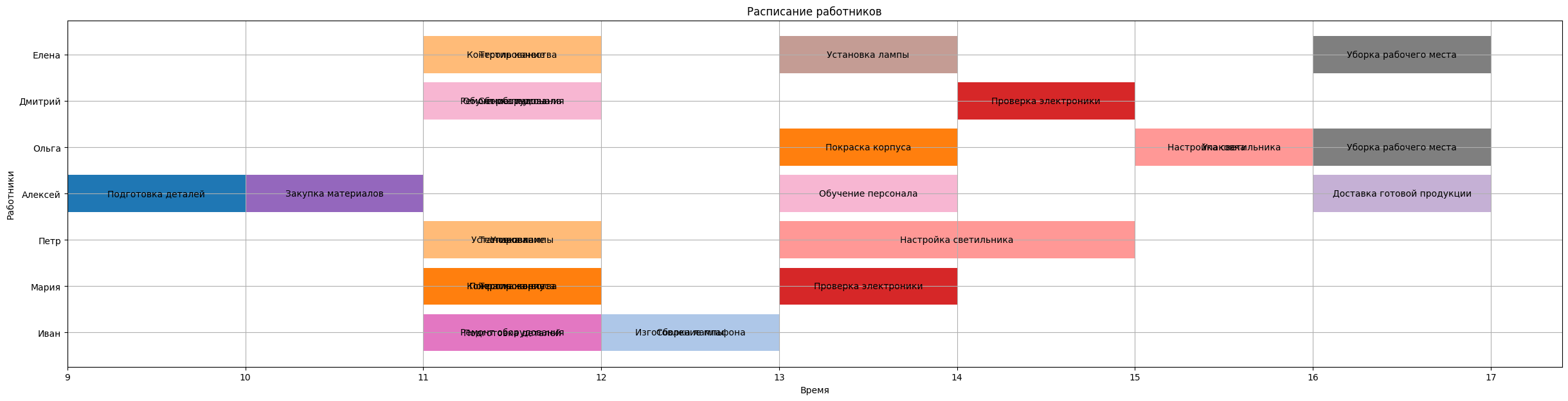


Рисунок 3 ‒ Лучшее назначение работников на рабочий день (островная модель)

Также были построены графики для более наглядного сравнения результатов работы базовой и островной моделей (Рисунок 4).

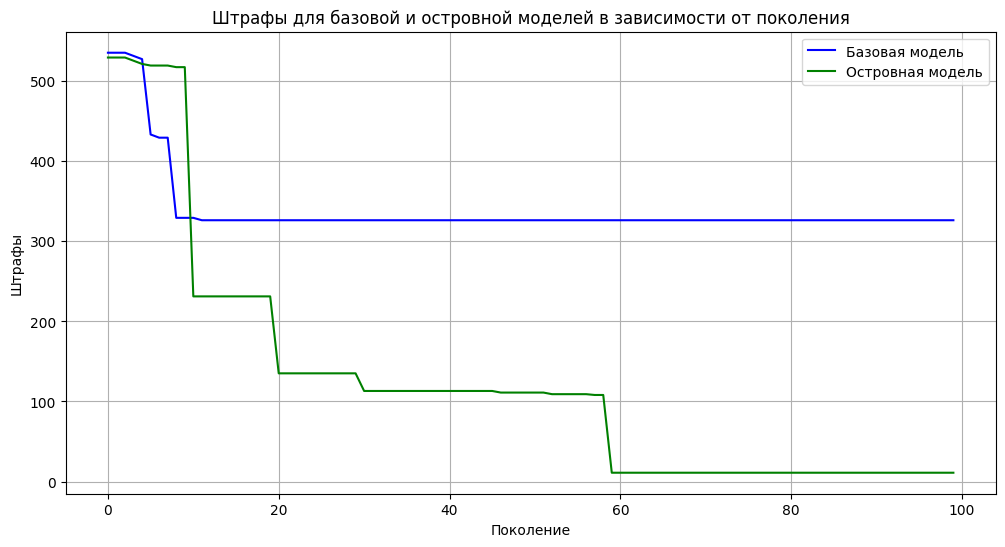


Рисунок 4 ‒ График штрафов для базовой и островной моделей

При анализе графика можно заметить, что в то время, как базовая модель показывает «усредненный» результат ‒ островная модель позволяет значительно сократить штрафы при решении задачи.

Вместе с графиком на рисунке 4 были выведены отдельные показатели для каждого из островов (Рисунок 5).

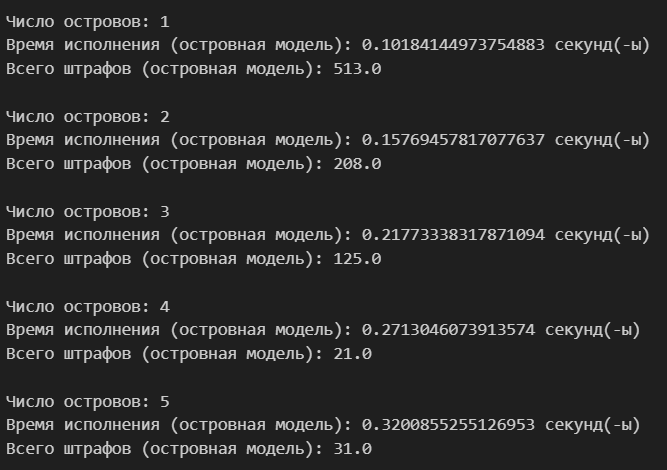


Рисунок 5 ‒ Метрики времени и штрафов для каждого острова

Более наглядно динамику количества островов и количества штрафов можно увидеть на следующих графике (Рисунок 6).

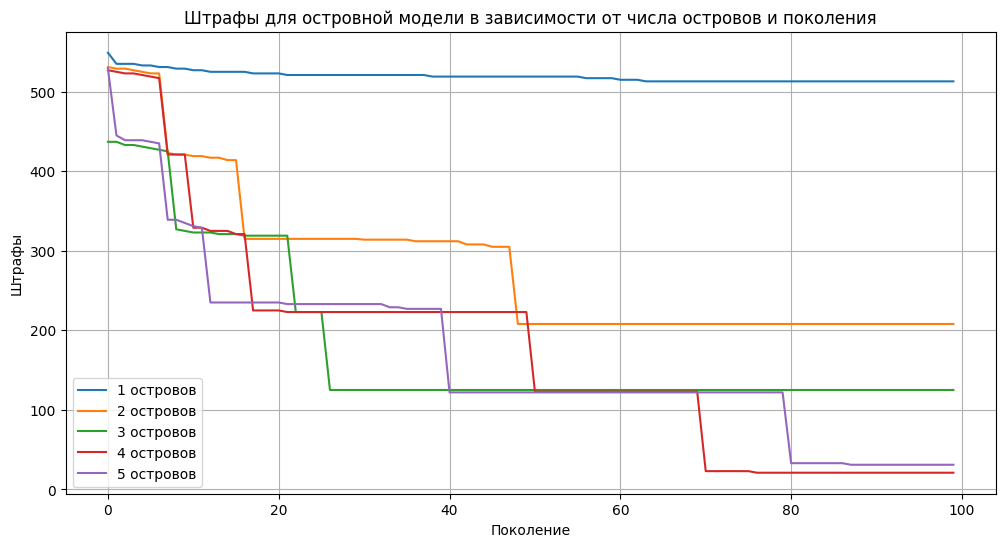


Рисунок 6 ‒ График штрафов островных моделей

При рассмотрении графиков и логов можно заметить, что с большим числом островов, используемых при вычислениях, растёт сходимость к нулю и равномерное возрастание времени требуемого на выполнение кода в зависимости от количества островов.

# Заключение

Во время выполнения данной работы была детально рассмотрена задача о назначении и её решение с применением генетического алгоритма. Реализация осуществлялась с помощью языка программирования Python. При реализации генетического алгоритма была использована базовая островная модель, которая позволяет получать более точные результаты за счёт обмена лучшими решениями между островами. Это особенно полезно для задач с большим количеством ограничений и сложной структурой зависимостей.

Были заранее заданы входные данные, включающие список работ с их временными рамками, длительностью и зависимостями, а также список работников с их навыками и ограничениями по рабочему времени. Эти данные были обработаны с помощью базовой модели (с одним островом) и островной модели (с пятью островами). Результатом работы алгоритмов стали скомпонованные расписания с учётом штрафов, которые являются метрикой «правильности» распределения работ по времени и работникам.

Следовательно, для сложных задач с различными комбинациями параметров наиболее эффективно использовать генетический алгоритм с островной моделью. Этот подход обеспечивает более высокую точность и стабильность в нахождении оптимального расписания.

# Список источников

1. Тарасов И.Е. Системы синхронизированного планирования ресурсов предприятия [Электронный ресурс]: учебное пособие / И.Е. Тарасов. – М.: МИРЭА – Российский технологический университет, 2021. – 96 с. (Дата обращения: 22.02.2025)
2. ГОСТ Р ИСО/МЭК 15288-2005. Информационная технология. Системная инженерия. Процессы жизненного цикла систем. (Дата обращения: 22.02.2025)
3. ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств. (Дата обращения: 22.02.2025)
4. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. (Дата обращения: 22.02.2025)
5. ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы. (Дата обращения: 22.02.2025)
6. Genetic Algorithms [Электронный ресурс]: https://www.geeksforgeeks.org/genetic-algorithms/ (дата обращения 28.02.2025)
7. Генетические алгоритмы. Лекция [Электронный ресурс]: https://sdo.nsuem.ru/pluginfile.php/310628/mod\_resource/content/1/ЛЕКЦИЯ%20генетические%20алгоритмы\_с.pdf (Дата обращения: 04.03.2024)
8. Исследование метрик оценки кода при формировании наборов данных с использованием генетического алгоритма [Электронный ресурс]: https://cyberleninka.ru/article/n/issledovanie-metrik-otsenki-koda-pri-formirovanii-naborov-dannyh-s-ispolzovaniem-geneticheskogo-algoritma/viewer (Дата обращения: 08.03.2024)

# Приложение А

import numpy as np

import time

import random

from scipy.stats import wilcoxon  # Используем Wilcoxon signed-rank test

import matplotlib.pyplot as plt

#работа

works = [

     {"name": "Подготовка деталей", "start\_time": 9, "end\_time": 12, "duration": 1, "dependencies": []},

    {"name": "Сборка лампы", "start\_time": 11, "end\_time": 15, "duration": 3, "dependencies": ["Покраска корпуса"]},

    {"name": "Покраска корпуса", "start\_time": 10, "end\_time": 14, "duration": 1, "dependencies": ["Подготовка деталей"]},

    {"name": "Тестирование", "start\_time": 13, "end\_time": 16, "duration": 2, "dependencies": []},

    {"name": "Упаковка", "start\_time": 15, "end\_time": 17, "duration": 1, "dependencies": ["Тестирование"]},

    {"name": "Изготовление плафона", "start\_time": 13, "end\_time": 16, "duration": 1, "dependencies": ["Сборка лампы"]},

    {"name": "Проверка электроники", "start\_time": 13, "end\_time": 15, "duration": 1, "dependencies": []},

    {"name": "Настройка светильника", "start\_time": 15, "end\_time": 16, "duration": 1, "dependencies": ["Сборка лампы"]},

    {"name": "Закупка материалов", "start\_time": 9, "end\_time": 13, "duration": 1, "dependencies": []},

    {"name": "Доставка готовой продукции", "start\_time": 16, "end\_time": 17, "duration": 1, "dependencies": ["Упаковка"]},

    {"name": "Контроль качества", "start\_time": 15, "end\_time": 17, "duration": 1, "dependencies": ["Тестирование"]},

    {"name": "Установка лампы", "start\_time": 14, "end\_time": 16, "duration": 2, "dependencies": ["Упаковка"]},

    {"name": "Ремонт оборудования", "start\_time": 10, "end\_time": 14, "duration": 2, "dependencies": ["Подготовка деталей"]},

    {"name": "Обучение персонала", "start\_time": 13, "end\_time": 16, "duration": 2, "dependencies": []},

    {"name": "Уборка рабочего места", "start\_time": 16, "end\_time": 17, "duration": 1, "dependencies": []},

]

#работники

workers = [

    {"name": "Иван", "skills": ["Подготовка деталей", "Сборка лампы", "Изготовление плафона", "Ремонт оборудования"], "max\_hours": 8},

    {"name": "Мария", "skills": ["Покраска корпуса", "Тестирование", "Проверка электроники", "Контроль качества"], "max\_hours": 8},

    {"name": "Петр", "skills": ["Упаковка", "Тестирование", "Настройка светильника", "Установка лампы"], "max\_hours": 8},

    {"name": "Алексей", "skills": ["Закупка материалов", "Подготовка деталей", "Доставка готовой продукции", "Обучение персонала"], "max\_hours": 8},

    {"name": "Ольга", "skills": ["Покраска корпуса", "Настройка светильника", "Упаковка", "Уборка рабочего места"], "max\_hours": 8},

    {"name": "Дмитрий", "skills": ["Сборка лампы", "Проверка электроники", "Ремонт оборудования", "Обучение персонала"], "max\_hours": 8},

    {"name": "Елена", "skills": ["Тестирование", "Контроль качества", "Установка лампы", "Уборка рабочего места"], "max\_hours": 8},

]

# Штрафы

PENALTY\_ORDER\_VIOLATION = 100  # Нарушение порядка выполнения

PENALTY\_REVERSE\_ORDER = 100    # Обратный порядок выполнения

PENALTY\_LOW\_QUALITY = 80       # Назначение на работу, которую работник не может выполнить качественно

PENALTY\_OVERTIME = 100         # Рабочий день более 8 часов

PENALTY\_SHORT\_LUNCH = 100      # Обеденный перерыв менее 30 минут

PENALTY\_LUNCH\_TIME = 2         # Штраф за каждую минуту смещения обеденного перерыва

PENALTY\_REPEATED\_WORK = 80     # Штраф за повторное назначение одной и той же работы одному сотруднику

# Функция для создания начальной популяции

def create\_schedule():

    schedule = []

    for worker in workers:

        worker\_schedule = []

        for work in works:

            if work["name"] in worker["skills"]:

                start\_time = random.randint(work["start\_time"], work["end\_time"] - work["duration"])

                worker\_schedule.append((work["name"], start\_time, start\_time + work["duration"]))

        schedule.append(worker\_schedule)

    return schedule

# Функция для оценки расписания

def evaluate\_schedule(schedule):

    penalty = 0

    # Проверка порядка выполнения работ

    for work in works:

        for dependency in work["dependencies"]:

            for worker\_schedule in schedule:

                work\_start = None

                dependency\_start = None

                for task in worker\_schedule:

                    if task[0] == work["name"]:

                        work\_start = task[1]

                    if task[0] == dependency:

                        dependency\_start = task[1]

                if work\_start is not None and dependency\_start is not None:

                    if work\_start < dependency\_start:

                        penalty += PENALTY\_ORDER\_VIOLATION

                    elif work\_start > dependency\_start:

                        penalty += PENALTY\_REVERSE\_ORDER

    # Проверка качества выполнения работ

    for worker\_schedule in schedule:

        for task in worker\_schedule:

            if task[0] not in workers[schedule.index(worker\_schedule)]["skills"]:

                penalty += PENALTY\_LOW\_QUALITY

    # Проверка рабочего времени и обеденного перерыва

    for worker\_schedule in schedule:

        total\_hours = sum(task[2] - task[1] for task in worker\_schedule)

        if total\_hours > 8:

            penalty += PENALTY\_OVERTIME

        # Проверка обеденного перерыва (30 минут)

        lunch\_duration = 0

        for task in worker\_schedule:

            if 12 <= task[1] <= 14:

                lunch\_duration += task[2] - task[1]

        if lunch\_duration < 0.5:

            penalty += PENALTY\_SHORT\_LUNCH

        elif lunch\_duration > 0.5:

            penalty += PENALTY\_LUNCH\_TIME \* abs(lunch\_duration - 0.5)

    # Проверка повторного назначения одной и той же работы одному сотруднику

    for worker\_schedule in schedule:

        tasks = [task[0] for task in worker\_schedule]

        if len(tasks) != len(set(tasks)):

            penalty += PENALTY\_REPEATED\_WORK

    return penalty

# Функция мутации

def mutate\_schedule(schedule):

    mutated\_schedule = [worker\_schedule.copy() for worker\_schedule in schedule]

    worker\_idx = random.randint(0, len(mutated\_schedule) - 1)

    worker\_schedule = mutated\_schedule[worker\_idx]

    if len(worker\_schedule) > 0:

        task\_idx = random.randint(0, len(worker\_schedule) - 1)

        task = worker\_schedule[task\_idx]

        new\_start\_time = random.randint(task[1] - 1, task[1] + 1)

        new\_start\_time = max(works[task\_idx]["start\_time"], min(works[task\_idx]["end\_time"] - works[task\_idx]["duration"], new\_start\_time))

        worker\_schedule[task\_idx] = (task[0], new\_start\_time, new\_start\_time + works[task\_idx]["duration"])

        if len(worker\_schedule) > 1:

            new\_position = random.randint(0, len(worker\_schedule) - 1)

            worker\_schedule[task\_idx], worker\_schedule[new\_position] = worker\_schedule[new\_position], worker\_schedule[task\_idx]

    return mutated\_schedule

# Функция отбора лучших расписаний

def select\_best\_schedules(population, num\_best=5):

    evaluated\_population = [(schedule, evaluate\_schedule(schedule)) for schedule in population]

    evaluated\_population.sort(key=lambda x: x[1])

    return [schedule for schedule, \_ in evaluated\_population[:num\_best]]

# Основной алгоритм

def genetic\_algorithm(num\_generations=100, population\_size=20):

    population = [create\_schedule() for \_ in range(population\_size)]

    penalties = []

    for \_ in range(num\_generations):

        best\_schedules = select\_best\_schedules(population)

        new\_population = best\_schedules.copy()

        while len(new\_population) < population\_size:

            parent = random.choice(best\_schedules)

            offspring = mutate\_schedule(parent)

            new\_population.append(offspring)

        population = new\_population

        penalties.append(evaluate\_schedule(best\_schedules[0]))

    return population[0], penalties

# Островная модель

def island\_model(num\_generations=100, population\_size=20, num\_islands=5, migration\_rate=10):

    islands = [[create\_schedule() for \_ in range(population\_size)] for \_ in range(num\_islands)]

    penalties = []

    for generation in range(num\_generations):

        for i in range(num\_islands):

            best\_schedules = select\_best\_schedules(islands[i])

            new\_population = best\_schedules.copy()

            while len(new\_population) < population\_size:

                parent = random.choice(best\_schedules)

                offspring = mutate\_schedule(parent)

                new\_population.append(offspring)

            islands[i] = new\_population

        if generation % migration\_rate == 0:

            best\_solutions = [select\_best\_schedules(island, num\_best=1)[0] for island in islands]

            for i in range(num\_islands):

                next\_island = (i + 1) % num\_islands

                islands[next\_island].append(best\_solutions[i])

                if len(islands[next\_island]) > population\_size:

                    islands[next\_island] = select\_best\_schedules(islands[next\_island], num\_best=population\_size)

        penalties.append(evaluate\_schedule(select\_best\_schedules(islands[0], num\_best=1)[0]))

    best\_schedules = [select\_best\_schedules(island, num\_best=1)[0] for island in islands]

    best\_schedule = min(best\_schedules, key=lambda s: evaluate\_schedule(s))

    return best\_schedule, penalties

# Визуализация расписания

def visualize\_schedule(schedule):

    fig, ax = plt.subplots(figsize=(30, 7))

    colors = plt.cm.tab20.colors  # Цвета для задач

    # Создаем словарь для хранения цветов задач

    task\_colors = {work["name"]: colors[i % len(colors)] for i, work in enumerate(works)}

    for i, worker\_schedule in enumerate(schedule):

        for task in worker\_schedule:

            ax.barh(i, task[2] - task[1], left=task[1], color=task\_colors[task[0]])

            ax.text((task[1] + task[2]) / 2, i, task[0], ha='center', va='center', color='black')

    ax.set\_yticks(range(len(workers)))

    ax.set\_yticklabels([worker["name"] for worker in workers])

    ax.set\_xlabel("Время")

    ax.set\_ylabel("Работники")

    ax.set\_title("Расписание работников")

    plt.grid(True)

    plt.show()

# Сравнение моделей

def run\_experiments():

    # Базовая модель

    start\_time = time.time()

    best\_schedule\_base, penalties\_base = genetic\_algorithm()

    end\_time = time.time()

    execution\_time\_base = end\_time - start\_time

    total\_penalty\_base = evaluate\_schedule(best\_schedule\_base)

    # Островная модель

    start\_time = time.time()

    best\_schedule\_island, penalties\_island = island\_model()

    end\_time = time.time()

    execution\_time\_island = end\_time - start\_time

    total\_penalty\_island = evaluate\_schedule(best\_schedule\_island)

    # Wilcoxon signed-rank test

    statistic, p\_value = wilcoxon(penalties\_base, penalties\_island)

    # Вывод результатов

    print(f"Время исполнения (базовая модель): {execution\_time\_base} секунд(-ы)")

    print(f"Всего штрафов (базовая модель): {total\_penalty\_base}")

    print(f"Время исполнения (островная модель): {execution\_time\_island} секунд(-ы)")

    print(f"Всего штрафов (островная модель): {total\_penalty\_island}")

    print(f"Статистика Wilcoxon: {statistic}")

    print(f"P-value: {p\_value}")

    if p\_value < 0.05:

        print("Отвергаем нулевую гипотезу, результаты моделей статистически значимо различаются.")

    else:

        print("Не можем отвергнуть нулевую гипотезу, результаты моделей не статистически значимо различаются.")

    # Визуализация расписания

    print("Лучшее расписание (базовая модель):")

    visualize\_schedule(best\_schedule\_base)

    print("Лучшее расписание (островная модель):")

    visualize\_schedule(best\_schedule\_island)

    # График штрафов для базовой и островной моделей

    plt.figure(figsize=(12, 6))

    plt.plot(penalties\_base, label="Базовая модель", color="blue")

    plt.plot(penalties\_island, label="Островная модель", color="green")

    plt.xlabel("Поколение")

    plt.ylabel("Штрафы")

    plt.title("Штрафы для базовой и островной моделей в зависимости от поколения")

    plt.legend()

    plt.grid(True)

    plt.show()

# График: Штрафы для островной модели в зависимости от числа островов и поколения

def plot\_island\_penalties():

    num\_islands\_list = [1, 2, 3, 4, 5]

    num\_generations = 100

    plt.figure(figsize=(12, 6))

    for num\_islands in num\_islands\_list:

        start\_time = time.time()

        \_, penalties = island\_model(num\_generations=num\_generations, num\_islands=num\_islands)

        end\_time = time.time()

        execution\_time = end\_time - start\_time

        total\_penalty = penalties[-1]

        print(f"Число островов: {num\_islands}")

        print(f"Время исполнения (островная модель): {execution\_time} секунд(-ы)")

        print(f"Всего штрафов (островная модель): {total\_penalty}")

        print()

        plt.plot(penalties, label=f"{num\_islands} островов")

    plt.xlabel("Поколение")

    plt.ylabel("Штрафы")

    plt.title("Штрафы для островной модели в зависимости от числа островов и поколения")

    plt.legend()

    plt.grid(True)

    plt.show()

# Запуск экспериментов

run\_experiments()

plot\_island\_penalties()