МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

ИТОГОВАЯ АТТЕСТАЦИОННАЯ РАБОТА

по дополнительной профессиональной программе профессиональной переподготовки

«Организация процесса разработки компьютерного программного обеспечения»

На тему: «Применение алгоритмов динамического программирования для решения практических задач»

Выполнили:

№	Фамилия, Имя, Отчество	Группа по ООП	Группа по ДПП ПП	Подпись
1	Своеволин Иван Сергеевич	М8О-303Б-21		
2	Сайфуллин Ильдар Камилович	М8О-303Б-21		
3	Трофимов Владислав Олегович	М8О-303Б-21		
4	Кудрявцев Андрей Георгиевич	М8О-303Б-21		
5	Сабурова Серафима Павловна	М8О-303Б-21		

Руководитель итоговой аттестационной работы:

Кандидат физико-математических наук, доцент кафедры 805

Подпись Е.А.Пегачкова

Рецензент итоговой аттестационной работы:

Кандидат технических наук, доцент кафедры 806

Подпись М.Б.Булакина

Присваиваемая квалификация

«Программист»

Москва 2023

СПИСОК ИСПОЛНИТЕЛЕЙ:

№	Фамилия, Имя, Отчество (полностью)	Название и номер раздела
1	Своеволин Иван Сергеевич	2.2 Задача о замене оборудования 3.2 Задача о замене оборудования
2	Сайфуллин Ильдар Камилович	2.5 Задача о распределении ресурсов 3.5 Задача о распределении ресурсов
3	Трофимов Владислав Олегович	2.3 Задача о рюкзаке 2.3 Задача о рюкзаке
4	Кудрявцев Андрей Георгиевич	2.4 Задача о распределении инвестиций 3.4 Задача о распределении инвестиций
5	Сабурова Серафима Павловна	1.Анализ и постановка задачи разработки IT-решения: направление и цели проекта 2.1 Задача об управлении гибридной системой 3.1 Задача об управлении гибридной системой

РЕФЕРАТ

Итоговая аттестационная работа состоит из 131 страниц, 18 рисунков, 20 таблиц, 37 использованных источников, четырех приложений.

БЫСТРОДЕЙСТВИЕ ГИБРИДНОЙ СИСТЕМЫ, НОСИТЕЛЬ, СИСТЕМА УПРАВЛЕНИЯ, НЕГЛАДКИЙ АНАЛИЗ, ОПТИМАЛЬНОЕ УПРАВЛЕНИЕ, ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ, РАСПРЕДЕЛЕНИЕ ИНСВЕСТИЦИЙ, РАСПРЕДЕЛЕНИЕ ОБОРУДОВАНИЯ, ОПТИМАЛЬНАЯ ЗАМЕНА, ЗАМЕНА ОБОРУДОВАНИЯ.

Итоговая аттестационная работа выполнена в формате IT-проекта на тему «Применение алгоритмов динамического программирования для решения практических задач» и сконцентрирована на достижении точечных целей группой подвижных объектов за наименьшее время.

Объектом разработки в данной работе является программный комплекс, позволяющий решить ряд оптимизационных задач.

Цель работы – решить ряд логистических и оптимизационных задач численными методами ввиду невозможности применения гладкого анализа.

Для достижения поставленной цели были написаны программы, позволяющие решить поставленные задачи.

Основными результатами работы, полученными в процессе разработки, являются программы.

Работа выполнялась проектной командой в соответствии с методологией проектного управления в IT-индустрии.

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей итоговой аттестационной работе применяют следующие термины с соответствующими определениями:

Гибридные системы — математические модели систем управления, в которых непрерывная динамика, порождаемая в каждый момент времени одной из априорно заданного набора непрерывных систем, перемежается с дискретными операциями, подающими команды либо на мгновенное переключение с одной системы на другую, либо на мгновенную перестройку

с заданных текущих координат на другие координаты, либо на то и другое одновременно. Гибридная динамика системы заключается в альтернативной комбинации непрерывной динамики с дискретной. Непрерывная и дискретная составляющие системы могут включать некоторые параметры, влияющие на поведение системы

Задача группового управления — задача определения и оперативной оптимизации ресурсов группы, обеспечивающих реализацию действий членов группы, необходимых для достижения групповой цели

Оптимальное управление — это задача проектирования системы, обеспечивающей для заданного объекта управления или процесса закон управления или управляющую последовательность воздействий, обеспечивающих максимум или минимум заданной совокупности критериев качества системы

Параметрическая оптимизация — изменение параметров объекта, и нахождение экстремума целевой функции, зависящей от этих параметров, при заданных ограничениях

ВВЕДЕНИЕ

Проект актуален ввиду существования множества практических задач, которые могут быть решены с использованием алгоритмов динамического программирования. К таким задачам относятся оптимизационные задачи, задачи нахождения путей наибольшей длины, задачи распределения ресурсов, задачи планирования и многое другое. Применение алгоритмов динамического программирования позволяет решать такие задачи с минимальными затратами по времени и памяти.

Одним из основных преимуществ использования алгоритмов динамического программирования является возможность разбиения сложной задачи на более простые подзадачи и решение каждой из них независимо. Затем полученные решения объединяются для получения окончательного ответа. Такой подход значительно упрощает и ускоряет процесс решения сложных задач.

Еще одним преимуществом алгоритмов динамического программирования является возможность сохранения результатов уже решенных подзадач и их повторное использование при решении новых задач. Это позволяет значительно сократить количество операций и выполнить вычисления более эффективно.

Объектом разработки является программная реализация алгоритмов решения оптимизационных задач.

Предметом исследования в работе являются задачи:

1. задача о быстродействии групп управляемых объектов

переменного состава;

2. задача о загрузке транспортного средства;
3. задача о замене оборудования;
4. задача о распределении ресурсов;

5. задача о распределении инвестиций.

Методическую основу работы составляют теория оптимального управления, оптимизация, дифференциальные уравнения, паттерны динамического программирования.

Стек технологий, используемых для разработки данной итоговой аттестационной работы, включает следующие компоненты:

1. python;
2. pycharm;
3. matplotlib.

В результате разработки итоговой аттестационной работы получены следующие результаты:

1. решена задача быстродействия гибридной системы;
2. решена задача о загрузке транспортного средства;
3. решена задача о замене оборудования;
4. решена задача о распределении ресурсов;
5. решена задача о распределении инвестиций.

В результате была достигнута цель проекта – создан программный комплекс, позволяющий решить ряд оптимизационных задач.

Аналогов алгоритмов не найдено.

1 АНАЛИЗ И ПОСТАНОВКА ЗАДАЧИ РАЗРАБОТКИ IT-РЕШЕНИЯ: НАПРАВЛЕНИЕ И ЦЕЛИ ПРОЕКТА

1.1 Актуальность разработки темы и выявление потребностей в IT-решении

Оптимизационные задачи возникают повсюду — от логистики до производства, от финансов до транспорта. Применение методов динамического программирования позволяет разбить задачу на меньшие подзадачи и итерационно решать их, чтобы найти оптимальное решение. Таким образом, процесс производства становится более оптимизированным и эффективным.

Примером является задача маршрутизации транспорта. Здесь требуется определить наилучший маршрут, который обеспечит максимальную эффективность доставки груза или пассажиров. Методы динамического программирования позволяют рассматривать различные варианты и условия маршрутов, учитывать преграды, дорожные условия и время, чтобы найти оптимальное решение и сэкономить время и ресурсы.

Использование алгоритмов динамического программирования для решения задач оптимизации позволяет значительно сократить время и усилия, затрачиваемые на поиск наилучшего решения. Они также позволяют учитывать множество факторов и ограничений, которые обычно сопутствуют оптимизационным задачам.

Одним из основных преимуществ использования алгоритмов динамического программирования является возможность разбиения сложной задачи на более простые подзадачи и решение каждой из них независимо. Затем полученные решения объединяются для получения окончательного ответа. Такой подход значительно упрощает и ускоряет процесс решения сложных задач.

Еще одним преимуществом алгоритмов динамического программирования является возможность сохранения результатов уже решенных подзадач и их повторное использование при решении новых задач. Это позволяет значительно сократить количество операций и выполнить вычисления более эффективно.

Проект по применению алгоритмов динамического программирования имеет большой потенциал для практической применимости, так как подобные алгоритмы находят широкое применение во многих областях. Они могут быть использованы для планирования логистики.

На рисунке 1 представлено, что в активном пользовании нет запросов, связанных с программной реализацией алгоритма решения задачи быстродействия гибридной системы.

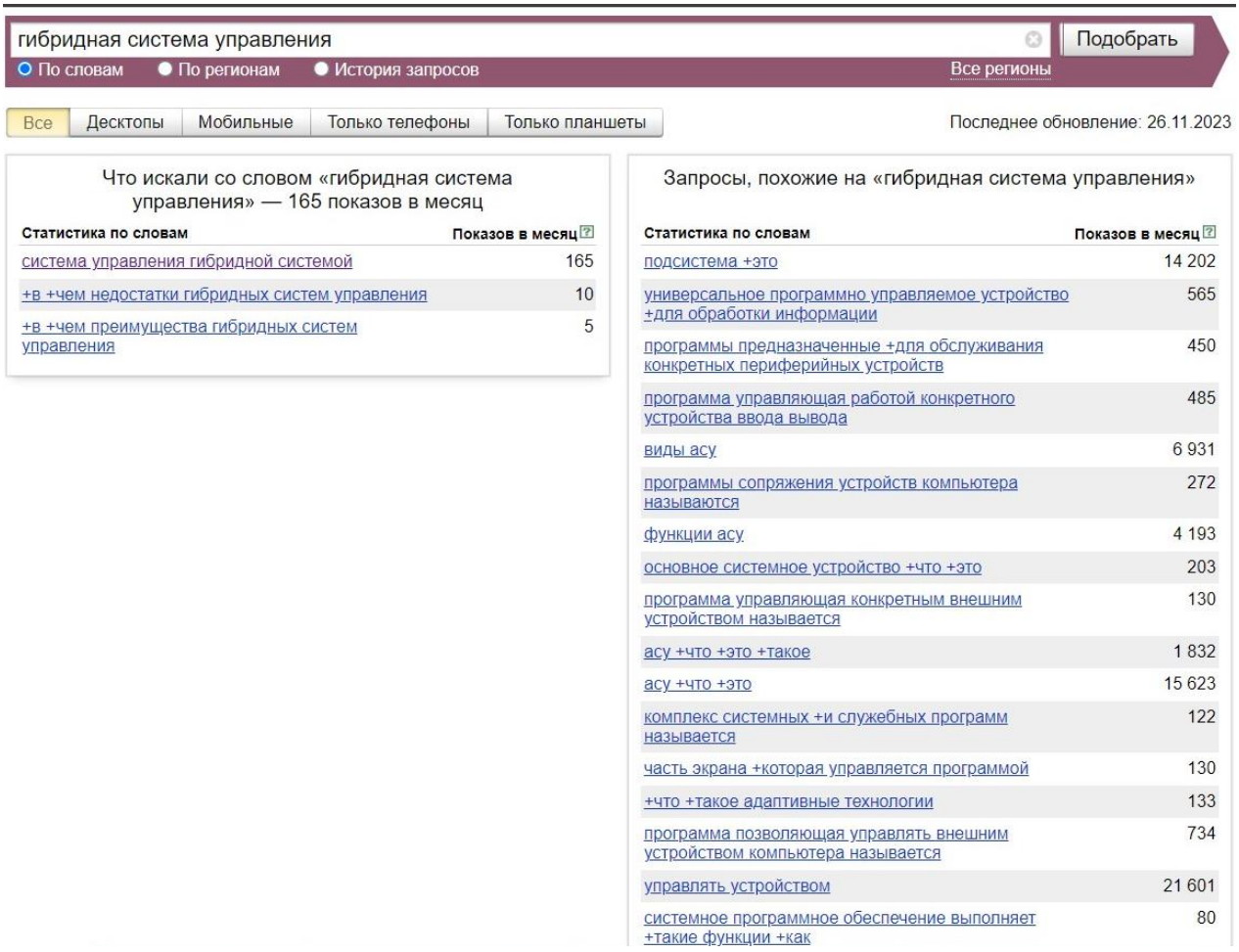

Рисунок 1 – Результаты запроса в системе Яндекс по запросу

“гибридная система управления”

На рисунке 2 представлено, что в активном пользовании нет запросов, связанных с программной реализацией алгоритма оптимальной замены оборудования.

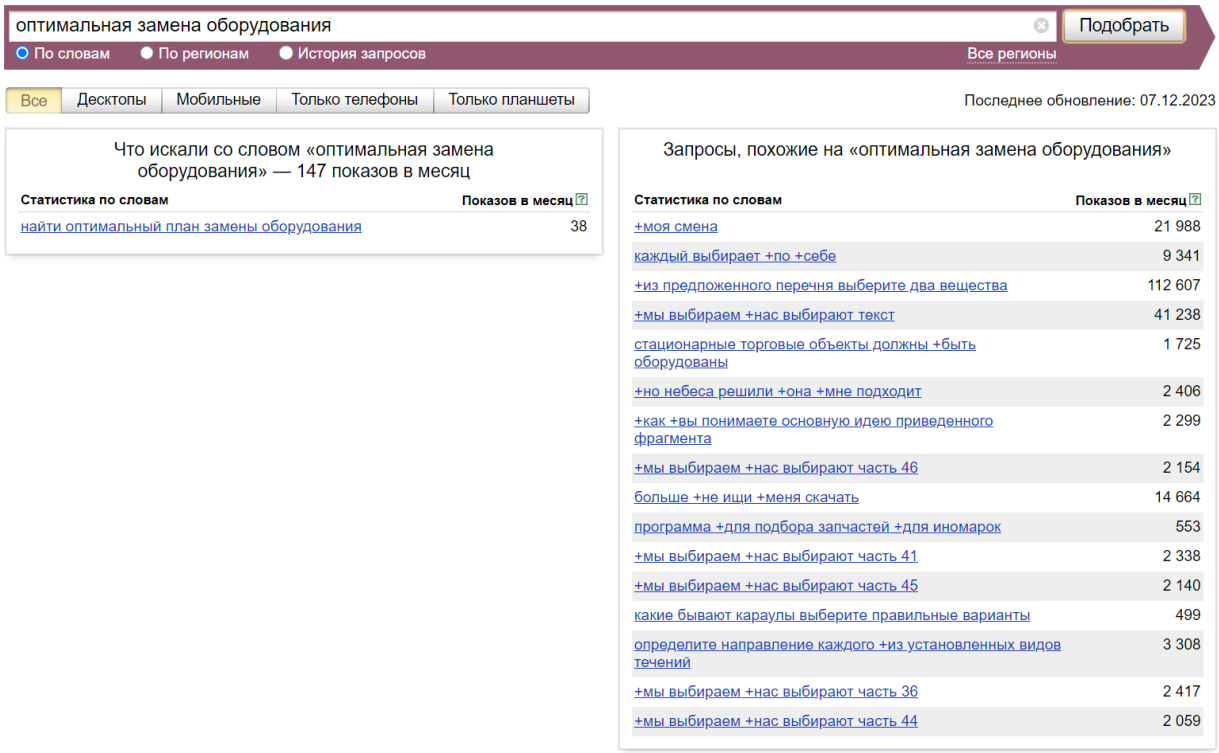

Рисунок 2 – Результаты запроса в системе Яндекс по запросу “оптимальная замена оборудования”

1.2 Критический обзор существующих подходов

На сегодняшний день решение оптимизационных задач возможно многими методами. От традиционных оптимизационных алгоритмов до передовых методов искусственного интеллекта и эволюционных стратегий — выбор инструментов огромен, что предоставляет широкие перспективы для научных исследований.

Многообразие методов включает в себя классические подходы к оптимизации, основанные на математических принципах, а также инновационные методы, разрабатываемые в области искусственного

интеллекта. Это обеспечивает исследователям возможность выбора подхода в зависимости от специфики задачи и требуемого уровня решения.

Данное разнообразие открывает направление для дальнейших исследований, целью которых является углубленное понимание эффективности различных методов и их применимости в конкретных сценариях. Таким образом, развитие области включает в себя не только поиск новых методов, но и их адаптацию под конкретные потребности, что способствует динамичному прогрессу в решении сложных задач.

1.3 Основание и техническое задание: цели, задачи и технические требования

Проблема проекта в основе итоговой аттестационной работы: потребность в решении ряда логистических задач, нуждающихся в оптимальном решении.

Обоснование: Удовлетворение потребности позволит прийти к таковым результатам:

1. минимизация затрат;

2. улучшение планирования;

3. максимизация эффективности использования ресурсов;

4. улучшение сервиса и удовлетворенности клиентов;

5. устойчивость к изменениям и готовность к риску.

В целом, решение ряда логистических оптимизационных задач является необходимым для повышения эффективности, снижения затрат, улучшения качества обслуживания клиентов и повышения конкурентоспособности компании на рынке. Также это позволяет быть более устойчивым к изменениям внешней среды и готовым к риску.

Цель проекта — решить логистические и оптимизационные задачи

численными методами ввиду невозможности применения гладкого анализа.

Техническое задание:

1. разработать модель решения задач оптимизации;
2. описать алгоритм решения задач оптимизации;
3. определить архитектуру решения;
4. определить язык программирования и утилиты для работы;
5. написать программный код;
6. провести тестирование;
7. оценить результаты работы;
8. подготовить документацию.

1.4 Выводы по разделу 1

В результате работы над задачей получены следующие выводам:

1. исходя из анализа актуальности проблемы можно резюмировать, необходимость решения поставленной задачи;

2. существует различные методы решения задачи;

3. необходимо подобрать оптимальный метод решения задачи в данной постановке – не гладкая численная конечномерная оптимизация.

2 РАЗРАБОТКА И РЕАЛИЗАЦИЯ IT-РЕШЕНИЯ: АРХИТЕКТУРА, ФУНКЦИОНАЛЬНОСТЬ И ОСОБЕННОСТИ

2.1 Задача об управлении гибридной системой

2.1.1 Теоретический анализ и модель решения задачи: Алгоритмы и подходы

Постановка задачи:

Пусть на промежутке времени [0, s] прямолинейное движение носителя описывается уравнениями:

$$\dot{x}(t) = Vcos\gamma, \dot{y}(t) = Vsin\gamma \quad (1)$$

Здесь x , y – координаты положения объекта управления на плоскости, γ – угол направления движения, отсчитываемый от положительного направления оси абсцисс, V – линейная. Управление осуществляется выбором угла направления γ .

Начальное состояние носителя задано:

$$x(0) = x^0, y(0) = y^0, \gamma(0) = \gamma^0 \quad (2)$$

Конечное состояние определяется попаданием в точку разделения:

$$S(x_s, y_s), x(s) = x_s, y(s) = y_s \quad (3)$$

В момент времени s происходит отделение от носителя m отделяемых объектов, движение которых описывается уравнениями:

$$\dot{x}_i(t) = v_i cos\gamma_i(t), \dot{y}_i(t) = v_i sin\gamma_i(t), i = 1, \dots, m \quad (4)$$

Здесь x_i, y_i, γ_i– плоские координаты и угол направления движения i -го объекта, v_i – его линейная скорость. Управление каждым объектом осуществляется выбором угла направленияγ_i.

Начальное состояние i -го отделяемого объекта определяются конечным состоянием носителя:

$$x_i(s) = x_i(s), y_i(s) = y(s), \gamma_i(s) = \gamma(s) \quad (6)$$

а конечное состояние объекта – заданной терминальной точкой

$$F_i(x_{F_i}, y_{F_i}), x_{i(T_i)} = x_{F_i}, y_{i(T_i)} = y_{F_i} \quad (7)$$

Движение отделяемого объекта заканчивается в момент T_i достижения цели.

Качество управления оценивается временем T достижения всех целей:

$$T = maxT_i, i = 1, \dots, m \quad (8)$$

Требуется найти наименьшее значение Tmin функционала (7) и оптимальный процесс, на котором это время достигается, т.е. решить задачу группового быстродействия:

$$maxT_i \to min, i = 1, \dots, m \quad (9)$$

олагаем, что максимальная скорость носителя больше максимальных скоростей отделяемых объектов, т.е. $V > v_i, i = 1, \dots, m$. В противном случае оптимальное разделение было бы в начальный момент времени.

Первоначально необходимо решить, какие функции необходимы для решения данной задачи, они описаны в таблице 1.

Таблица 1 – Название функции и её смысл

Имя функции	Смысл
zero_arr()	Создает массив с нулевыми значениями, размерность которого определяется глобальной переменной "n", и возвращает этот массив.
inf_arr()	Создает массив, заполненный значениями "math.inf" (бесконечность), также с размерностью, определяемой глобальной переменной "n", и возвращает этот массив.

Продолжение таблицы 1

Имя функции	Смысл
find_max()	Осуществляет поиск наибольших значений в массиве "points". Для этого происходит итерация по элементам массива, и с помощью условных операторов сравниваются значения каждого элемента с текущими максимальными значениями "max1" и "max2" соответственно. В итоге функция возвращает наибольшие значения "max1" и "max2".
find_min()	Аналогично функции "find_max()" выполняет поиск наименьших значений в массиве "points". Она проходит по каждому элементу массива и сравнивает их с текущими минимальными значениями "min1" и "min2". После завершения итерации функция возвращает наименьшие значения "min1" и "min2".
find_max_time()	Функция "find_max_time()" находит индекс элемента в массиве "time" с максимальным значением и возвращает два индекса: "k1" – индекс элемента с наибольшим значением, и "k2" – индекс предыдущего элемента с наибольшим значением. Функция итерируется по каждому элементу массива "time" и сравнивает его со значением "max_t". Если текущий элемент больше или равен "max_t", то происходит обновление переменных "max_t", "k2" и "k1". В итоге функция возвращает "k1" и "k2".

Продолжение таблицы 1

rasst(x, y, x1, y1)	Вычисляет расстояние между двумя точками в двумерном пространстве с помощью формулы для нахождения расстояния между двумя точками в прямоугольной системе координат. Функция принимает координаты двух точек (x, y) и (x1, y1) и возвращает вычисленное расстояние.
top(v1, v2)	Вычисляет скалярное произведение двух векторов "v1" и "v2". Произведение скалярных компонент векторов складывается, и результат возвращается функцией.
down(v1, v2)	Вычисляет произведение длин двух векторов "v1" и "v2". Длины векторов определяются с помощью формулы для вычисления модуля вектора в прямоугольной системе координат. Результатом функции является произведение длин векторов.
peny1	Принимает на вход точку и вычисляет угол между вектором, состоящим из координат точки и точки-носителя, и единичным вектором, который образуется из значения глобальной переменной "gamma". Затем функция возвращает значение данного угла.
peny2	Принимает на вход две точки: точку разделения и точку цели. Функция вычисляет угол между вектором, состоящим из координат точки разделения и точки-носителя, и вектором, который образуется из координат точки цели и точки разделения. Затем функция возвращает значение данного угла.

Продолжение таблицы 1

optimize1	Оптимизирует процесс поиска оптимальной точки на графе. Функция использует циклы, чтобы перебрать все возможные комбинации координат точек на графе. Затем для каждой комбинации координат вычисляется время, необходимое для достижения данной точки, с учетом всех ограничений и коэффициентов. Функция возвращает оптимальное время и координаты точки.
optimize2	Также осуществляет оптимизацию процесса поиска оптимальной точки на графе. Функция также использует циклы, чтобы перебрать все возможные комбинации координат точек на графе. Однако здесь используется другой подход к оптимизации. Функция перебирает координаты точки начала и в зависимости от данной точки вычисляет время, необходимое для достижения данной точки, с учетом всех ограничений и коэффициентов. Функция возвращает оптимальное время, координаты точки и значение прямоугольника, полученное из функции "peny1".

Ключевыми являются функции optimize2, optimize1, peny2, peny1. Все остальные функции используются как вспомогательный для глобальной задачи.

2.1.2 Выбор технологического стека: обоснование и применение

Для решения задачи будет использоваться язык Python, в том числе его библиотеки Matplotlib, Numpy, Math для упрощения процесса реализации алгоритма. Этот язык будет использоваться ввиду его гибкости и перспективы разработки десктопного приложения для автоматизированной работы.

Используется среда Pycharm, Anaconda.

Библиотека Math необходима для реализации математического функционала в процессе решения задачи и реализации алгоритма.

Matplotlib используется для отрисовки графиков.

Numpy необходим для корректной отрисовки графиков и работы с матричными структурами.

2.1.3 Разработка программного решения

Необходимо написать код, в котором определяются несколько функций для работы с массивами и вычисления различных величин, таких как максимальное и минимальное значение в массиве, индексы точек с максимальным временем, расстояние между точками, скалярное произведение и произведение длин векторов. Этот функционал необходим для решения поставленной задачи.

Описание действия каждой функции(алгоритм):

Алгоритм zero_arr(n):

1. используется глобальная переменная n;
2. создается массив zero_array размера n, состоящий из нулей;
3. возвращается созданный массив.

Алгоритм inf_arr(n):

1. используется глобальная переменная n;
2. создается массив inf_array размера n, заполненный значениями math.inf (бесконечность);
3. возвращается созданный массив.

Алгоритм find_max():

1. инициализируем глобальную переменную points;

2. инициализируем переменные max1 и max2 со значением 0;

3. используем цикл for, чтобы пройти по значениям массива points;

4. если значение элемента points по первому индексу больше max1, обновляем max1;

5. если значение элемента points по второму индексу больше max2, обновляем max2;

6. возвращаем значения max1 и max2.

Алгоритм find_min():

1. инициализируем глобальную переменную points;

2. инициализируем переменные min1 и min2 со значением math.inf;

3. используем цикл for, чтобы пройти по значениям массива points;

4. если значение элемента points по первому индексу меньше min1, обновляем min1;

5. если значение элемента points по второму индексу меньше min2, обновляем min2;

6. возвращаем значения min1 и min2.

Алгоритм find_max_time(time):

1. инициализируем глобальные переменные n и points;

2. инициализируем переменные max_t, k1 и k2 со значениями 0;

3. используем цикл for, чтобы пройти по значениям массива time;

4. если значение элемента time больше или равно max_t, обновляем max_t и переносим предыдущие значения k1 в k2, а текущий индекс итерации в k1;

5. возвращаем значения k1 и k2.

Алгоритм rasst(x, y, x1, y1):

1. вычисляем расстояние между точками (x, y) и (x1, y1) с помощью формулы расстояния между двумя точками в двумерном пространстве;

2. возвращаем вычисленное расстояние.

Алгоритм top(v1, v2):

1. инициализируем глобальную переменную gamma;

2. вычисляем скалярное произведение векторов v1 и v2, умножая их соответствующие координаты и складывая результаты;

3. возвращаем полученное значение.

Алгоритм down(v1, v2):

1. вычисляем произведение длин векторов v1 и v2 путем вычисления квадратного корня суммы квадратов их координат и умножения результатов;

2. возвращаем полученное значение.

Алгоритм peny1(point):

1. глобальные переменные: функция использует две глобальные переменные: гамма и носитель;

2. векторные определения:

1) vect1: вектор, представленный как [cos(gamma), sin(gamma)];

2) vect2 разность входной точкой и координатами в носителе.

3. расчет угла: если компонент x или y вектора vect2 равен нулю, phi устанавливается в 0. В противном случае он вычисляет косинус угла (cos_phi) между векторами vect1 и vect2. Затем значение cos_phi корректируется, чтобы оно оставалось в диапазоне [-1, 1]. Угол phi вычисляется с помощью функции обратного косинуса (math.acos);

4. возврат значения: функция возвращает вычисленный угол phi.

Алгоритм peny2(point):

1. глобальные переменные: point1 и point2;

2. вычисляем вектор vect1 как разность между координатами точек point1 и nositel;

3. вычисляем вектор vect2 как разность между координатами точек point2 и point1;

4. если знаменатель функции down(vect1, vect2) равен 0, то присвоить phi значение 0;

5. если знаменатель функции down(vect1, vect2) не равен 0, то вычислить cos_phi как отношение числителя функции top(vect1, vect2) к знаменателю функции down(vect1, vect2);

6. если cos_phi больше 1, то присвоить cos_phi значение 1;

7. если cos_phi меньше -1, то присвоить cos_phi значение -1;

8. вычислить значение угла phi как arccos(cos_phi) с помощью функции math.acos();

9. вернуть значение phi;

Алгоритм optimize1():

1. глобальные переменные: grid_find, points, n, V, v, t;

2. создать пустой список opt_point для хранения оптимальной точки;

3. создать пустой список t_iter для хранения времени прибытия в каждую точку;

4. создать список t_t со значениями от 0 до n-1 для хранения времени прибытия в каждую точку в порядке их следования;

5. присвоить переменной max_arg значение бесконечности;

6. создать список s_now_ts для хранения расстояний от оптимальной точки до каждой точки из списка points;

7. создать список s_now_ts для хранения расстояния от оптимальной точки до носителя;

8. для каждой координаты i из диапазона grid_find[0] до grid_find[1] (начальное и конечное положение x) и для каждой координаты j из диапазона grid_find[2] до grid_find[3] (начальное и конечное положение y), выполнить следующие действия:

1) вычислить расстояние s_now_nos от оптимальной точки до носителя с помощью функции rasst();

2) для каждой точки из списка points, вычислить расстояние s_now_ts (нынешнее положение цели) от оптимальной точки до этой точки с помощью функции rasst();

3) вычислить время t_iter, необходимое для достижения носителя и каждой точки из списка points, используя формулу t_iter[step] = (s_now_nos / V) + (s_now_ts[step] / v) + peny1((i, j))*koeff1 + peny2((i,j), (points[step][0], points[step][1]))*koeff2 (сумма времени движения от старта до цели и времени, потраченного на поворот);

4) если максимальное значение в списке t_iter меньше или равно значению переменной max_arg, то:

а) присвоить max_arg значение максимального значения в списке t_iter;

б) скопировать значения из списка t_iter в список t_t;

в) Присвоить значения координат i и j переменной opt_point.

9. Вернуть список t_t и оптимальную точку opt_point.

Алгоритм optimize2:

1. глобальные переменные: points, point_razd, nositel, min_time, V, v, rect_nos_razd;

2. создать пустой список t_iter для хранения времени прибытия в

каждую точку;

3. создать пустой список t_time для хранения времени прибытия в каждую точку в порядке их следования;

4. создать список s_now_ts для хранения расстояний от оптимальной точки до каждой точки из списка pointsж;

5. создать переменную opt_point и присвоить ей значение [0, 0];

6. создать переменную point_start и присвоить ей значение [point_razd[0]-1, point_razd[1]-1];

7. присвоить переменной min_iter максимальное значение из списка min_time;

8. для каждой координаты i из диапазона от 0 до 2/d и для каждой координаты j из диапазона от 0 до 2/d, выполнить следующие действия:

1) вычислить расстояние s_now_nos от оптимальной точки до носителя с помощью функции rasst();

2) вычислить значение rect1 с помощью функции peny1() для координат (point_start[0] + i * d, point_start[1] + j * d);

3) для каждой точки из списка points, вычислить расстояние s_now_ts от оптимальной точки до этой точки с помощью функции rasst();

4) для каждой точки из списка points, вычислить значение rect2 с помощью функции peny2() для координат (point_start[0] + i*d, point_start[1] + j*d) и (points[step][0], points[step][1]);

5) вычислить время t_iter, необходимое для достижения носителя и каждой точки из списка points, используя формулу t_iter[step] = (s_now_nos / V) + (s_now_ts[step] / v) + rect1*koeff1 + rect2*koeff2;

6) если максимальное значение в списке t_iter меньше значения переменной min_iter, то:

а) присвоить min_iter значение максимального значения в списке t_iter;

б) скопировать значения из списка t_iter в список t_time;

в) присвоить координаты i * d + point_start[0] и j * d + point_start[1] переменным opt_point[0] и opt_point[1] соответственно;г) присвоить значение rect1 переменной rec.

9. вернуть список t_time, оптимальную точку opt_point и значение rec.

2.1.4 Вывод по разделу 2.1

В ходе исследования проблемы оптимального управления гибридными системами был предложен и реализован алгоритм, основанный на методах динамического программирования. Данный алгоритм позволяет с высокой эффективностью по времени определить наилучшее время для разделения носителя, минимизирую общее время движения. Решение задачи основывается на динамическом программировании, что позволяет избежать повторных вычислений и снизить вычислительную сложность задачи.

2.2 Задача о замене оборудования

2.2.1 Постановка задачи

Рассмотрим задачу по оптимизации стратегии эксплуатации оборудования. В современных условиях эффективное управление оборудованием становится критическим фактором для успешной деятельности предприятий.

Задача заключается в том, чтобы найти оптимальный момент замены текущего оборудования на новое, учитывая финансовые аспекты, такие как годовой доход, затраты на обслуживание, остаточную стоимость оборудования, цены на новое оборудования.

Условие задачи может быть сформулировано следующим образом: найти оптимальную стратегию эксплуатации оборудования на определенный

период, если даны:

1. годовой доход в зависимости от возраста;

2. ежегодные затраты на обслуживание в зависимости от возраста;

3. остаточная стоимость в зависимости от возраста;

4. стоимость нового оборудования;

5. возраст оборудования к началу эксплуатационного периода.

Таким образом, решением задачи будет являться стратегия, обеспечивающую максимальную прибыль, по сохранению или замене оборудования в каждый год планового периода.

2.2.2 Формализация задачи

2.2.2.1 Переменные

Описание переменных представлено в таблице 2

Таблица 2 – переменные в задаче

N	продолжительность работы оборудования, эксплуатационный период, лет
t	общий возраст оборудования, независимо от времени эксплуатации, лет
$r(t)$	ежегодная прибыль от использования оборудования возраста t, условных единиц

Продолжение таблицы 2

$u(t)$	ежегодная стоимость обслуживания оборудования возраста t, условных единиц
$s(t)$	остаточная стоимость оборудования возраста t, условных единиц
P	цена на новое оборудование, условных единиц
t_0	возраст оборудования к началу эксплуатационного периода, лет

2.2.2.1 Ограничения

1. $0 \leq t_0 \leq max\{t\}$ – возраст оборудования к началу эксплуатационного периода может быть равным нулю $t_0 = 0$, в таком случае оборудования новое;

2. $0 < P$ – задача имеет смысл, если цена на оборудование больше 0;

3. $0 < N \leq max\{t\}$ – задача имеет смысл, если эксплуатационный период больше 0, также пусть он меньше максимального возраста оборудования;

4. $s(t) < P$ – задача имеет смысл, если стоимость нового оборудования превосходит остаточную стоимость оборудования/

2.2.3 Алгоритм

2.2.3.1 Входные данные

В контексте решения задачи оптимизации по замене оборудования на протяжении эксплуатационного периода, предлагается применение алгоритма динамического программирования. Данный подход позволяет эффективно решать задачи оптимизации с перекрывающимися подзадачами, что является характерной особенностью данной задачи.

Использование алгоритма динамического программирования в данной задаче позволит систематизировать процесс принятия решений и найти оптимальные стратегии для замены оборудования, учитывая долгосрочные финансовые аспекты.

По условию задачи даны переменные в таблице 3, а также эксплуатационный период N и возраст оборудования к началу эксплуатационного периода t_0.

Таблица 3 – известные переменные

t	0	1	…	$n-1$	n
$r(t)$	$r(0)$	$r(1)$	…	$r(n-1)$	$r(n)$
$u(t)$	$u(0)$	$u(1)$	…	$u(n-1)$	$u(n)$
$s(t)$	$s(0)$	$s(1)$	…	$s(n-1)$	$s(n)$

2.2.3.2 Этап условной оптимизации

Уравнения Беллмана являются ключевым элементом в динамическом программировании. Они позволяют систематически рассматривать каждую

точку времени и оптимально принимать решение о замене оборудования. Каждый шаг оптимизации учитывает влияние текущего решения на последующие периоды, обеспечивая устойчивость и эффективность стратегии на протяжении всего эксплуатационного периода. Таким образом, уравнения Беллмана предоставляют мощный инструмент для решения данной задачи оптимизации.

Чтобы решить задачу, применим принцип оптимальности Беллмана. Рассмотрим интервалы (годы) планового периода в последовательности от конца к началу. Введем функцию условно-оптимальных значений функции $F_k(t)$. Эта функция показывает максимальную прибыль, получаемую от работы оборудования возраста t лет за последние $N-k+1$ лет планового периода.

Например, при $k=N$ рассматривается последний год планового периода, при $k=N-1$ – последние два года и т.д., при $k=1$ – последние N лет, т.е. весь плановый период.

На рисунке 3 представлена рассматриваемая задача замены оборудования в виде сети с указанием возможного возраста в начале каждого года работы.

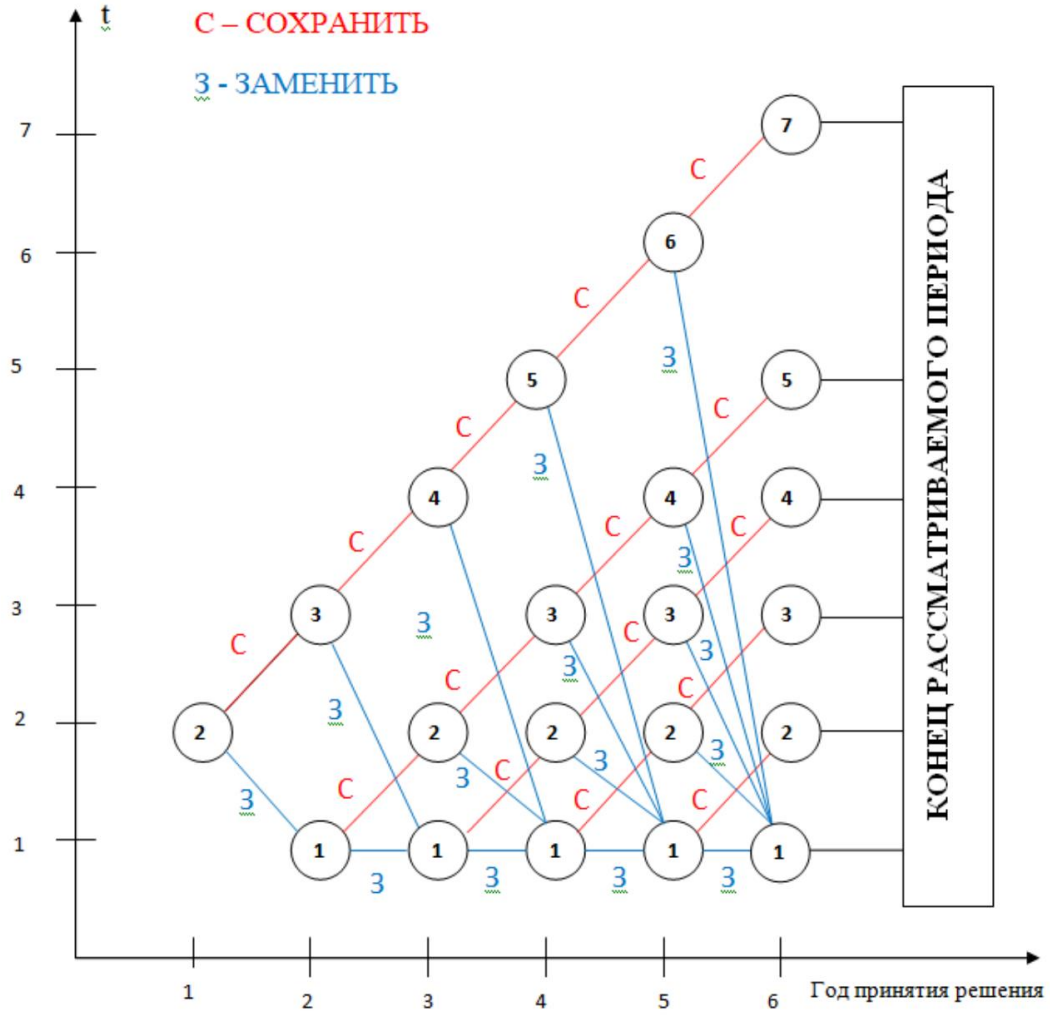

Рисунок 3 – Схема определения возраста оборудования на начало года

На каждом этапе n-этапного процесса нужно принять решение о сохранении или замене оборудования. Решение должно обеспечивать получение максимальной прибыли.

Функциональные уравнения, основанные на принципе оптимальности Беллмана, имеют вид:

$$F_k(t) = max\{r(t) - u(t), s(t) - p + r(0) - u(0)\}, k = N \qquad (10)$$

$$F_k(t) = max\{r(t) - u(t) + F_{k+1}(t+1),$$

$$s(t) - p + r(0) - u(0) + F_{k+1}(0)\}, k = 1, \dots, N-1 \qquad (11)$$

Оба уравнения состоят из двух частей:

1. первая часть определяет доход, получаемый при сохранении оборудования;

2. вторая часть — доход, получаемый при замене оборудования на новое.

В обоих уравнениях функция $r(t) - u(t)$ – есть разность между стоимостью произведенной продукции и эксплуатационными издержками на k –м этапе процесса.

Вторая часть характеризуется следующим образом: функция $s(t) - p$ представляет чистые издержки по замене оборудования возраст которого t лет.

Функция $r(0)$ выражает доход, получаемый от нового оборудования (оборудования возраста 0 лет).

Предполагается, что переход от работы на оборудовании возраста t лет к работе на новом оборудовании совершается мгновенно.

Уравнения позволяют оценить варианты замены и сохранения оборудования, с тем, чтобы принять тот из них, который предполагает больший доход. Эти соотношения дают возможность не только выбрать линию поведения при решении вопроса о сохранении или замене оборудования, но и определить прибыль, получаемую при принятии каждого из этих решений.

Если прибыль не зависит от выбора решения о сохранении или замене оборудования, то принимается решение – сохранить, т.к. это потенциально может понести убытки времени и финансов.

Таким образом, подсчитываются значения максимальных прибылей для каждого года эксплуатации для оборудования любого возможного

возраста, результаты записываются в таблицу вида таблицы 4, а также на этом этапе известно при каком выборе (сохранения/замена) было получено значение прибыли.

Таблица 4 – матрица максимальных прибылей

k	0	1	…	$n-1$	n
1	$F_1(0)$	$F_1(1)$	…	$F_1(n-1)$	$F_1(n)$
2	$F_2(0)$	$F_2(1)$	…	$F_2(n-1)$	$F_2(n)$
…	…	…	…	…	…
$N-1$	$F_{N-1}(0)$	$F_{N-1}(1)$	…	$F_{N-1}(n-1)$	$F_{N-1}(n)$
N	$F_N(0)$	$F_N(1)$	…	$F_N(n-1)$	$F_N(n)$

2.2.3.3 Этап безусловной оптимизации

Данный этап – N-шаговый процесс:

1. по таблице в пересечении года эксплуатации $k=1$ с возрастом оборудования к началу эксплуатационного периода $t=t_0$ находится значение максимальной прибыли за весь плановый период. На этом шаге в вектор U запишем выбор для этого значения функции: U=(A), где A – выбор сохранения/замены;

2. если на $i-1$ шаге была выбрана замена оборудования, то текущий возраст оборудования $t=0$, иначе t инкрементируется, далее по таблице в пересечении года эксплуатации $k=2$ с текущим возрастом оборудования находится значение максимальной прибыли за оставшийся плановый период. На этом шаге в вектор U запишем выбор для этого значения функции: U=(A, B), где B – выбор сохранения/замены.

Дальнейшие шаги повторяются аналогично второму, по окончании N шагов, вектор U будет иметь N значений – это и есть решение задачи – стратегия, обеспечивающую максимальную прибыль, по сохранению или замене оборудования в каждый год планового периода.

2.2.3.4 Пример работы алгоритма

Условие задачи: найти оптимальную стратегию эксплуатации оборудования на период $N=6$ лет, если:

1. годовой доход $r(t)$, ежегодные затраты на обслуживание $u(t)$ и остаточная стоимость $s(t)$ в зависимости от возраста заданы в таблице 5;

Таблица 5 – условия задачи

t	0	1	2	3	4	5	6
$r(t)$	8	7	7	6	6	5	5
$u(t)$	1	2	1	2	2	3	2
$s(t)$	12	10	8	8	7	6	4

2. стоимость нового оборудования равна $P=16$;

3. возраст оборудования к началу эксплуатационного периода t_0 составлял 1 год.

Решение:

Для упрощения расчётов вычтем из годового дохода $r(t)$ ежегодные траты на обслуживание $u(t)$ и получим чистый доход $r'(t)$ в таблице 6.

Таблица 6 – чистый годовой доход $r'(t)$

$r'(t)$	7	5	6	4	4	2	3

I этап. Условная оптимизация.

1-й шаг: По условию $N = 6, N - k + 1 = 1 = k = 6$.

$$F_k(t) = max\{r'(t), s(t) - p + r'(0)\}, k = N \quad (12)$$

$$F_6(0) = max\{7{,}12 - 16 + 7\} = 7(C)$$

$$F_6(1) = max\{5{,}10 - 16 + 7\} = 5(C)$$

$$F_6(2) = max\{6{,}8 - 16 + 7\} = 6(C)$$

$$F_6(3) = max\{4{,}8 - 16 + 7\} = 4(C)$$

$$F_6(4) = max\{4{,}7 - 16 + 7\} = 4(C)$$

$$F_6(5) = max\{2{,}6 - 16 + 7\} = 2(C)$$

$$F_6(6) = max\{3{,}4 - 16 + 7\} = 3(C)$$

2-й шаг: По условию $N = 6, N - k + 1 = 2 = k = 5$.

$$F_k(t) = max\{r'(t) + F_{k+1}(t+1), s(t) - p + r'(0) + F_{k+1}(0)\}, \text{при} k = 1, \dots, N-1 \quad (13)$$

$$F_5(0) = max\{7 + 5{,}12 - 16 + 7 + 7\} = 12(C)$$

$$F_5(1) = max\{5 + 6{,}10 - 16 + 7 + 7\} = 11(C)$$

$$F_5(2) = max\{6 + 4{,}8 - 16 + 7 + 7\} = 10(C)$$

$$F_5(3) = max\{4 + 4{,}8 - 16 + 7 + 7\} = 8(C)$$

$$F_5(4) = max\{4 + 2{,}7 - 16 + 7 + 7\} = 6(C)$$

$$F_5(5) = max\{2 + 3{,}6 - 16 + 7 + 7\} = 5(C)$$

$$F_5(6) = max\{3 + 0{,}4 - 16 + 7 + 7\} = 2(3)$$

3-й шаг: По условию $N = 6, N - k + 1 = 3 = k = 4$.

$$F_4(0) = max\{7 + 11{,}12 - 16 + 7 + 12\} = 18(C)$$

$$F_4(1) = max\{5 + 10{,}10 - 16 + 7 + 12\} = 15(C)$$

$$F_4(2) = max\{6 + 8{,}8 - 16 + 7 + 12\} = 14(C)$$

$$F_4(3) = max\{4 + 6{,}8 - 16 + 7 + 12\} = 11(3)$$

$$F_4(4) = max\{4 + 5{,}7 - 16 + 7 + 12\} = 10(3)$$

$$F_4(5) = max\{2 + 2{,}6 - 16 + 7 + 12\} = 9(3)$$

$$F_4(6) = max\{3 + 0{,}4 - 16 + 7 + 12\} = 7(3) \quad (14)$$

4-й шаг: По условию $N = 6, N - k + 1 = 4 = k = 3$.

$$F_3(0) = max\{7 + 15{,}12 - 16 + 7 + 18\} = 22(C)$$

$$F_3(1) = max\{5 + 14{,}10 - 16 + 7 + 18\} = 19(C)$$

$$F_3(2) = max\{6 + 11{,}8 - 16 + 7 + 18\} = 17(C)$$

$$F_3(3) = max\{4 + 10{,}8 - 16 + 7 + 18\} = 17(3)$$

$$F_3(4) = max\{4 + 9{,}7 - 16 + 7 + 18\} = 16(3)$$

$$F_3(5) = max\{2 + 7{,}6 - 16 + 7 + 18\} = 15(3)$$

$$F_3(6) = max\{3 + 0{,}4 - 16 + 7 + 18\} = 13(3) \quad (15)$$

5-й шаг: По условию $N = 6, N - k + 1 = 4 = k = 2$.

$$F_2(0) = max\{7 + 19{,}12 - 16 + 7 + 22\} = 26(C)$$

$$F_2(1) = max\{5 + 17{,}10 - 16 + 7 + 22\} = 23(3)$$

$$F_2(2) = max\{6 + 17{,}8 - 16 + 7 + 22\} = 23(3)$$

$$F_2(3) = max\{4 + 16{,}8 - 16 + 7 + 22\} = 21(3)$$

$$F_2(4) = max\{4 + 15{,}7 - 16 + 7 + 22\} = 20(3)$$

$$F_2(5) = max\{2 + 13{,}6 - 16 + 7 + 22\} = 19(3)$$

$$F_2(6) = max\{3 + 0{,}4 - 16 + 7 + 22\} = 17(3) \quad (16)$$

6-й шаг: По условию $N = 6, N - k + 1 = 4 = k = 1$.

$$F_1(0) = max\{7 + 23{,}12 - 16 + 7 + 26\} = 30(C)$$

$$F_1(1) = max\{5 + 23{,}10 - 16 + 7 + 26\} = 28(\text{C})$$

$$F_1(2) = max\{6 + 21{,}8 - 16 + 7 + 26\} = 27(\text{C})$$

$$F_1(3) = max\{4 + 20{,}8 - 16 + 7 + 26\} = 25(3)$$

$$F_1(4) = max\{4 + 19{,}7 - 16 + 7 + 26\} = 24(3)$$

$$F_1(5) = max\{2 + 17{,}6 - 16 + 7 + 26\} = 23(3)$$

$$F_1(6) = max\{3 + 0{,}4 - 16 + 7 + 26\} = 21(3) \quad (17)$$

Таким образом, имеем таблицу 7 максимальных прибылей.

Таблица 7 – максимальные прибыли

k	0	1	2	3	4	5	6
1	30 (С)	28 (С)	27 (С)	25 (З)	24 (З)	23 (З)	21 (З)
2	26 (С)	23 (З)	23 (С)	21 (З)	20 (З)	19 (З)	17 (З)

3	22 (С)	19 (С)	17 (С)	17 (З)	16 (З)	15 (З)	13 (З)
4	18 (С)	15 (С)	14 (С)	11 (З)	10 (З)	9 (З)	7 (З)
5	12 (С)	11 (С)	10 (С)	8 (С)	6 (С)	5 (С)	2 (З)
6	7 (С)	5 (С)	6 (С)	4 (С)	4 (С)	2 (С)	3 (С)

II этап. Безусловная оптимизация.

1. $k = 1, t = t_0 = 1$ = Прибыль составит 28 условных единиц, оптимальный выбор – сохранить оборудование, U = (С);

2. $k = 2, t = 2$ = Прибыль составит 23 условных единиц, оптимальный выбор – сохранить оборудование, U = (С, С);

3. $k = 3, t = 3$ = Прибыль составит 17 условных единиц, оптимальный выбор – заменить оборудование, U = (С, С, З);

4. $k = 4, t = 0$ = Прибыль составит 18 условных единиц, оптимальный выбор – сохранить оборудование, U = (С, С, З, С);

5. $k = 5, t = 1$ = Прибыль составит 11 условных единиц, оптимальный выбор – сохранить оборудование, U = (С, С, З, С, С);

6. $k = 6, t = 2$ = Прибыль составит 6 условных единиц, оптимальный выбор – сохранить оборудование, U = (С, С, З, С, С, С).

Таким образом, за 6 лет эксплуатации оборудования замену надо произвести в начале 3-го года эксплуатации.

2.2.3.5 Вывод по алгоритму

В ходе исследования проблемы замены оборудования на протяжении длительного эксплуатационного периода был предложен и реализован алгоритм, основанный на методах динамического программирования. Данная стратегия оптимизации предоставляет уникальный инструмент для принятия решений в условиях долгосрочной перспективы, учитывая годовой доход, ежегодные затраты на обслуживание и остаточную стоимость оборудования в зависимости от его возраста.

Данный алгоритм позволяет с эффективностью $O(N * max\{t\})$ по времени определить оптимальные моменты для замены оборудования, максимизируя чистую прибыль на протяжении всего периода эксплуатации. Решение задачи основывается на динамическом программировании, что позволяет избежать повторных вычислений и снизить вычислительную сложность задачи.

Использование данного алгоритма позволяет предприятиям принимать обоснованные решения по замене и обновлению оборудования, что в свою очередь способствует оптимизации экономических показателей и повышению эффективности бизнес-процессов. Такой подход особенно ценен в условиях быстро меняющейся технологической среды, где правильные стратегии обновления оборудования могут существенно повлиять на конкурентоспособность предприятия.

2.2.4 Реализация алгоритма

2.2.4.1 Выбор технологического стека

Для реализации алгоритма будет использоваться Python 3.11 ввиду следующих причин:

1. простота и читаемость кода

Python известен своей простотой и читаемостью, что облегчает разработку и понимание кода. Алгоритм является нетривиальным, и использование Python упрощает его визуальное восприятие и анализ.

2. поддержка сообществом

Python имеет активное сообщество разработчиков, что обеспечивает поддержку и доступ к множеству ресурсов. Возможность быстро получить ответы на вопросы и решить проблемы является важным аспектом выбора языка.

3. высокоуровневые абстракции

Python предоставляет высокоуровневые абстракции, что позволяет более просто выражать сложные концепции. Это может быть важно для четкого представления и реализации математических выражений в алгоритме.

Дополнительные инструменты и библиотеки использованы не будут, т.к. в них нет необходимости.

В качестве среды разработки выбран Pycharm.

2.2.4.2 Функциональная спецификация

Оптимизация замены оборудования с применением динамического программирования на Python:

1. описание задачи

Задача состоит в разработке алгоритма оптимизации стратегии замены оборудования с использованием методов динамического программирования. Этот алгоритм принимает входные данные, такие как годовой доход, ежегодные затраты на обслуживание, остаточная стоимость в зависимости от

возраста оборудования, стоимость нового оборудования и начальный возраст оборудования.

2. входные параметры

В программе будет необходимо получить и обработать следующие данные:

1) r – список годовых доходов от оборудования;

2) u – список ежегодных затрат на обслуживание оборудования;

3) s – список остаточных стоимостей оборудования в зависимости от его возраста;

4) p – стоимость нового оборудования;

5) t_0 – возраст оборудования на момент начала анализа;

6) planned_period – продолжительность эксплуатационного периода.

3. выходные данные

Результат программы должен включать в себя следующие необходимые данные:

1) profit_matrix – матрица максимальных прибылей, содержащая информацию о максимальных прибылях и выбранных стратегиях для каждого возможного состояния (года и возраста оборудования);

2) optimal_strategy – список, представляющий оптимальную стратегию замены/сохранения оборудования на каждый год анализируемого периода.

4. этапы решения:

1. условная оптимизация (_conditional_optimization):

1) создание матрицы для хранения максимальных прибылей и выбора стратегий для каждого возможного состояния;

2) итеративный процесс вычисления максимальных прибылей с учетом выбора замены/сохранения оборудования.

2. безусловная оптимизация (_unconditional_optimization):

1) сохранение выбора сохранения/замены оборудования для каждого года.

3. основная функция (optimal_replacement):

1) инициализация переменных и подготовка входных данных;

2) вызов условной и безусловной оптимизации;

3) возврат результатов в виде матрицы максимальных прибылей и оптимальной стратегии.

5. пример использования:

Должен быть предоставлен пример использования алгоритма с конкретными данными и выводом, что облегчает понимание и демонстрирует работу алгоритма.

6. комментарии и пояснения:

В коде должны быть предоставлены комментарии, объясняющие каждый шаг алгоритма и выводы для обеспечения ясности и понимания.

2.2.4.3 Программная реализация

Разработка будет вестись согласно этапам функциональной спецификации, но уже с включенными комментариями и пояснениями.

В приложении Г прикреплены и прокомментированы 3 функции:

1) этап условной оптимизации

2) этап безусловной оптимизации

3) основная функция

2.2.4.4 Тестирование, пример

Тестирование программы будет проводиться на тех же данных, которые были использованы для примера работы алгоритма в пункте 2.2.3.4

Код:

```
r = [8, 7, 7, 6, 6, 5, 5] # стоимость продукции, произведенной в течение каждого года с помощью этого оборудования

u = [1, 2, 1, 2, 2, 3, 2] # ежегодные затраты, связанные с эксплуатацией оборудования

s = [12, 10, 8, 8, 7, 6, 4] # остаточная стоимость оборудования

p = 16 # стоимость нового оборудования

t_0 = 1 # возраст оборудования на момент анализа

planned_period = 6 # продолжительность работы оборудования
```

Вызывается функция, вычисляющая оптимальную стратегию замены оборудования, возвращаемые объекты матрицы максимальных прибылей и списка оптимального управления оборудованием сохраняются

Код:

```
profit_matrix, optimal_strategy = optimal_replacement(r=r, u=u, s=s, p=p, t_0=t_0, period=planned_period)
```

Распечатывается матрица максимальных прибылей

Код:

```
print("\n Матрица максимальных прибылей:")

for row in profit_matrix:

    print(row)
```

Распечатывается список оптимального управления оборудованием

Код:

```
print("\n Оптимальная стратегия:")
print(*list(f'|{i + 1} год: {"Сохранить" if x[1] == "C" else "Заменить"}|' for i, x in enumerate(optimal_strategy)),
    sep=' -> ')
```

Вывод программы

Условная оптимизация:

F_6(0) = max(7, 7 + 12 - 16) = 7 (C)

F_6(1) = max(5, 7 + 10 - 16) = 5 (C)

F_6(2) = max(6, 7 + 8 - 16) = 6 (C)

F_6(3) = max(4, 7 + 8 - 16) = 4 (C)

F_6(4) = max(4, 7 + 7 - 16) = 4 (C)

F_6(5) = max(2, 7 + 6 - 16) = 2 (C)

F_6(6) = max(3, 7 + 4 - 16) = 3 (C)

F_5(0) = max(7 + 5, 12 - 16 + 7 + 7) = 12 (C)

F_5(1) = max(5 + 6, 10 - 16 + 7 + 7) = 11 (C)

F_5(2) = max(6 + 4, 8 - 16 + 7 + 7) = 10 (C)

F_5(3) = max(4 + 4, 8 - 16 + 7 + 7) = 8 (C)

F_5(4) = max(4 + 2, 7 - 16 + 7 + 7) = 6 (C)

F_5(5) = max(2 + 3, 6 - 16 + 7 + 7) = 5 (C)

F_5(6) = max(3 + 0, 4 - 16 + 7 + 7) = 2 (3)

F_4(0) = max(7 + 11, 12 - 16 + 7 + 12) = 18 (C)

F_4(1) = max(5 + 10, 10 - 16 + 7 + 12) = 15 (C)

F_4(2) = max(6 + 8, 8 - 16 + 7 + 12) = 14 (C)

F_4(3) = max(4 + 6, 8 - 16 + 7 + 12) = 11 (3)

F_4(4) = max(4 + 5, 7 - 16 + 7 + 12) = 10 (3)

F_4(5) = max(2 + 2, 6 - 16 + 7 + 12) = 9 (3)

F_4(6) = max(3 + 0, 4 - 16 + 7 + 12) = 7 (3)

F_3(0) = max(7 + 15, 12 - 16 + 7 + 18) = 22 (C)

F_3(1) = max(5 + 14, 10 - 16 + 7 + 18) = 19 (C)

F_3(2) = max(6 + 11, 8 - 16 + 7 + 18) = 17 (C)

F_3(3) = max(4 + 10, 8 - 16 + 7 + 18) = 17 (3)

F_3(4) = max(4 + 9, 7 - 16 + 7 + 18) = 16 (3)

F_3(5) = max(2 + 7, 6 - 16 + 7 + 18) = 15 (3)

F_3(6) = max(3 + 0, 4 - 16 + 7 + 18) = 13 (3)

F_2(0) = max(7 + 19, 12 - 16 + 7 + 22) = 26 (C)

F_2(1) = max(5 + 17, 10 - 16 + 7 + 22) = 23 (3)

F_2(2) = max(6 + 17, 8 - 16 + 7 + 22) = 23 (C)

F_2(3) = max(4 + 16, 8 - 16 + 7 + 22) = 21 (3)

F_2(4) = max(4 + 15, 7 - 16 + 7 + 22) = 20 (3)

F_2(5) = max(2 + 13, 6 - 16 + 7 + 22) = 19 (3)

F_2(6) = max(3 + 0, 4 - 16 + 7 + 22) = 17 (3)

F_1(0) = max(7 + 23, 12 - 16 + 7 + 26) = 30 (C)

F_1(1) = max(5 + 23, 10 - 16 + 7 + 26) = 28 (C)

F_1(2) = max(6 + 21, 8 - 16 + 7 + 26) = 27 (C)

F_1(3) = max(4 + 20, 8 - 16 + 7 + 26) = 25 (3)

F_1(4) = max(4 + 19, 7 - 16 + 7 + 26) = 24 (3)

F_1(5) = max(2 + 17, 6 - 16 + 7 + 26) = 23 (3)

F_1(6) = max(3 + 0, 4 - 16 + 7 + 26) = 21 (3)

Безусловная оптимизация

1-й год эксплуатации

Возраст оборудования: 1

Выгоднее оставить это оборудование

2-й год эксплуатации

Возраст оборудования: 2

Выгоднее оставить это оборудование

3-й год эксплуатации

Возраст оборудования: 3

Выгоднее заменить это оборудование

4-й год эксплуатации

Возраст оборудования: 0

Выгоднее оставить это оборудование

5-й год эксплуатации

Возраст оборудования: 1

Выгоднее оставить это оборудование

6-й год эксплуатации

Возраст оборудования: 2

Выгоднее оставить это оборудование

Матрица максимальных прибылей:

[(30, 'С'), (28, 'С'), (27, 'С'), (25, 'З'), (24, 'З'), (23, 'З'), (21, 'З')]

[(26, 'С'), (23, 'З'), (23, 'С'), (21, 'З'), (20, 'З'), (19, 'З'), (17, 'З')]

[(22, 'С'), (19, 'С'), (17, 'С'), (17, 'З'), (16, 'З'), (15, 'З'), (13, 'З')]

[(18, 'С'), (15, 'С'), (14, 'С'), (11, 'З'), (10, 'З'), (9, 'З'), (7, 'З')]

[(12, 'С'), (11, 'С'), (10, 'С'), (8, 'С'), (6, 'С'), (5, 'С'), (2, 'З')]

[(7, 'С'), (5, 'С'), (6, 'С'), (4, 'С'), (4, 'С'), (2, 'С'), (3, 'С')]

Оптимальная стратегия:

|1 год: Сохранить| -> |2 год: Сохранить| -> |3 год: Заменить| -> |4 год: Сохранить| -> |5 год: Сохранить| -> |6 год: Сохранить|

2.2.4.5 Результаты разработки

Приведена матрица максимальных прибылей для каждого года и оптимальная стратегия в виде последовательности решений о замене или сохранении оборудования. Данные представлены для наглядного понимания работы алгоритма и эффективности принятых стратегий.

Также приведены расчеты рекуррентных формул для возможности сравнить все вычисления с примером для алгоритма из пункта 2.2.3.4. Они полностью совпадают.

В процессе решения задачи оптимизации замены оборудования был разработан и реализован алгоритм с использованием методов динамического программирования на языке программирования Python. Были проведены

эксперименты с заданными входными данными, включающими годовой доход, ежегодные затраты на обслуживание, остаточную стоимость, стоимость нового оборудования, и начальный возраст оборудования.

Реализация алгоритма успешно определяет оптимальную стратегию эксплуатации оборудования на заданный период, указывая, в какие годы следует производить замену, а в какие оставлять оборудование без изменений. Результаты реализации алгоритма согласуются с ожидаемыми выводами, демонстрируя его эффективность в принятии решений по оптимизации.

2.2.5 Выводы по разделу 2.2

В ходе данного исследования подробно рассмотрена задачу оптимизации замены оборудования и разработали эффективный алгоритм для её решения. Основываясь на методах динамического программирования и использовании языка программирования Python, создан функциональный и легко настраиваемый инструмент, который позволяет принимать обоснованные решения относительно времени замены и сохранения оборудования.

Реализация алгоритма была успешно протестирована с заданными входными данными, и результаты его работы соответствуют ожиданиям, что подтверждает его надежность и применимость в практических сценариях. Введение методов динамического программирования в данную задачу обеспечивает оптимальное решение, основанное на полной информации о доходах, затратах и стоимости оборудования на каждом этапе его эксплуатации.

Объединяя теоретический аспект и практическое применение, данная работа позволяет лучше понять и оптимизировать стратегии замены оборудования в различных промышленных и бизнес-сценариях.

Использование программирования на Python делает данный метод доступным для широкого круга специалистов, обладающих базовыми навыками в области программирования.

Таким образом, разработанный алгоритм предоставляет полезный инструмент для принятия стратегических решений в области обслуживания и замены оборудования, что может привести к существенной экономии ресурсов и повышению эффективности в производственных и бизнес-процессах.

2.3 Задача о рюкзаке

2.3.1 Теоретический анализ и модель решения задачи: Алгоритмы и подходы

Задача о рюкзаке — это комбинаторная оптимизационная задача, которая заключается в выборе определенного набора предметов из заданного множества так, чтобы их суммарная стоимость была максимальной, при условии, что их общий вес не превышает заданную вместимость рюкзака.

Задача о рюкзаке является классической задачей комбинаторной оптимизации и применяется в различных областях, включая логистику, финансы, производство, управление запасами и другие, суть ее изображена на рисунке 4.

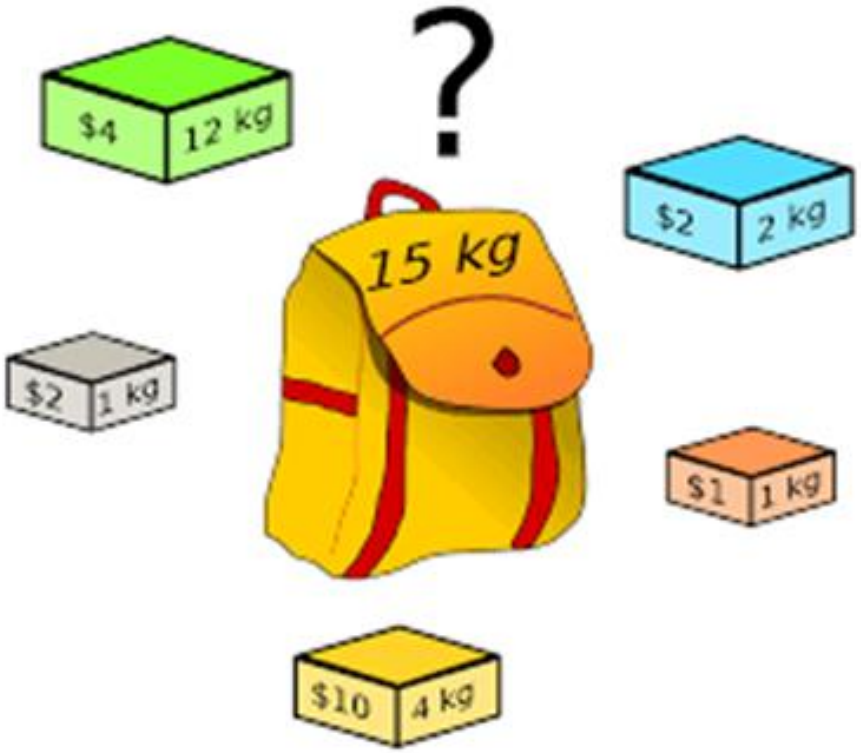

Рисунок 4 – иллюстрация задачи о рюкзаке

Существует несколько вариантов задачи о рюкзаке:

1. рюкзак (0/1 Knapsack)

Описание: Каждый предмет может быть полностью взят или вовсе не взят.

Формальная постановка: Пусть есть n предметов, у каждого из которых есть вес w_i и стоимость v_i. Также есть рюкзак с максимальной грузоподъемностью W. Задача состоит в выборе подмножества предметов с максимальной суммарной стоимостью, при условии, что суммарный вес выбранных предметов не превышает W.

2. дробный рюкзак (Fractional Knapsack)

Описание: Предметы можно делить на части, чтобы взять только часть предмета.

Формальная постановка: Аналогична 0/1 рюкзаку, но теперь можно брать части предметов. Каждый предмет имеет вес w_i, стоимость v_i, и определенное количество c_i (которое может быть дробным). Задача состоит в максимизации суммарной стоимости предметов, уложив их в рюкзак с грузоподъемностью W.

3. многомерный рюкзак (Multidimensional Knapsack)

Описание: Предметы имеют несколько параметров, и задача – удовлетворить ограничения по всем параметрам.

Формальная постановка: Предметы характеризуются вектором параметров (wi1, wi2, ..., wik), и рюкзак имеет ограничения по всем этим параметрам. Задача заключается в выборе комбинации предметов для максимизации суммарной стоимости, с учетом ограничений по параметрам.

4. многорюкзаковая задача (Multiple Knapsack)

Описание: Есть несколько рюкзаков с разными грузоподъемностями, и задача – распределить предметы между ними.

Формальная постановка: По сути, это набор из нескольких задач о 0/1 рюкзаке, где каждый рюкзак имеет свою грузоподъемность.

5. задача о рюкзаке с зависимостью (Knapsack with Dependencies)

Описание: Предметы могут зависеть друг от друга, и выбор одного предмета может влиять на возможность выбора другого.

Формальная постановка: Каждый предмет имеет связи с другими предметами. Например, выбор предмета A может сделать предмет B недоступным. Задача – найти оптимальное сочетание предметов с учетом зависимостей.

Что решает задача о рюкзаке:

1. оптимизация ресурсов: Задача о рюкзаке решает проблему оптимального распределения ограниченных ресурсов, таких как вместимость рюкзака (грузоподъемность), между различными объектами (предметами) с разными характеристиками (весами и стоимостями);

2. максимизация ценности: Цель задачи состоит в том, чтобы максимизировать общую ценность выбранных предметов, уложив их в рюкзак, не превышая установленные ограничения по весу

Можно ли обойтись без нее:

1. во многих сценариях задача о рюкзаке представляет собой удобный и формализованный способ решения определенных проблем оптимизации. В ситуациях, где необходимо управлять распределением ограниченных ресурсов, эта задача может предоставить важные инсайты в принятии решений;

2. однако существуют ситуации, где задача о рюкзаке может быть упрощена или заменена другими методами оптимизации в зависимости от конкретного контекста. Например, если есть дополнительные ограничения или дополнительные переменные, может потребоваться более сложная модель оптимизации.

Задача о рюкзаке, впервые выдвинутая математиком и экономистом Дороти Вейгандт в её статье "A Problem in Thermo-Economics" ("Проблема в термо экономике"), опубликованной в 1927 году в журнале "Econometrica", стала ключевым объектом исследования в области оптимизации и математического программирования. Вейгандт столкнулась с необходимостью решения задачи, где бюджет ограничен, и требовалось определить, какие товары следует выбрать для максимизации полезности при

установленных ценах и бюджетных ограничениях.

Анализируя схожесть этой задачи с проблемой распределения энергии в термодинамике, Вейгандт формализовала задачу и продемонстрировала использование определенного вида математического программирования при ее решении. Несмотря на то, что сама Вейгандт не применяла термин "задача о рюкзаке", её работа стала первым известным описанием этой задачи, которая впоследствии получила это название.

Впоследствии задача о рюкзаке привлекла внимание других математиков, и началось активное исследование, приведшее к разработке различных методов и алгоритмов для ее решения. Вода термо экономики, привнесенная Вейгандт, соединилась с потоками оптимизации, наполнив исследования новыми подходами к решению задачи о рюкзаке.

Различные алгоритмы решения задачи о рюкзаке:

1. жадные алгоритмы:

Жадные алгоритмы представляют собой подход к решению задач оптимизации, при котором на каждом этапе выбирается локально оптимальное решение в надежде, что таким образом будет достигнут глобально оптимальный результат. В контексте задачи о рюкзаке это означает выбор предметов с наивысшим соотношением стоимость/вес на каждом шаге.

Одним из преимуществ жадных алгоритмов является их эффективность и простота реализации. Однако, в случае задачи о рюкзаке, жадный подход может не всегда гарантировать получение оптимального решения. В некоторых ситуациях, когда жадный выбор на каждом шаге не учитывает будущие возможности, может произойти ситуация, когда глобально оптимальное решение не будет достигнуто. Пример: Жадный выбор

предметов с наивысшим соотношением стоимости к весу.

2. метод ветвей и границ

Метод ветвей и границ в контексте задачи о рюкзаке — это эффективный алгоритм для поиска оптимального решения. В данной задаче, где требуется выбрать определенный набор предметов с ограничением по весу, метод ветвей и границ предоставляет эффективный путь поиска оптимального решения без полного перебора всех возможных комбинаций.

3. динамическое программирование:

Динамическое программирование для задачи о рюкзаке представляет собой эффективный метод поиска оптимального решения, основанный на разбиении задачи на более мелкие подзадачи и использовании рекурсии для нахождения оптимального значения.

Пример: Метод заполнения таблицы, где значение ячейки (i, w) представляет собой максимальную стоимость, которую можно получить, уложив первые i предметов в рюкзак с вместимостью w.

Динамическое программирование для задачи о рюкзаке:

В задаче о рюкзаке этот метод может быть применен путем разделения задачи на подзадачи, рассматривая выбор предметов с учетом оставшейся вместимости рюкзака. Рекурсивно строятся решения подзадач, учитывая вес и стоимость предметов. Промежуточные результаты хранятся в таблице или массиве, чтобы избежать повторных вычислений. Таблица заполняется снизу вверх, начиная с самых маленьких подзадач и постепенно увеличивая их размер, пока не достигнут исходной задачи. Оптимальное решение определяется после заполнения таблицы, используя промежуточные результаты. Применение динамического программирования к задаче о рюкзаке позволяет эффективно находить оптимальное решение, избегая

повторных вычислений и сохраняя результаты подзадач для дальнейшего использования.

Давайте рассмотрим шаги этого метода более подробно:

1. формулировка задачи

Рассмотрим задачу о рюкзаке. У нас имеется набор предметов, каждый из которых характеризуется двумя параметрами: весом (weight) и стоимостью (value). Помимо этого, у нас есть рюкзак ограниченной вместимости (capacity). Данная задача заключается в выборе оптимального набора предметов так, чтобы их суммарный вес не превышал вместимость рюкзака, а суммарная стоимость была максимальной.

Примечание: Вес и стоимость предметов могут быть представлены в виде списков, где элемент с индексом i соответствует i-му предмету.

2. формулировка подзадач

Задача о рюкзаке разбивается на последовательность подзадач. Можно рассмотреть каждый предмет и каждый возможный вес рюкзака как отдельную подзадачу. Для каждой из этих подзадач стремимся определить оптимальную стоимость, которую можно достичь с учетом ограничений веса.

Обозначим оптимальную стоимость для i-го предмета и рюкзака с вместимостью w как optimal_value(i, w). Таким образом, формулируются подзадачи, например, optimal_value(1, 2) представляет собой оптимальную стоимость для первого предмета при вместимости рюкзака в 2 единицы веса.

3. рекуррентное соотношение

Оптимальная стоимость для каждой подзадачи мож где: i – индекс предмета, w – текущий вес рюкзака, weight[i] – вес i-го предмета, value[i] – стоимость i-го предмета.

Это соотношение говорит о том, что оптимальная стоимость для текущей подзадачи равна максимуму из двух случаев: либо предмет не добавлен в рюкзак (optimal_value(i-1, w)), либо добавлен (optimal_value(i-1, w -weight[i]) + value[i]).

Выбирается вариант с максимальной стоимостью.

Оптимальная стоимость для каждой подзадачи может быть выражена следующим рекуррентным соотношением:

optimal_value(i, w) = max (optimal_value(i − 1, w), optimal_value(i − 1, w − weight[i]) + value[i]), где:

1) i – индекс предмета;

2) w – текущий вес рюкзака;

3) weight[i] – вес i-го предмета;

4) value[i] – стоимость i-го предмета.

Это соотношение говорит о том, что оптимальная стоимость для текущей подзадачи равна максимуму из двух случаев: либо предмет не добавлен в рюкзак (optimal_value(i − 1, w)), либо добавлен (optimal_value(i − 1, w − weight[i]) + value[i].

Выбирается вариант с максимальной стоимостью.

Принцип рекурсивного соотношения показан на рисунке 5.

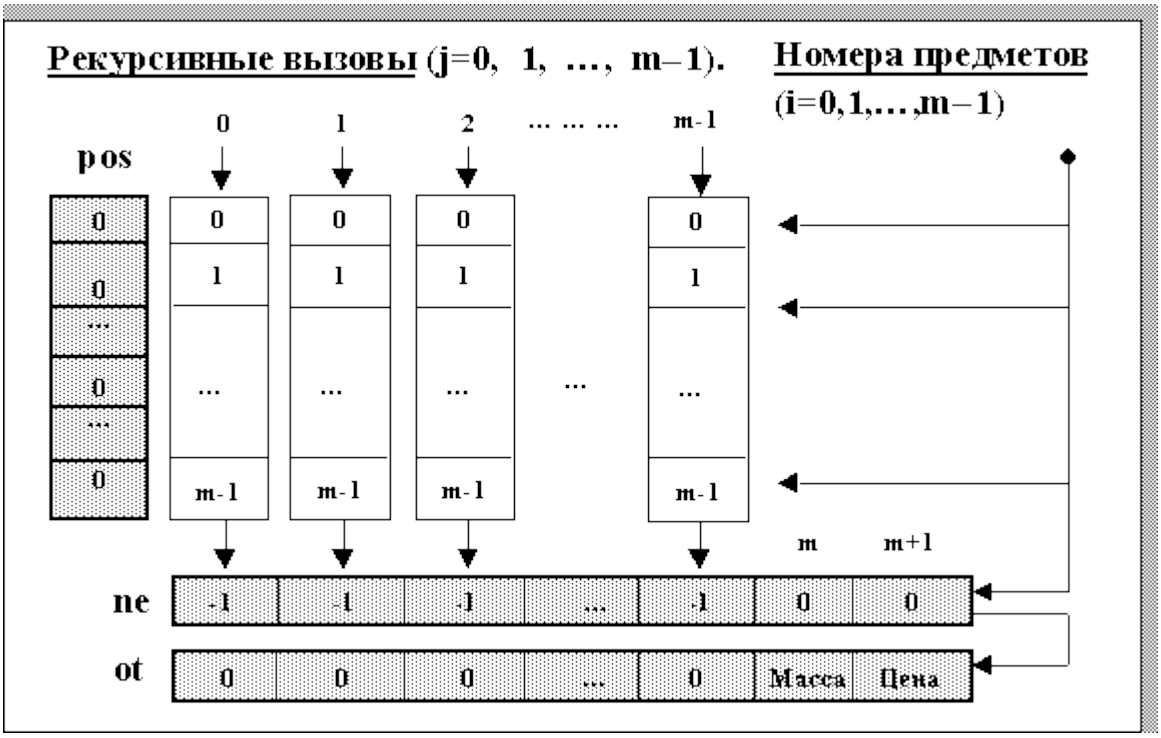

Рисунок 5 – рекурсивные соотношения

4. заполнение таблицы.

Представим, что у нас есть следующий набор предметов как в таблице 8.

Таблица 8 – начальный набор предметов

Предмет	Вес	Стоимость
A	2	10
B	3	15
C	5	7

И рюкзак с максимальной вместимостью capacity = 5

Для каждого предмета и каждого возможного веса рюкзака применяем рекуррентное соотношение.

Заполняем таблицу, начиная с верхнего левого угла и двигаясь вправо.

Начнем с верхнего левого угла таблицы (0, 0) и будем заполнять ее, двигаясь вправо:

Для ячейки (0, 0) значение будет 0, так как рюкзак с весом 0 не может

Далее, для весов 1, 2, 3, 4, 5 будем применять рекуррентное соотношение и заполнять соответствующие ячейки:

1. заполнение строки для предмета А:

1) Для веса 1: max(0, 0 + 10) = 10;

2) Для веса 2: max(0, 0 + 10) = 10;

3) Для веса 3: max(0, 0 + 10) = 10;

4) Для веса 4: max(0, 0 + 10) = 10;

5) Для веса 5: max(0, 0 + 10) = 10.

Записываем данные в таблицу 9.

Таблица 9 – заполненная таблица для предмета А

Предметы / Вес	0	1	2	3	4	5
0	0	0	0	0	0	0
A	0	10	10	10	10	10
B	0					
C	0					

2. заполнение строки для предмета B:

1) для веса 1: max(10, 0 + 15) = 15;

2) для веса 2: max(10, 0 + 15) = 15;

3) для веса 3: max(10, 10 + 15) = 25;

4) для веса 4: max(10, 10 + 15) = 25;

5) для веса 5: max(10, 10 + 15) = 25.

Записываем данные в таблицу 10.

Таблица 10 – заполненная таблица для предмета A и B

Предметы / Вес	0	1	2	3	4	5
0	0	0	0	0	0	0
A	0	10	10	10	10	10
B	0	15	15	25	25	25
C	0					

3. заполнение строки для предмета C:

1) для веса 1: max(15, 0 + 7) = 15;

2) для веса 2: max(15, 0 + 7) = 15;

3) для веса 3: max(25, 10 + 7) = 27;

4) для веса 4: max(25, 15 + 7) = 22;

5) для веса 5: max(25, 15 + 7) = 22.

Записываем данные в таблицу 11.

Таблица 11 – полностью заполненная таблица

Предметы / Вес	0	1	2	3	4	5
0	0	0	0	0	0	0
A	0	10	10	10	10	10
B	0	15	15	25	25	25
C	0	15	15	27	22	22

Таким образом, таблица теперь содержит оптимальные стоимости для каждой комбинации предмета и веса рюкзака. Можно использовать эту таблицу для определения оптимального решения и выбора предметов, которые следует включить в рюкзак для максимизации стоимости.

1. лучшее решение

Псевдокод для оптимального решения задачи о рюкзаке с использованием динамического программирования:

function решение_рюкзака(веса, стоимости, вместимость_рюкзака):

n = длина(веса)

dp = создать_матрицу(n + 1, вместимость_рюкзака + 1)

// Инициализация базовых случаев

для каждого i от 0 до n:

```
        dp[i][0] = 0
    для каждого w от 0 до вместимость_рюкзака:
        dp[0][w]= 0
    // Заполнение матрицы динамического программирования
    для каждого i от 1 до n:
        для каждого w от 1 до вместимость_рюкзака:
            если веса[i-1] <= w:
                dp[i][w] = максимум(dp[i-1][w], стоимости[i-1] + dp[i-1][w-веса[i-1]])
            иначе:
                dp[i][w] = dp[i-1][w]
    вернуть dp[n][вместимость_рюкзака]
// Пример использования
веса = [2, 3, 4, 5];
стоимости = [3, 4, 5, 6];
вместимость_рюкзака = 5;
результат = решение_рюкзака(веса, стоимости, вместимость_рюкзака);
вывести(результат);
```

2. худшее решение

```
function худшее_решение(веса, стоимости, вместимость_рюкзака,
```

```
текущий_индекс) {
    // Базовый случай: если рюкзак полон или достигнут конец предметов
    если (вместимость_рюкзака <= 0 || текущий_индекс < 0) {
        вернуть 0;
    }
    // Вариант 1: пропустить текущий предмет и рассмотреть оставшиеся
    результат1 = худшее_решение(веса, стоимости, вместимость_рюкзака, текущий_индекс - 1);
    // Вариант 2: взять текущий предмет, уменьшить вместимость и рассмотреть оставшиеся предметы
    результат2 = стоимости[текущий_индекс] + худшее_решение(
        веса, стоимости, вместимость_рюкзака - веса[текущий_индекс], текущий_индекс - 1
    );
    // Вернуть максимум из вариантов
    вернуть максимум(результат1, результат2);
}
// Пример использования
веса = [2, 3, 4, 5];
стоимости = [3, 4, 5, 6];
вместимость_рюкзака = 5;
```

результат = худшее_решение(веса, стоимости, вместимость_рюкзака, длина(веса) - 1);

вывести(результат);

Время выполнения :

1. лучшее решение

Время выполнения лучшего решения для задачи о рюкзаке с использованием динамического программирования зависит от количества предметов (n) и вместимости рюкзака (вместимость_рюкзака). В данном случае, алгоритм имеет временную сложность O(n * вместимость_рюкзака), что делает его эффективным для большинства практических сценариев.

Однако, стоит отметить, что динамическое программирование все равно может быть затратным по времени для очень больших значений вместимости рюкзака или большого числа предметов.

2. худшее решение

Время выполнения "худшего решения" будет экспоненциальным относительно количества предметов. Точнее, временная сложность будет O(2^n), где n – количество предметов.

Это связано с тем, что в данном подходе многие подзадачи решаются повторно без сохранения результатов, что приводит к экспоненциальному увеличению количества вызовов функции.

Для оптимизации рекурсивных решений часто используется мемоизация, т.е. сохранение результатов подзадач и их последующее использование.В этом заключается основное различие двух решений

2.3.2 Выбор технологического стека: обоснование и применение

Для решения задачи о рюкзаке с используется язык программирования Python

Выбор Python для написания алгоритма решения задачи о рюкзаке может быть обусловлен несколькими причинами:

1. простота и читаемость кода: Python известен своей простотой и лаконичностью синтаксиса, что упрощает написание и понимание кода. Это особенно полезно при реализации алгоритмов, так как позволяет ясно выражать логику решения задачи;

2. большая стандартная библиотека: Python поставляется с обширной стандартной библиотекой, включающей в себя множество полезных модулей. Это может быть полезно при реализации части функционала, такого как ввод-вывод данных, без необходимости написания кода с нуля;

3.поддержка различных парадигм программирования: Python поддерживает несколько парадигм программирования, включая императивное, объектно-ориентированное и функциональное программирование. Это дает возможность выбрать наилучший стиль программирования для конкретной задачи;

4.активное сообщество и ресурсы: Python имеет огромное и активное сообщество разработчиков. Это означает, что вы найдете множество ресурсов, обучающих материалов, библиотек и фреймворков, что может упростить решение задач и облегчить процесс разработки;

5.широкое применение в научных и исследовательских областях: Python часто используется в научных и исследовательских областях, в том числе для решения задач оптимизации, что может сделать его предпочтительным выбором в подобных сценариях.

Все эти факторы делают Python идеальным выбором для написания алгоритма решения задачи о рюкзаке.

Также были выбраны библиотеки Matplotlib, Numpy .

Matplotlib предоставляет интуитивный интерфейс для создания разнообразных графиков и диаграмм, что делает его привлекательным для широкого круга пользователей. Эта библиотека поддерживает различные типы графиков, от линейных до трехмерных, предоставляя гибкость для создания разнообразных визуализаций. Большое сообщество и обширная документация Matplotlib также облегчают получение поддержки и решение проблем.

NumPy обеспечивает эффективное хранение и манипулирование данными, что особенно важно при выполнении численных вычислений. Богатый набор функций линейной алгебры и математических операций делает NumPy незаменимым инструментом для научных вычислений, обработки данных и разработки алгоритмов в Python.

Выбор этих инструментов позволит более эффективно и наглядно решать задачу о рюкзаке, а также использовать результаты в дальнейших исследованиях и разработках.

2.3.3 Разработка программного решения

Программная реализация.

1. knapsack_dynamic(weights, values, capacity):

Параметры:

1) weights: Массив весов предметов;

2) values: Массив стоимостей предметов;

3) capacity: Вместимость рюкзака.

Описание: функция решает задачу о рюкзаке с использованием динамического программирования.

Возвращаемое значение: максимальная стоимость предметов, которые можно поместить в рюкзак с заданной вместимостью.

2. plot_knapsack_solution(weights, values, capacities):

Параметры:

1) weights: Массив весов предметов;

2) values: Массив стоимостей предметов;

3) capacities: Список вместимостей рюкзака для построения графика.

Описание: функция строит график зависимости максимальной стоимости от вместимости рюкзака для заданных весов и стоимостей.

Возвращаемое значение: график, отображаемый в окне интерактивного графического интерфейса (GUI) библиотеки Matplotlib.

3. генерация случайных данных:

1) np.random.seed(): Инициализация генератора случайных чисел для воспроизводимости результатов. np.random.seed() используется, чтобы гарантировать воспроизводимость результатов генерации случайных данных. Это означает, что при каждом запуске программы будут созданы одни и те же случайные данные;

2) num_items: Количество предметов в рюкзаке;

3) weights = np.random.randint(): Массив случайных весов предметов;

4) values = numpy.random.randinteger(): Массив случайных стоимостей предметов;

5) capacities = [np.random.randint()]: Список случайных вместимостей рюкзака для построения графика;

4. пример использования:

Вызов `plot_knapsack_solution(weights, values, capacities)` с сгенерированными данными для демонстрации решения задачи о рюкзаке с помощью динамического программирования и построения графика.

Анализ полученных графиков.

Благодаря генерации случайных данных и построению графиков для различных вместимостей рюкзака, можно проанализировать результаты:

Различные вместимости приводят к различным оптимальным значениям: Как видно на графиках для разных вместимостей рюкзака, оптимальные значения максимальной стоимости меняются. Это подчеркивает важность учета вместимости при решении задачи о рюкзаке. Оптимальные наборы предметов будут различными в зависимости от того, сколько предметов можно уложить в рюкзак.

Эффективность решения задачи о рюкзаке динамическим программированием: Графики также подтверждают, что использование динамического программирования позволяет эффективно решать задачу о рюкзаке для различных вместимостей. Оптимальные значения стоимости достигаются с использованием динамического программирования, что подчеркивает его эффективность в решении подобных задач комбинаторной оптимизации.

Плавный характер графиков: Графики обычно имеют плавный

характер, что отражает непрерывный характер задачи о рюкзаке. Переход от одной вместимости к другой вызывает постепенные изменения оптимальных значений стоимости.

Настройка параметров: Изменение параметров генерации, таких как количество предметов и диапазоны весов и стоимостей, может влиять на форму и характер графиков. Это демонстрирует, как различные входные данные могут влиять на решение задачи о рюкзаке.

2.3.4 Выводы по разделу 2.3

Динамическое программирование в решении задачи о рюкзаке выделяется на других решений следующим:

1. оптимальная подструктура

Оптимальная подструктура в контексте задачи о рюкзаке означает, что оптимальное решение для всей задачи может быть выражено через оптимальные решения ее подзадач. Другими словами, если известно оптимальное решение для некоторого подмножества предметов и ограниченной вместимости, то можно использовать эту информацию для построения оптимального решения для более крупного подмножества предметов.

В контексте динамического программирования, это свойство позволяет нам эффективно решать подзадачи и затем комбинировать их решения для получения ответа к исходной задаче. Динамическое программирование сохраняет результаты уже решенных подзадач в таблице, избегая повторных вычислений.

2. перекрытие подзадач

В задаче о рюкзаке часто возникает ситуация, когда различные

подмножества предметов могут встречаться в нескольких подзадачах. Это означает, что решение для некоторых комбинаций предметов и вместимостей рюкзака может быть использовано в нескольких случаях.

Динамическое программирование предотвращает повторные вычисления, сохраняя результаты этих подзадач в таблице. Когда алгоритм сталкивается с одной и той же подзадачей впервые, он решает её и сохраняет результат. При следующем обращении к той же подзадаче алгоритм просто извлекает ранее вычисленное значение из таблицы, вместо того чтобы решать её заново.

Преимущества перекрытия подзадач:

1) экономия времени

Перекрытие подзадач позволяет избежать повторных вычислений. Результаты уже решенных подзадач сохраняются, и при необходимости могут быть мгновенно использованы. Это значительно снижает общее время выполнения алгоритма.

2) снижение вычислительной сложности

Поскольку многие подзадачи могут быть решены один раз и использованы в различных контекстах, это снижает общую вычислительную сложность алгоритма.

3) оптимизация памяти

Хранение результатов в таблице позволяет эффективно управлять памятью. Сохраняем только те результаты, которые действительно нужны для решения задачи.

Применение концепции перекрытия подзадач делает динамическое программирование эффективным и удобным методом для решения задачи о рюкзаке, особенно когда есть множество пересекающихся подзадач.

3. эффективность при больших объемах данных

Метод динамического программирования с мемоизацией проявляет высокую эффективность при обработке больших объемов данных. Одной из ключевых причин является возможность избежать повторных вычислений благодаря сохранению результатов решаемых подзадач.

При обработке большого количества данных, где число подзадач может значительно возрастать, этот механизм позволяет существенно сократить количество вычислений.

Также использование мемоизации снижает временную сложность алгоритма, что особенно важно при работе с большими объемами данных. Уменьшение времени выполнения и избежание избыточных вычислений делают метод динамического программирования с мемоизацией эффективным инструментом для оптимизации при обработке больших объемов данных.

4. сложная комбинаторика предметов

Сложная комбинаторика предметов требует тщательного рассмотрения различных методов для оптимального решения задачи. Когда имеется обширное количество предметов и разнообразные комбинации для упаковки, важно выбрать подход, который обеспечит эффективное решение проблемы.

Динамическое программирование позволяет избежать избыточных вычислений, сохраняя результаты подзадач и повторно используя их при необходимости. Это значительно ускоряет процесс принятия решений и позволяет эффективно управлять большим объемом данных.

Таким образом, выбор подхода к решению задачи о решении задачи о рюкзаке играет ключевую роль в обеспечении оптимального и быстрого результата, особенно когда сталкиваешься с множеством предметов и огромным числом возможных комбинаций.

5. учет весов и стоимостей

Задача о рюкзаке, представляющая собой одну из важных задач комбинаторной оптимизации, часто включает в себя не только учет весов предметов, но и их стоимостей. Эта двойная задача, требующая максимизации стоимости переносимых предметов при ограничении на общий вес, представляет дополнительные вызовы при выборе метода решения.

В контексте учета весов и стоимостей, динамическое программирование выделяется как эффективный метод решения задачи о рюкзаке. Этот подход удобно адаптируется для одновременной оптимизации двух критериев – веса и стоимости. Динамическое программирование позволяет эффективно обрабатывать большие объемы данных, разбивая сложную задачу на более простые подзадачи и сохраняя результаты для последующего использования.

В отличие от некоторых других методов, таких как полный перебор или жадные алгоритмы, которые могут сталкиваться с трудностями в учете двух критериев одновременно, динамическое программирование предоставляет эффективный инструмент для решения задачи о рюкзаке в условиях сложной комбинаторики весов и стоимостей предметов. Такой подход позволяет эффективно балансировать требования об учете обоих критериев, обеспечивая оптимальное решение задачи о рюкзаке. оптимальная подструктура.

2.4 Задача о распределении инвестиций

2.4.1 Теоретический анализ и модель решения задачи: Алгоритмы и подходы

2.4.1.1 Постановка задачи

Необходимо распределить X денежных единиц между N разными инвестиционными проектами (предприятиями), если известна прибыль P_n (K) получаемая при вложении k = 1, …, X денежных единиц в N-ое предприятие таким образом, чтобы суммарная прибыль была максимальна.

Переменные описаны в таблице 12.

Таблица 12 – переменные

X	Кол-во денежных единиц
N	Кол-во различных предприятий, в которые могут быть вложены денежные средства
$P_n(k)$	Прибыль от вложения k денежных единиц в n-ое предприятие
k	Натуральные числа от 1 до X
n	Натуральные числа от 1 до N

2.4.1.2 Алгоритм

Рассмотрим применение алгоритма на примере задачи.

Условие задачи представлено в таблице 13.

Таблица 13 – условие задачи

Инвестируемые Средства	Возможная прибыль		
X	p1(x)	p2(x)	p3(x)
1	3,22	3,33	4,27
2	3,57	4,87	7,64
3	4,12	5,26	10,25
4	4	7,34	15,93
5	4,85	9,49	16,12

Составим математическую модель задачи:

1. число шагов равно числу предприятий – 3.

2. пусть s – количество средств, имеющихся в наличии перед данным шагом, и характеризующих состояние системы на каждом шаге;

3. управление на i-ом шаге (i=1,2,3) выберем x_i - количество средств, инвестируемых в i-ое предприятие;

4. выигрыш $p_i(x_i)$ на i-ом шаге – это прибыль, которую приносит i-ое предприятие при инвестировании в него средств x_i. Если через выигрыш в целом обозначить общую прибыль W, то $W = p_1(x_1) + p_2(x_2) + p_3(x_3)$;

5. если в наличии имеются средства в количестве s денежных единиц и в i-ое предприятие инвестируется x денежных единиц, то для дальнейшего инвестирования остается (s-x) тыс. ден. ед. Таким образом, если на i-ом шаге

система находилась в состоянии s и выбрано управление x, то на (i+1)-ом шаге система будет находится в состоянии (s-x), и, следовательно, функция перехода в новое состояние имеет вид: $f_i(s, x) = s-x$;

6. на последнем (i=3) шаге оптимальное управление соответствует количеству средств, имеющихся в наличии, а выигрыш равен доходу, приносимым последним предприятием: $x_3(s) = s$, $W_3(s) = p_3(s)$;

7. согласно принципу оптимальности Беллмана, управление на каждом шаге нужно выбирать так, чтобы оптимальной была сумма выигрышей на всех оставшихся до конца процесса шагах, включая выигрыш на данном шаге. Основное функциональное уравнение примет вид:

$$Wi(s) = max\{x\} \leq s\{pi(x) + Wi + 1(s - x)\} \quad (18)$$

Пошаговая оптимизация представлена в таблице 14.

Таблица 14 – пошаговая оптимизация

s	i = 3		i = 2		i = 1	
	$X_3(s)$	$W_3(s)$	$X_2(s)$	$W_2(s)$	$X_1(s)$	$W_1(s)$
1	1	4,27	0	4,27		
2	2	7,64	0	7,64		
3	3	10,25	1	10,97		
4	4	15,93	0	15,93		
5	5	16,12	1	19,26	0	19,26

В первой колонке таблицы записываются возможные состояния системы, в верхней строке – номера шагов с оптимальным управлением и выигрышем на каждом шаге, начиная с последнего. Так как для последнего шага i=3 функциональное уравнение имеет вид x3(s)=s, W3(s)=p3(s), то две колонки таблицы, соответствующие i=3, заполняются автоматически по таблице исходных данных. На шаге i=2 основное функциональное уравнение имеет вид:

$$W2(s) = maxx \leq s\{p2(x) + W3(s-x)\} \quad (19)$$

Поэтому для проведения оптимизации на этом шаге заполним таблицу 15 для различных состояний s при шаге i=3.

Таблица 15 – различные состояния s при шаге i=3

s	x	s-x	$p_2(x)$	$W_3(s-x)$	$p_2(x)+W_3(s-x)$	$W_2(s)$
1	0	1	0	4,27	4,27	4,27
	1	0	3,33	0	3,33	
2	0	2	0	7,64	7,64	7,64
	1	1	3,33	4,27	7,6	
	2	0	4,87	0	4,87	
3	0	3	0	10,25	10,25	10,97
	1	2	3,33	7,64	10,97	
	2	1	4,87	4,27	9,14	
	3	0	5,26	0	5,26	
4	0	4	0	15,93	15,93	15,93
	1	3	3,33	10,25	13,58	
	2	2	4,87	7,64	12,51	
	3	1	5,26	4,27	9,53	
	4	0	7,34	0	7,34	

Продолжение таблицы 15

5	0	5	0	16,12	16,12	19,26
	1	4	3,33	15,93	19,26	
	2	3	4,87	10,25	15,12	
	3	2	5,26	7,64	12,9	
	4	1	7,34	4,27	11,61	
	5	0	9,49	0	9,49	

На шаге i=1 основное функциональное уравнение имеет вид:

$$W1(s) = maxx \leq s\{p1(x) + W2(s-x)\} \quad (20)$$

А состояние системы перед первым шагом s=5, поэтому для проведения оптимизации на этом шаге заполним таблицу 16.

Таблица 16 – состояние системы перед первым шагом s=5.

s	x	s-x	$p_2(x)$	$W_3(s-x)$	$p_2(x)+W_3(s-x)$	$W_2(s)$
5	0	5	0	16,12	16,12	19,26
	1	4	3,33	15,93	19,26	
	2	3	4,87	10,25	15,12	
	3	2	5,26	7,64	12,9	
	4	1	7,34	4,27	11,61	
	5	0	9,49	0	9,49	

Видно, что наибольшее значение выигрыша составляет 19,26. При этом оптимальное управление на первом шаге составляет $x_1(s_1) = 0$ ($s_1 = 5$), на

втором шаге $x_2(s_2) = 1$ ($s_2 = s_1 - x_1 = 5$) и на третьем шаге $x_3(s_3) = 4$ ($s_3 = s_2 - x_2 = 4$). Это означает, что (0, 1, 4) – оптимальный план распределения инвестиций между предприятиями. Таким образом, для получения наибольшей общей прибыли в размере 19,26 денежных единиц, необходимо вложить 1 денежную единицу во второе предприятие и 4 денежные единицы в третье предприятие.

2.4.2 Выбор технологического стека: обоснование и применение

Для решения задачи будет использоваться язык C#, который считается одним из самых универсальных. Он применяется в самых разных сферах. Например, для создания продвинутых бизнес-приложений, видеоигр, функциональных веб-приложений, приложений для Windows, macOS, мобильных программ для iOS и Android.

Перечислим некоторые плюсы выбора этого языка для данной задачи:

1. ООП подход

C# – это объектно-ориентированный язык программирования, что делает его идеальным для создания сложных математических моделей и задач. ООП позволяет создавать четкую и удобную структуру программы, учитывая различные математические концепции, такие как классы, объекты, наследование и полиморфизм.

2. интеграция с .NET Framework

C# разработан Microsoft и напрямую связан с .NET Framework, что предоставляет огромное количество библиотек и инструментов для работы с математическими вычислениями, графикой, обработкой данных и другими математическими операциями. Это упрощает разработку и оптимизацию оконного приложения для решения математических задач.

3. удобство работы с графическим интерфейсом

C# позволяет легко создавать интуитивно понятные и привлекательные пользовательские интерфейсы, что особенно важно для оконных приложений, предназначенных для решения математических задач. C# обладает мощными инструментами для работы с графикой и интерактивными элементами, что делает его отличным выбором для создания пользовательских интерфейсов и визуализации математических данных.

2.4.3 Разработка программного решения

Нужно написать код, в котором определяются несколько функций для работы с массивами и вычисления различных величин, таких как максимальное и минимальное значение в массиве и др. А также запрограммировать окно, которое будет использоваться для ввода и вывода. Этот функционал необходим для решения поставленной задачи. Для начала необходимо определить, какие функции необходимы для решения данной задачи. Название функции и ее назначение представлены в таблице 17. Код функций приведён в приложении Д

Таблица 17 – описание функций программы

Функция	Назначение
void SetLOG(string s)	Заполнения логов вычислений при работе программы
int max(int val1, int val2,out int index)	Определение какое из двух чисел наибольшее

Продолжение таблицы 17

Функция	Назначение
void SetLOG(string s)	Заполнения логов вычислений при работе программы
int fmax(int fi, int Xi)	Определение наибольшего значения суммарной прибыли при использовании данного количества денежных единиц для данного проекта
public MainWindow()	Инициализировать окно, используемое для получения вводных данных и демонстрации результатов работы программы
void UpdateMatrix(int[][] table, ref DataGrid DataViewer)	Обновление содержимого таблицы внутри окна
private void btnLoadTable_Click(object sender, RoutedEventArgs e)	Загрузка таблицы вводных данных
private void btnStart_Click(object sender, RoutedEventArgs e)	Исполнение вычислений и вывод результата после нажатия соответствующей кнопки в окне программы

2.4.4 Вывод по разделу 2.4

В ходе исследования проблемы нахождения решения задачи о об оптимальном распределении инвестиций был предложен и реализован алгоритм, основанный на методах динамического программирования. Этот алгоритм позволяет с высокой точностью и эффективностью найти искомое решение. Данное решение основывается на динамическом программировании, что позволяет избежать повторных вычислений и снизить вычислительную сложность задачи.

2.5 Задача о распределении ресурсов

2.5.1 Задача об оптимальном распределении ресурсов между предприятиями

Одной из важнейших задач управления экономическими и производственными системами является задача распределения ресурсов. Ресурс является источником или запасом, из которого производится прибыль. Ресурсы обычно классифицируются на основе их доступности, а также возможности их возобновления. Обычно ресурсами являются материалы, энергия, услуги, персонал, знания или другие активы, которые преобразуются для получения прибыли предприятия и компании.

В экономике задача распределения ресурсов заключается в распределении доступных ресурсов для различных целей. В контексте макроэкономики ресурсы могут распределяться различными способами, в том числе, такими как рынки или централизованное планирование. В управлении проектами распределение ресурсов или управление ресурсами – это планирование действий и ресурсов, востребованных в проектах с учетом их доступности.

Для решения задач оптимального распределения ресурсов используется алгоритм, основанный на методе динамического программирования. Реализация данного алгоритма представляет научно-практический интерес.

Таким образом, работы обусловлена необходимостью решения задач оптимального распределения ресурсов, возникающих в управлении экономическими и производственными системами, на основе метода динамического программирования.

Объект исследования – задача оптимального распределения ресурсов на основе динамического программирования.

Предмет исследования – алгоритм решения задачи распределения ресурсов методом динамического программирования.

Целью работы является обоснование метода динамического программирования для решения задачи распределения ресурсов, решение задачи распределения ресурсов аналитически и с помощью программных средств.

Для достижения поставленной цели необходимо решить следующие задачи:

1. изучить общий подход динамического программирования;
2. разработать алгоритм для решения задачи распределения ресурсов;
3. выполнить программную реализацию разработанного алгоритма.

Общая постановка задачи динамического программирования

Динамическое программирование – это метод математической оптимизации и метод компьютерного программирования. Этот метод был разработан Ричардом Беллманом в 1950-х годах и нашел применение во многих областях: от аэрокосмической техники до экономики. В динамическом программировании задача разделяется на подзадачи, и решения этих подзадач объединяются вместе для достижения общего решения основной задачи. При использовании подходов, таких как «разделяй и властвуй», подзадача может быть решена несколько раз. Динамическое программирование должно решить каждую из этих подзадач только один раз, тем самым уменьшив количество вычислений, а затем сохранит результат,

избегая повторного вычисления на более позднем этапе, когда требуется решение для этой подзадачи.

Динамическое программирование используется для решения задач оптимизации (например, поиск кратчайшего пути), где может существовать множество решений, но интерес представляет только поиск оптимального решения для одного и того же (существует множество решений, которые достигают оптимального значения).

Эти проблемы должны иметь свойства перекрывающихся подзадач. Иными словами, решаемая задача может быть разбита на подзадачи, которые многократно используются, причем рекурсивный алгоритм решает одну и ту же подзадачу много раз, а не создает новую подзадачу. Например, числа Фибоначчи и оптимальная подструктура. В конечном итоге, оптимальное решение может быть построено из оптимальных решений подзадач.

Следует отметить, что во многих случаях алгоритмы перебора могут работать быстрее, чем алгоритмы динамического программирования, но они не гарантируют оптимальное решение задачи.

2.5.2 Метод динамического программирования

При рассмотрении классического метода ДП принимаются следующие основные допущения относительно характера решаемых задач. Данные допущения выполняются для широкого класса экономических и технических систем и процессов, а отказ от них приводит к существенному повышению сложности и громоздкости методов решения соответствующих задач.

Отсутствие последействия: состояние x_i системы S после шага с номером i непосредственно зависит только от предшествующего состояния x_{i-1} и от управления u_i на данном шаге. Это предложения можно записать следующим образом:

$$x_i = f_i(x_{i-1}, u_i), \quad (21)$$

или, в развернутом виде,

$$x_1 = f_1(x_0, u_1), x_2 = f_2(x_1, u_2), \dots, x_N = f_N(x_{N-1}, u_N).$$
(22)

Зависимость состояния x_i от предшествовавших ранее состояний $x_{i-2}, x_{i-3}, \dots$ (т.е. от всей долгой предыстории) является косвенной, проявляющейся только через x_{i-1}.

Для частных целевых функций z_iна каждом из шагов процесса принимается аналогичное предположение:

$$z_i = z_i(x_{i-1}, u_i).$$
(23)

Аддитивность целевой функции относительно разбиения процесса на шаги: результирующая целевая функция Z для всего многошагового процесса представляется в виду суммы частных целевых функций z_i на каждом из шагов:

$$Z = z_1 + z_2 + \cdots + z_N.$$
(24)

Целевая функция Z данного вида иначе называется аддитивным критерием.

Замечания по оптимизации многошаговых процессов: необходимо подчеркнуть, что оптимизация процесса по совокупности шагов не сводится к независимой оптимизации по отдельным шагам. Действительно, достижение максимального эффекта на одном из шагов может потребовать слишком больших затрат ресурсов и, как следствие, привести к снижению экономического эффекта как на последующих шагах, так и по всему многошаговому процессу в целом. Наглядным примером может служить бег на длинную дистанцию, который никоим образом не сводится к многократному повторению бега на короткую дистанцию на максимальной скорости.

Это замечание может быть проиллюстрировано математически следующим свойством: максимум(или минимум) суммы не равен сумме максимумов слагаемых. Точнее, для произвольных функций *a(t)* и *b(t)* можно гарантировать лишь выполнение неравенства

$$\max_t\{a(t)+b(t)\} \le \max_t\{a(t)\} + \max_t\{b(t)\}$$
(25)

2.5.2.1 Методика вычисления оптимального значения задачи

Как может быть осуществлено вычисление оптимального значения Z задачи управления многошаговыми процессами в частном случае, когда начальное состояние x_0 системы фиксировано, или, иными словами, множество X_0 начальных состояний системы сводится к единственному элементу x_0. Проводимые рассуждения позволят ближе подойти к понимаю общего принципа оптимальности решения рассматриваемых задач. Для определенности нужно рассмотреть случай поиска максимума.

Прежде всего простейший из многошаговых процессов, включающий два шага управления, $N = 2$. В этом случае вектор управляющих переменных (u_1, u_2) содержит две компоненты, и целевая функция Z является функцией двух переменных u_1, u_2:

$$Z = Z(u_1, u_2) = z_1(x_0, u_1) + z_2(x_1, u_2),$$
(26)

где $x_1 = f_1(x_0, u_1)$.Для определения оптимального значения Z необходимо вычислить максимум целевой функции по обеим управляющим переменных u_1, u_2:

$$Z = \max_{u_1,u_2}\{z_1(x_0, u_1) + z_2(x_1, u_2)\}.$$
(27)

Максимум по двум переменным всегда может быть найден путем последовательного вычисления максимумов по каждой из переменных, следовательно,

$$Z = \max_{u_1}\{\max_{u_2} z_1(x_0, u_1) + z_2(x_1, u_2)\}\}.$$
(28)

Конечно, для справедливости такого преобразования необходимо, чтобы множества векторов (u_1, u_2), учитываемых при одновременном и при последовательном способах вычисления Z, полностью совпали. В полученной сумме первое слагаемое $z_1(x_0, u_1)$ не зависит от переменной u_2, по

которой вычисляется внутренний максимум. Следовательно, это слагаемое может быть вынесено за знак внутреннего максимума:

$$\max_{u_2}\{z_1(x_0,u_1)+z_2(x_1,u_2)\}=z_1(x_0,u_1)+\max_{u_2}\{z_2(x_1,u_2)\}(29)$$

Таким образом, справедливо равенство

$$Z \quad =\max_{u_1}\{z_1(x_0,u_1)+\max_{u_2}\{z_2(x_1,u_2)\}\}.$$

(30)

В результате проведенных преобразований вычисление максимума по двум переменным u_1,u_2 удалось «разнести» на два последовательных вычисления максимума по переменных u_2 и u_1, т.е. двумерная задача свелась к последовательному решению двух одномерных задач. Как правило, одномерная задача существенно проще двумерной(и тем более многомерной), а проведенные преобразования соответствуют общей закономерности: решить несколько задач низшей размерности проще, чем решить одну задачу высшей размерности.

Структура последнего равенства «подсказывает», что вычисление Z следует разбить на 2 шага – это соответствует структуре многошагового процесса и приводит к более простым выражениям. С этой целью введем новую вспомогательную функцию $B_1(x_1)$, принимая в качестве нее внутренний максимум:

$$B_1(x_1)=\max_{u_2}\{z_2(x_1,u_2)\}.$$

(31)

Смысл функции $B_1(x_1)$ состоит в том, что она представляет собой максимальное значение частной целевой функции z_2 на втором шаге процесса при условии, что перед этим шагом управляемая система находилась в состоянии x_1. С учетом введенной функции $B_1(x_1)$ по переменной u_1 вычисляется при условии, что x_1 не является произвольным, а определяется равенством $x_1=f_1(x_0,u_1)$.

Преобразуем два последних выражения к однотипной форме. Для единообразия обозначений введем две новых функции: функцию $B_2(x_2)$, тождественно равную нулю: $B_2(x_2)=0$ для всех x_2, и функцию $B_0(x_0)$,

определенную только для одного значения x_0 и равную оптимальному значению Z , подлежащему вычислению: $B_0(x_0) = Z$. С использованием введенных обозначений полученные выше равенства можно записать следующим образом:

$$B_1(x_1) = \max_{u_2}\{z_2(x_1, u_2) + B_2(x_2) | x_2 = f_2(x_1, u_2)\},$$
$$B_0(x_0) = \max_{u_1}\{z_1(x_0, u_1) + B_1(x_1) | x_1 = f_1(x_0, u_1)\}.$$
(32)

Важно подчеркнуть, что полученные два равенства полностью совпадают по структуре и отличаются только индексами переменных и функций. Таким образом, проведенные преобразования с использованием вспомогательных функций $B_0(x_0), B_1(x_1)$ и $B_2(x_2)$ не только позволили упростить вычисления оптимального значения задачи Z путем снижения размерности, но и привели вычисления к однотипной форме, соответствующей структуре процесса. Расчеты проводятся в порядке убывания индексов: $B_2(x_2) = 0$ по определению, с учетом этого равенства далее вычисляется $B_1(x_1)$; наконец, вычисляется $B_0(x_0) = Z$. На вид последних двух равенств стоит обратить особое внимание, поскольку он является характерным для записи общего принципа оптимальности.

Отметим, что для одношагового процесса $N = 1$ можно формально провести такую же логику. Вектор управляющих переменных в этом случае содержит только одну компоненту, и целевая функция Z является функцией одной переменной u_1:

$$Z = Z(u_1) = z_1(x_0, u_1).$$
(33)

В этом случае решение задачи оптимизации сводится к поиску максимума функции одной переменной:

$$Z = \max_{u_1}\{z_1(x_0, u_1)\}.$$

Причем введение соответствующих вспомогательных функций $B_0(x_0) = Z$ и $B_1(x_1) = 0$позволяет придать последнему соотношению уже полученный

выше типовой вид (хотя, конечно, для $N = 1$ такое преобразование не имеет глубокого смысла и является скорее усложнением, чем упрощением задачи).

Рассмотренная методика вычисления оптимального значения задачи Z может быть распространена и на общий случай многошаговых процессов. При этом могут быть проведены рассуждения, подобные известному методу математической индукции. Действительно, для $N = 1$ и $N = 2$способ вычисления Z уже рассмотрен выше. Предположим, что можно вычислить оптимальное значение задачи для любого процесса, включающего $N - 1$ шаг, и рассмотрим N-шаговый процесс. На первом шаге под действием управления u_1 система из начального состояния x_0 перейдет в состояние $x_1 = f_1(x_0, u_1)$, обеспечив экономический эффект $z_1(x_0, u_1)$. При этом для последующего перевода системы из состояния x_1 в конечное состояние x_N остается ровно $N - 1$шаг. В силу предположения для получаемого «укороченного» на 1 шаг процесса может быть вычислено оптимальное значение задачи – обозначим его через $B_1(x_1)$ и назовем условно-оптимальным, поскольку оно относится не ко всему процессу и зависит от выбора состояния x_1. В соответствии с аддитивностью критерия для любого фиксированного управления u_1 наибольшее значение целевой функции Z всего N-шагового процесса будет равно

$$z_1(x_0, u_1) + B_1(x_1)$$

(частный экономический эффект на первом шаге плюс максимальный эффект на последующих $N - 1$ шагах). Следовательно, для вычисления оптимального значения Z всей задачи (обозначим его для единообразия, как и выше, через $B_0(x_0)$ достаточно взять максимум от рассмотренной суммы по всевозможным управлениям u_1 на первом шаге:

$$Z = B_0(x_0) = \max_{u_1}\{z_1(x_0, u_1) + B_1(x_1) | x_1 = f_1(x_0, u_1)\}. \quad (34)$$

Данное соотношение совпадает по форме с типовыми равенствами, полученными выше для случаев $N = 2, N = 1$; конечно, его можно было бы также получить путем строгих аналитических преобразований, аналогичных

проведенным выше для случая $N = 2$. Следуя подобной методике, можно вычислить оптимальное значение задачи для любого многошагового процесса.

Проведенные выкладки можно сопроводить следующими нестрогими рассуждениями. Как показано выше, проводить оптимизацию многошагового процесса по отдельным шагам независимо нельзя, так что придется проводить ее по всем шагам в совокупности. Однако, находясь на некотором промежуточном шаге, нельзя повлиять на итоге уже проведенных шагов. Таким образом, придется оптимизировать текущий шаг (с экономическим эффектом z_i) с учетом всех последующих шагов, пытаясь предусмотреть их итоге (экономический эффект B_i). При этом начинать оптимизацию необходимо с последнего шага, поскольку он не имеет последующих и позволяет провести расчеты наиболее просто.

Последовательное проведение подобных рассуждений приводит к установлению принципа оптимальности, лежащего в основе метода ДП решения управления многошаговыми процессами.

2.5.2.2 Принцип оптимальности Беллмана

Согласно Р. Беллману, основной принцип оптимальности управления многошаговыми процессами может быть словесно выражен следующим образом: «Оптимальное поведение обладает тем свойством, что, каковы бы ни были исходное состояние и первоначальное решение, последующие решения должны составлять оптимальное поведение относительно состояния, получающегося в результате первоначального решения». Иными словами, любой участок оптимальной траектории, в том числе и завершающий, также является оптимальным, а ошибки в управлении, приводящие к отклонениям от оптимальной территории, впоследствии не могут быть исправлены. Конечно, столь общее положение не может быть

непосредственно применено к решению задач ДП и нуждается в конкретизации.

Для строгой формулировки принципа оптимальности введем, как и выше, ряд вспомогательных функций $B_0(x_0), B_1(x_1), \dots, B_N(x_N)$. Функции $B_i(x_i), i = 0,1,2,\dots,N$, имеют важный экономический смысл и представляют собой максимальные значения (рассмотрим для определенности случай задачи на поиск максимума) сумм частных целевых функций

$$z_{i+1}\big(x_i, u_{i+1} + \dots + z_N(x_{N-1}, u_N)\big).$$

(35)

Вычисляемые по всем допустимым «укороченным» наборам управлений $(u_{i+1}, \dots, u_N)$. Иными словами, $B_i(x_i)$ – условно-оптимальное значение целевой функции при переводе системы из состояния x_iпосле шага с номером i в конечное состояние x_N; условность оптимального значения состоит в том, что оно относится не ко всему процессу, а к его заключительной части, и зависит от выбора состояния x_i, являющегося начальным для «укороченного» процесса. Тем самым функции $B_i(x_i)$, называемые функциями Беллмана, характеризуют экстремальные свойства управляемой системы S на последних шага процесса. При этом имеет место просто и важное соотношение

$$B_N(x_N) = 0,$$

справедливое по той причине, что состояние x_N уже является конечным, дальнейших изменений состояний системы не происходит, и соответствующий экономический эффект равен 0.

Принцип оптимальности Р. Беллмана, лежащий в основе метода ДП решения задач, выражается следующим основным функциональным уравнение:

$$B_{i-1}(x_{i-1}) = \max_{u_1}\{z_i(x_{i-1}, u_i) + B_i(x_i) | x_i = f_i(x_{i-1}, u_i)\},$$

(36)

в котором индекс i изменяется по номерам всех шагов процесса в обратном порядке: $i = N, N-1, N-2, \dots, 2,1$.

По своей структуре функциональное уравнение Беллмана является реккурентным. Это означает, что в последовательности функций $B_0(x_0), B_1(x_1), \dots, B_N(x_N)$ каждая предшествующая выражается через последующую.

Важно подчеркнуть, что при вычислении максимума в функциональном уравнении Беллмана для каждого фиксированного значения x_{i-1}одновременно с $B_{i-1}(x_{i-1})$ определяется то значение переменной u_i (одно или несколько) , для которых этот максимум достигается. Это значение зависит от состояния x_{i-1}, и обозначать будем через $u_i\ (x_{i-1})$ символ «~», называемый «тильда», обозначает некоторую условность или приближенность). Фактически $u_i\ (x_{i-1})$ является (возможно, многозначной) функцией, называемой условно-оптимальным управлением (условность заключается в зависимости управления от выбора состояния x_{i-1}). И хотя функции $u_i\ (x_{i-1})$ явно не фигурируют в уравнении (1.16), они играют не менее важную роль, чем функции Беллмана $B_{i-1}(x_{i-1})$, и используются для окончательного построения расчетов последовательно вычисляются и запоминаются условно-оптимальные управления $u_N\,(x_{N-1}), u_{N-1}(x_{N-2}), \dots, u_1\ (x_0)$.

Следует заметить, что принцип оптимальности Беллмана рассматривает конкретную решаемую задачу с оптимальным значения $B_0(x_0)$ не обособленно, а как представителя семейства подобных ей задач с оптимальными значениями $B_1(x_1), \dots, B_N(x_N)$ меньшей размерности, т.е. более простых. Между задачами этого семейства существует связь, которая описывается функциональным уравнением (36) и позволяет, начиная с простейшей функции $B_N(x_N) = 0$, последовательно вычислить все остальные функции $B_{N-1}(x_{N-1}), \dots, B_0(x_0)$, т.е. фактически получить решение всего семейства задач. Данный прием, заключающийся в замене задачи семейством однотипных задач, в результате решения которых находится решение исходной задачи, называется принципом инвариантного погружения.

Полностью аналогичное выражение имеет принцип оптимальности и для решения задач на минимум, при этом в функциональном уравнении (36) обозначение максимума «max» просто меняется на обозначение минимума «min».

2.5.2.3 Метод динамического программирование и его основные этапы

Рассмотренный принцип оптимальности лежит в основе метода ДП решения задач управления многошаговыми процессами. Метод ДП включает три основных этапа:

1. предварительный этап;
2. этап условной оптимизации;
3. этап безусловной оптимизации.

Предварительный этап проводится с целью уменьшения вычислительной работы на последующем этапе решения и, по существу, заключается в нахождении всех допустимых значений управления u_i и фазовых переменных x_i (т.е. фактически областей определения функций $B_i(x_i)$ или, в более сложных случаях, множеств, содержащих эти области определения). Иными словами, на данном этапе отбрасываются все заведомо неподходящие, нереализуемые значения фазовых и управляющих переменных. Проводится предварительный этап в естественном порядке от первого шага к последнему: $i = 1,2,\dots,N$, а опираются соответствующие расчеты на уравнение процесса $x_i = f_i(x_{i-1}, u_i)$. Данный этап особенно удобен при табличном способе задания функций, фигурирующих в условии задачи.

Этап условной оптимизации заключается в непосредственном вычислении функций Беллмана $B_i(x_i)$ и проводится, как и предписывает принцип оптимальности, в обратном порядке от последнего шага к первому: $i = N, N-1,\dots,2,1$. Расчет проводится следующим образом. Для последнего шага

при $i = N$ с учетом условия $B_N(x_N) = 0$принцип оптимальности Беллмана принимает следующий наиболее простой вид:

$$B_{N-1}(x_{N-1}) = \max_{u_N}(x_{N-1}, u_N)\}. \quad (37)$$

Иначе говоря, при планировании последнего шага нет необходимости учитывать прогноз на будущее. При этом для каждого допустимого значения аргумента x_{N-1} (определенного на предварительном этапе) максимум достигается при некотором управлении $u_N = u_N(x_{N-1})$. Вычисленная функция $B_{N-1}(x_{N-1})$ позволяет перейти к предшествующему шагу при $i = N - 1$ и снова применить принцип оптимальности – он уже не будет столь простую форму записи. Продолжая аналогичным образом, завершим данный этап вычислением функций $B_0(x_0)$ и $u_1(x_0)$ после прохождения первого шага при $i = 1$.

Этап безусловной оптимизации проводится с целью окончательного вычисления оптимального значения задачи Z и построения оптимального управления $(u_1, u_2, \dots, u_N)$ и оптимальной траектории $x_0, x_1, \dots, x_N$. Проводится данный этап в естественном порядке от первого шага к последнему: $i = 1,2, \dots, N$. Построение оптимального решения начинается с определения начального состояния x_0. Если начальное состояние определено однозначно, то оптимальное значение задачи равно $B_0(x_0)$; при этом нужно принять $x_0 = x_0$. Если же начальное состояние x_0 не определено однозначно, а принимает значения из некоторого множества начальных состояний X_0, то оптимальное значение задачи вычисляется по формуле:

$$Z = \max_{x_0 \in X_0}\{B_0(x_0)\}. \quad (38)$$

В этом случае в качестве x_0 принимаем то значение переменной x_0, при котором данный максимум достигается (таких значений может быть одно или несколько).

При построении оптимального управления и оптимальной траектории используются функции $u_i(x_{i-1})$, вычисленные на этапе условной

оптимизации. На первом шаге при $i = 1$, используя уже известное значение x_0 , находим:

$$u_1 = u_1(x_0), x_1 = f_1(x_0, u_1). \quad (39)$$

На втором шаге при $i = 2$, используя вычисленное x_1 , находим:

$$u_2 = u_2(x_1), x_2 = f_2(x_1, u_2) \quad (40)$$

Продолжая аналогично, получим на последнем шаге при $i = N$:

$$u_N = u_N(x_{N-1}), x_N = f_N(x_{N-1}, u_N). \quad (41)$$

Таким образом, полностью определяются оптимальные решение $(u_1, u_2, \dots, u_N)$ и оптимальная траектория $x_0, x_1, \dots, x_N$ системы. Отметим, что и оптимальное решение, и оптимальная траектория могут быть определены неоднозначно. Важно, что функция Беллмана $B_i(x_i)$ не участвует в построении оптимального решения задачи и, следовательно, не требует длительного хранения в памяти при реализации метода ДП на ЭВМ.

Тем самым, в ходе решения задачи методом ДП весь многошаговый процесс просчитывается три раза в переменных направлениях.

Отметим следующие основные достоинства метода ДП:

1. сравнительная простота и однотипность расчетов, что является удобным для алгоритмизации и программирования задач при их решении;

2. снижение трудоемкости решения задач за счет более полного использования свойств управляемых систем;

3. отсутствие специальных ограничений на природу, характер и свойства функций f, z. Они, например, могут не являться линейными, выпуклыми, непрерывными, дифференцируемыми, и могут быть заданы как таблично, так и аналитически, т.е. в виде формул

Обладая несомненными достоинствами, метод ДП не лишен и отдельных недостатков, основным из которых является необходимость хранения большого объема промежуточной информации. Действительно, на этапе безусловной оптимизации используются условно-оптимальные

управления $u_1(x_0), u_2(x_1), \ldots, u_N(x_{N-1})$, вычисляемые и запоминаемые на предшествующем этапе условной оптимизации.

2.5.3 Применение метода динамического программирования в решении прикладных задач

К особенностям математической модели динамического программирования относятся:

1. формализация задачи оптимизации в качестве итогового многошагового процесса управления;

2. определение критерия оптимальности операции или показателя эффективности целевой функцией, являющейся аддитивной от любого шага оптимизации;

3. зависимость выбора управляющего воздействия x_k на каждом шаге исключительно от состояния самой системы на этом шаге с учетом отсутствия влияния на предшествующие шаги (отсутствие обратной связи);

4. состояние системы S_k после каждого шага управления зависит исключительно от предшествующего состояния данной системы и управляющего воздействия x_k (нет последействия), причем оно может быть сохранено в виде уравнения состояния системы;

5. зависимость управления x_k на каждом шаге от конечного числа управляющих переменных, а состояния системы S_k – от конечного числа параметров; определение оптимального управления (вектора X) как последовательности оптимальных управлений, число которых устанавливает количество шагов задачи. Основные свойства задач, в которых можно применять метод динамического программирования: задача должна допускать интерпретацию как многошаговый процесс принятия решения; задача должна быть определена для любого числа этапов (шагов) и иметь структуру, не зависящую от их числа;

6. при рассмотрении задачи на каждом шаге должно быть задано множество параметров, описывающих состояние системы;

2.5.4 Общая постановка задачи оптимального распределения ресурсов

Задача формулируется следующим образом.

Имеется определенное количество ресурсов, к которым относятся денежные ресурсы и материальные ресурсы (трудовые ресурсы, сырье, оборудование и т. п.).

Указанные ресурсы требуется распределить между несколькими объектам их использования, разделяя на отдельные промежутки планового периода или части имеющихся ресурсов по различным объектам таким образом, чтобы в итоге получилось оптимальное распределение, гарантирующее максимальную суммарную эффективность.

В качестве показателей эффективности можно использовать полученную прибыль, суммарные затраты или объемы готовой продукции.

Описание задачи оптимального распределения ресурсов.

В течении n лет между p предприятиями необходимо распределить имеющееся начальное количество ресурсов (средств) S_0. Выделив в k-ом году -му предприятию средства $x_{ki}(k=1,n;\ i=1,p)$, предприятие получит прибыль в размере $f_{ki}(x_{ki})$ и к концу года ресурсы возвращаются в количестве $g_{ki}(x_{ki})$. На следующем этапе распределения полученная прибыль может быть использована полностью или частично, либо не использована вовсе. Требуется определить оптимальный способ распределения ресурсов (средств), которые следует выделить предприятию в *k*-ом году, чтобы суммарная прибыль от *k* предприятий за *n* лет была максимальной.

Таким образом, показателем эффективности всего процесса распределения будем считать суммарную прибыль от *k* предприятий за *n* лет:

$$Z = \sum_{i=1}^{p} \sum_{k=1}^{n} f_{ki}(x_{ki}), \qquad (42)$$

Величина S_{k-1} характеризует количество ресурсов в начале k-го года, т.е. является параметром состояния системы. Управлением на k-ом шаге является выбор переменных $x_{k1}, x_{k2}, \dots, x_{kp}$, характеризующих средства, выделяемые k-му предприятию. Предположим, что прибыль на k-ом этапе распределения не используется. Тогда уравнение состояния имеет вид:

$$S_k = S_{k-1} - \sum_{i=1}^{p} x_{ki} + \sum_{i=1}^{p} g x_{ki}, \quad (43)$$

Если же некоторая часть прибыли используется на следующем этапе распределения, то к правой части уравнения прибавляется полученная величина прибыли.

Необходимо определить np переменных x_{ki}удовлетворяющих условиям (42) и максимизирующих функцию (41). Вычислительная схема динамического программирования начинается с определения функции $Z_k\ (S_{k-1})$, обозначающей прибыль, начиная с k-го года до конца текущего периода, при оптимальном разделении средств между p предприятий, если в k-ом году распределялось S_{k-1}средств. Функции $Z_k\ (S_{k-1})$ для $k = 1,2, \dots, n-1$удовлетворяют рекуррентным соотношениям, которые примут вид:

$$Z_k\ S_{k-1} = \max_{0 \le \sum_{i=1}^{p} x_{ki} \le S_{k-i}} \sum_{i=1}^{p} f_{ki} S_{k-1}, x_k + Z_k\ (S_k) \quad (44)$$

При $k = n$ согласно (1) получаем:

$$Z_n\ S_{n-1} = \max_{0 \le \sum_{i=1}^{p} x_{ki} \le S_{k-i}} \sum_{i=1}^{p} f\ \ (x_n) \quad (45)$$

Процесс поиска решения состоит в последовательном вычислении уравнений (3.4) и (3.3) для всех допустимых состояний системы S_k, *(k = n – 1, n – 2, … , 1).* Причем каждое уравнение зависит от *p* переменных и является задачей оптимизации функции.

В итоге задача с *np* переменными сводится к ряду из *n* задач, каждая из которых содержит *p* переменных.

2.5.5 Решение задач оптимального распределения ресурсов

На период лет составляется план работы двух предприятий (*p=2)*. Имеется определенное количество средств, которые составляют S_0. Средства в размере x, выделенные предприятию П1, к концу года приносят прибыль в количестве $f_1(x)$ и возвращаются в количестве $g_1(x) < x$, аналогично для предприятия П2, приносят прибыль в количестве $f_2(x)$ и возвращаются в количестве $g_2(x) < x$. В конце года оставшиеся ресурсы снова перераспределяются между предприятиями П1 и П2, полученный доход в предприятия не вкладываются, новые средства не поступают.

Требуется выбрать оптимальный способ распределения начальных средств на *n* лет.

Рассмотрим процесс распределения ресурсов как многошаговый с количеством шагов n. Номер каждый шаг равен номеру года. Два предприятия с вложенными в них средствами представляют собой управляемую систему. Система имеет один параметр состояния $S_{k-1}(k = 1, n)$ − количество средств, которые нужно распределить в начале k-го года. На каждом шаге имеем две переменные управления: x_{k1} и x_{k2} − количество средств, выделенных предприятию П1 и П2 соответственно. Так как средства ежегодно распределяются полностью, то $x_{k2} = S_{k-1} - x_{k1}(k = 1, n)$. На каждом шаге задача становится одномерной. Обозначим x_{k1} через x_k, тогда:

$$x_{k2} = S_{k-1} - x_k.$$

Показатель эффективности на k-ом шаге равен $f_1 x_k + f_2 S_{k-1} - x_k$ − прибыль полученная от двух предприятий в течении k-го года.

Показатель эффективности всего процесса – это прибыль, полученная от двух предприятий в течении *n* лет, равная:

$$Z = \sum_{k=1}^{n} [f_1(x_k) + f_2(S_{k-1} - x_k)]. \quad (46)$$

Уравнение, характеризующее состояние системы, представляет собой остаток средств S_k после распределения на k-ом шаге и запишется следующим образом:

$$S_k = g_1 x_k + g_2(S_{k-1} - x_k) \quad (47)$$

Введем величину $Z_k\,(S_{k-1})$, которая представляет собой возможную оптимальную прибыль, полученную после распределения средств в размере S_{k-1}между двумя предприятиями, начиная с k-го года до конца планируемого периода. Получим основные оптимальные уравнения для этих функций:

$$Z_n\ S_{n-1} = \max_{0 \le x_n \le S_{n-1}} f_1 x_n + f_2 S_{n-1} - x_n;$$

$$Z_k\ S_{n-1} = \max_{0 \le x_k \le S_{k-1}} f_1 x_k + f_2 S_{k-1} - x_k + Z_{k+1}(S_k), \quad (48)$$

где S_k – находится из уравнения состояния (4.6).

Решаем поставленную задачу для следующих условий:

$$S_0 = 10000; n = 4;$$

$$f_1 x = 0{,}4x; f_2 x = 0{,}3x;$$

$$g_1 x = 0{,}5x; g_2 x = 0{,}8x;$$

Если x_k и $S_{k-1} - x_k$ – количество средств, выделенных предприятиям П1 и П2 в k-м году, то общая прибыль, полученная от двух предприятий, равна: а уравнение состояния принимает вид:

$$S_k = 0{,}5x_k + 0{,}8S_{k-1} - x_k = 0{,}8S_{k-1} - 0{,}3x_k$$

Основные функциональные уравнения (4.7) запишутся следующим образом:

$$Z_4\ S_3 = \max_{0 \le x_4 \le S_2} 0{,}1x_4 + 0{,}3S_3;$$

$$Z_k\ S_{k-1} = \max_{0 \le x_k \le S_{k-1}} 0{,}1x_k + 0{,}3S_{k-1} + Z_{k+1}(0{,}8S_{k-1} - 0{,}3x_k) k = 1{,}2{,}3.$$

I этап. Условная оптимизация.

4-й шаг. Условный оптимальный доход равен:

$$Z_4\ S_3 = \max_{0 \le x_4 \le S_2} 0{,}1x_4 + 0{,}3S_3 = 0{,}4S_3$$

Так как показатель эффективности $Z_4(S|3)$является линейной функцией относительно x_4 и эта переменная входит в выражение со знаком плюс, то данный показатель достигает максимума в конце интервала $0 \le x_4 \le S_3$, т.е. при $x_4 = S_3$.

3-й шаг:

$$Z_3\ S_2 = \max_{0 \le x_2 \le S_2} 0{,}1x_3 + 0{,}3S_2 + 0{,}4(0{,}8S_2 - 0{,}3x_3) = \max_{0 \le x_2 \le S_2} -0{,}02x_3 + 0{,}62S_2 = 0{,}62S_2.$$

Коэффициент при x_3 отрицателен, поэтому максимум в этой линейной функции относительно x_3 достигается в начале интервала $0 \le x_3 \le S_2$, т.е

$$x_3 = 0.$$

2-й шаг:

$$Z_2\ S_1 = \max_{0 \le x_2 \le S_1} -0{,}086x_2 + 0{,}796S_1 = 0{,}796S_1;$$

$$x_2 = 0.$$

1-й шаг:

$$Z_1\ S_0 = \max_{0 \le x_2 \le S_0} -0{,}1388x_1 + 0{,}9368S_0 = 0{,}9368S_0;$$

$$x_1 = 0.$$

Результат условной оптимизации:

$$Z_1 S_0 = 0{,}9368S_0, x_1 = 0;$$
$$Z_2 S_1 = 0{,}796S_1, x_2 = 0;$$
$$Z_3 S_2 = 0{,}62S_2, x_3 = 0;$$
$$Z_4 S_3 = 0{,}4S_3, x_4 = S_3.$$

II этап. Безусловная оптимизация.

Полученное значение эффективности на первом шаге $Z_1 S_0 = 0{,}9368S_0$, зная что начальное количество средств $S_0 = 10000$, то максимальный суммарный доход составит 9368.

Зная $x_1 = 0$, находим $S_1 = 8000$ используя S_1 и $x_2 = 0$ получаем $S_2 = 6400$. Аналогично $x_3 = 0$, $S_3 = 5120$ из чего следует, что $x_4 = 5120$ т.к. $x_4 = S_3$. Получаем что, средства следует распределить как указано в таблице 18.

Таблица 18 – Результат распределения средств

Год	Предприятие	
	П1	П2
1	0	10000

Продолжение таблицы 18

2	0	8000
3	0	6400
4	5120	0

В данной главе была сформулирована постановка задачи оптимального распределения ресурсов. Так же представлена математическая модель данной задачи.

Аналитически решена задача оптимального распределения ресурсов с применением метода динамического программирования.

2.5.6 Реализация алгоритма решения задачи оптимального распределения ресурсов

2.5.6.1 Методика вычисления оптимального значения задачи

У нас есть p предприятий и n лет. Каждое предприятие имеет прибыль profits[company][year] и затраты costs[company][year] на производство в каждый из n лет. Нам также известно общее количество ресурсов total_resources.

Общая идея:

1. используем метод динамического программирования для пошагового решения задачи.

2. для каждого года и количества ресурсов рассмотрено каждое предприятие и решаем, стоит ли использовать его в текущей ситуации.

3. сохраняем максимальную прибыль и путь выбранных предприятий для каждой комбинации года и ресурсов.

4. в конце концов, находим максимальную прибыль и выбранные предприятия для максимальной прибыли в последнем году.

2.5.6.2 Цель

Максимизировать общую прибыль, учитывая ограничение на общее количество ресурсов.

2.5.6.3 Инициализация

Создаем трехмерный массив dp, где dp[year][resources] хранит максимальную прибыль и выбранные предприятия для каждого года и количества ресурсов.

Изначально все значения dp устанавливаются в 0.

```
dp = [[[0, []] for _ in range(total_resources + 1)] for _ in range(n + 1)]
```

Итерация по годам и ресурсам:

1. для каждого года и количества ресурсов перебираем предприятия.

2. проверяем, можно ли использовать текущее предприятие в текущем году с текущим количеством ресурсов.

```
for year in range(1, n + 1):
    for resources in range(total_resources + 1):
        for company in range(p):
            if resources >= costs[company][year - 1]:
                # Вычисление максимальной прибыли с использованием текущего предприятия
                current_profit = profits[company][year - 1] + dp[year - 1][resources - costs[company][year - 1]][0]

                # Обновление dp, если текущая прибыль больше
                if current_profit > dp[year][resources][0]:
                    dp[year][resources] = [current_profit, dp[year - 1][resources - costs[company][year - 1]][1] + [company + 1]]

            # Если не используем текущее предприятие
            if dp[year][resources][0] < dp[year - 1][resources][0]:
                dp[year][resources] = dp[year - 1][resources]
```

Нахождение максимальной прибыли в последнем году:

max_profit = dp[n][total_resources][0]

Нахождение выбранных предприятий для каждого года:

Восстановление пути выбранных предприятий.

chosen_companies_each_year = dp[n][total_resources][1]

2.5.6.4 Пример

Давайте рассмотрим простой пример:

Шаг 1: Объявление переменных

```
profits = [
   [10, 20, 15],
   [5, 10, 25],
   [15, 5, 20]
]
costs = [
   [2, 5, 3],
   [1, 3, 4],
   [4, 2, 2]
]
total_resources = 15
p = 3
n = 3
dp = [[[0, []] for _ in range(total_resources + 1)] for _ in range(n + 1)]
```

Шаг 2: Итерация по годам и ресурсам

Год 1: Рассмотрим первый год (year = 1), у нас есть 15 ресурсов.

```
for resources in range(16):
   for company in range(3):
```

Для предприятия A (company=0):

Проверяем, можно ли использовать A в этот год с текущими ресурсами.

Считаем прибыль и сравниваем с текущей максимальной прибылью.

Обновляем dp[1][resources], если прибыль больше.

Повторяем для предприятий B и C.

В результате получаем обновленный dp[1][resources] для всех возможных значений ресурсов.

Год 2 и Год 3: Повторяем тот же процесс для годов 2 и 3, используя информацию о предыдущих годах.

```
for year in range(2, 4):
    for resources in range(16):
        for company in range(3):
```

Шаг 3: Нахождение максимальной прибыли в последнем году

```
max_profit = dp[3][15][0]
```

Шаг 4: Нахождение выбранных предприятий для каждого года

```
chosen_companies_each_year = dp[3][15][1]
```

Результаты:

max_profit равен максимальной прибыли в последнем году, который в данном случае равен 65.

chosen_companies_each_year содержит список предприятий для каждого года, что означает, что оптимальное распределение ресурсов выглядит так: [2, 3, 1].

Объяснение результатов:

В первом году оптимально использовать второе предприятие (B) с прибылью 20 и стоимостью 5.

Во втором году оптимально использовать третье предприятие (C) с прибылью 25 и стоимостью 4.

В третьем году оптимально использовать первое предприятие (A) с прибылью 15 и стоимостью 3.

Таким образом, при заданных ограничениях на ресурсы, программа выбрала оптимальные предприятия для каждого года, максимизируя общую прибыль в конце периода.

На выходе из программы получено оптимальное распределение ресурсов между предприятиями на каждый год так, чтобы максимизировать общую прибыль в конце периода.

Все промежуточные значения хранятся в матрице dp, что позволяет отслеживать пути, ведущие к максимальной прибыли.

2.6 План разработки проекта

Таблица 19 – Календарный план разработки проекта

№	Содержание работы	Результат	Начало	Окончание	Исполнитель
1	Сбор информации	Актуальная подборка информации	11.10.23	16.10.23	Сабурова С.П.
2	Постановка задачи	Математическое описание задачи	16.10.23	20.10.23	Своеволин И.С
3	Определение необходимого для решения функционала	Готовое алгоритмическое описание	20.10.23	3.11.23	Трофимов В.О.
4	Написание программы	Готовый код	3.11.23	20.11.23	Сайфуллин И.К.

Продолжение таблицы 19

5	Тестирование программы	Скрины итоговых расчетов в предполагаемых “тонких” местах	20.11.23	25.11.23	Кудрявцев А.Г.
6	Итоговая подготовка отчета	Готовый отчет	25.11.23	3.12.23	Сабурова С.П.

Решение данной администрировалось посредством приложения “Ежедневник” с общей timeline и выведением checkpoint. Также была применена доска MIRO

2.7 Выводы по разделу 2

1. Был написан программный код для численного решения задач;
2. Код был протестирован на логические уязвимости;
3. Проект администрировался с динамическим отслеживание выполнения работ, что помогло придерживаться графика;
4. Решение задачи оптимально.

3 РЕЗУЛЬТАТЫ РАБОТЫ

3.1 Задача управления гибридной системы

На рисунках виден результат работы программы:

1. Расчет оптимальной траектории в случае двух целей, заданных скоростей, не заданных углов – рисунок 6;

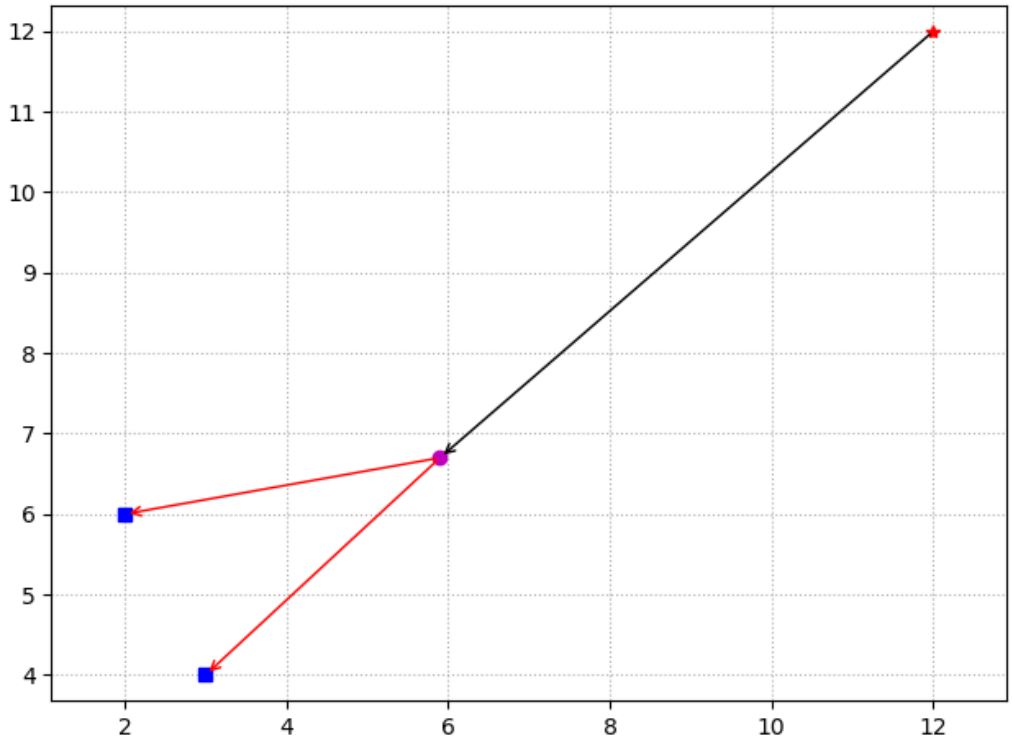

Рисунок 6 – Две цели без учета скоростей и углов

2. Расчет оптимальной траектории в случае десяти целей, заданных скоростей, не заданных углов – рисунок 7;

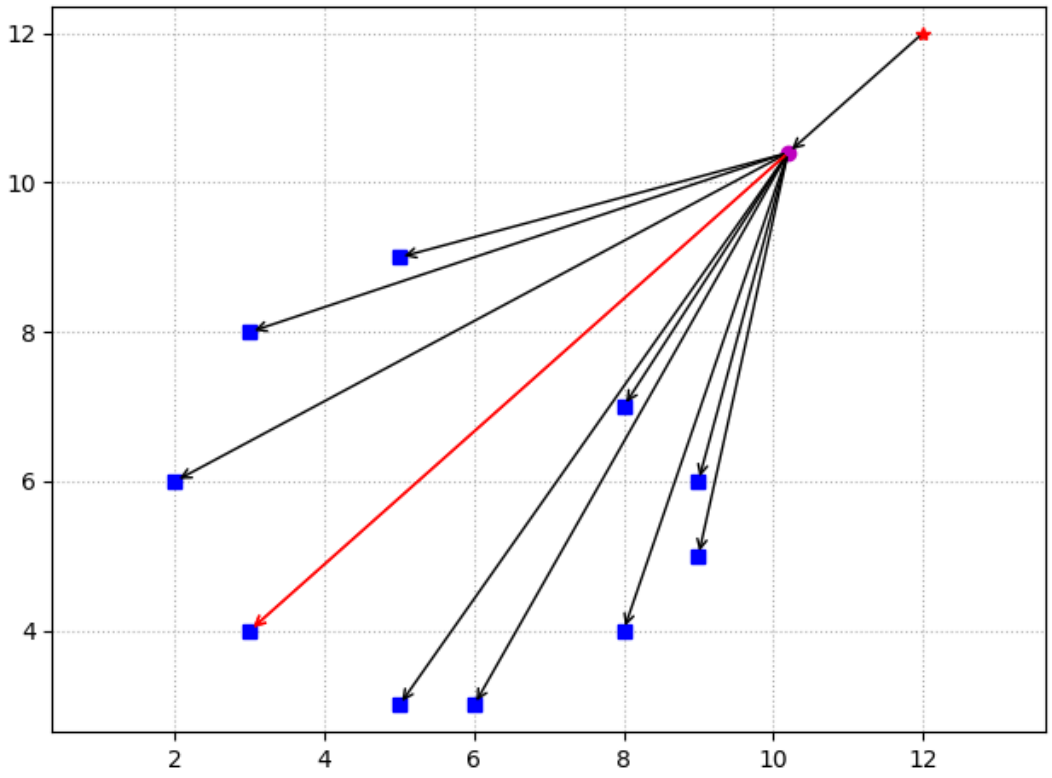

Рисунок 7 – Десять целей без учета углов

3. Расчет оптимальной траектории в случае двух целей, заданных скоростей и углов – рисунок 8;

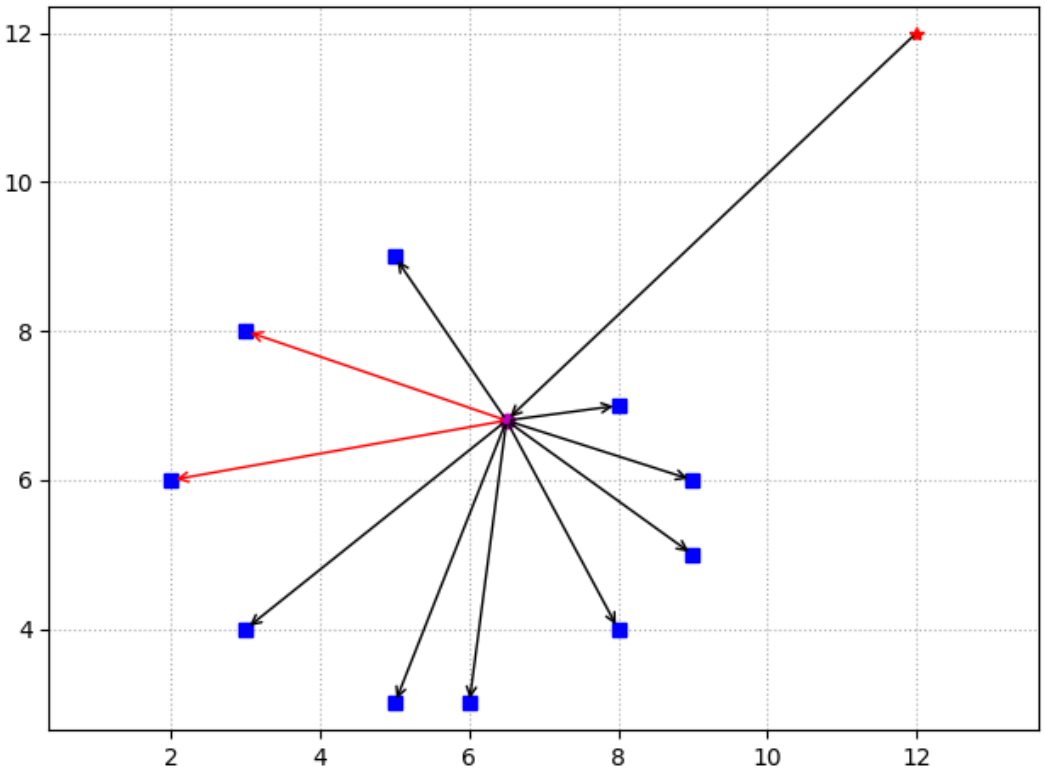

Рисунок 8 – Десять целей с учетом скоростей и углов

3.2 Задача о замене оборудования

На рисунках представлен результат работы программы:

1. Входные данные r = [8, 7, 7, 6, 6, 5, 5], u = [1, 2, 1, 2, 2, 3, 2], s = [12, 10, 8, 8, 7, 6, 4], p=16, t_0=1, planned_period=6, результат на рисунке 9.

```
 Матрица максимальных прибылей:
[(30, 'С'), (28, 'С'), (27, 'С'), (25, '3'), (24, '3'), (23, '3'), (21, '3')]
[(26, 'С'), (23, '3'), (23, 'С'), (21, '3'), (20, '3'), (19, '3'), (17, '3')]
[(22, 'С'), (19, 'С'), (17, 'С'), (17, '3'), (16, '3'), (15, '3'), (13, '3')]
[(18, 'С'), (15, 'С'), (14, 'С'), (11, '3'), (10, '3'), (9, '3'), (7, '3')]
[(12, 'С'), (11, 'С'), (10, 'С'), (8, 'С'), (6, 'С'), (5, 'С'), (2, '3')]
[(7, 'С'), (5, 'С'), (6, 'С'), (4, 'С'), (4, 'С'), (2, 'С'), (3, 'С')]

 Оптимальная стратегия:
|1 год: Сохранить| -> |2 год: Сохранить| -> |3 год: Заменить| -> |4 год: Сохранить| -> |5 год: Сохранить| -> |6 год: Сохранить|
```

Рисунок 9

2. Входные данные r = [10, 9, 8, 7, 6, 5, 4], u = [2, 2, 3, 2, 2, 4, 3], s = [13, 11, 9, 9, 8, 5, 3], p=16, t_0=1, planned_period=6, результат на рисунке 10

```
Матрица максимальных прибылей:
[(37, 'С'), (34, 'С'), (32, 'С'), (32, 'З'), (31, 'З'), (28, 'З'), (26, 'З')]
[(31, 'С'), (29, 'З'), (27, 'З'), (27, 'З'), (26, 'З'), (23, 'З'), (21, 'З')]
[(26, 'С'), (23, 'С'), (21, 'С'), (21, 'З'), (20, 'З'), (17, 'З'), (15, 'З')]
[(20, 'С'), (18, 'З'), (16, 'З'), (16, 'З'), (15, 'З'), (12, 'З'), (10, 'З')]
[(15, 'С'), (12, 'С'), (10, 'С'), (9, 'С'), (8, 'З'), (5, 'З'), (3, 'З')]
[(8, 'С'), (7, 'С'), (5, 'С'), (5, 'С'), (4, 'С'), (1, 'С'), (1, 'С')]

Оптимальная стратегия:
|1 год: Сохранить| -> |2 год: Заменить| -> |3 год: Сохранить| -> |4 год: Заменить| -> |5 год: Сохранить| -> |6 год: Сохранить|
```

Рисунок 10

3.3 Задача о рюкзаке

Графики при разных значениях вместимости:

1. capacity = 35, values =[6 9 6 1 1], weights = [2 13 8 14 7], результат на рисунке 11;

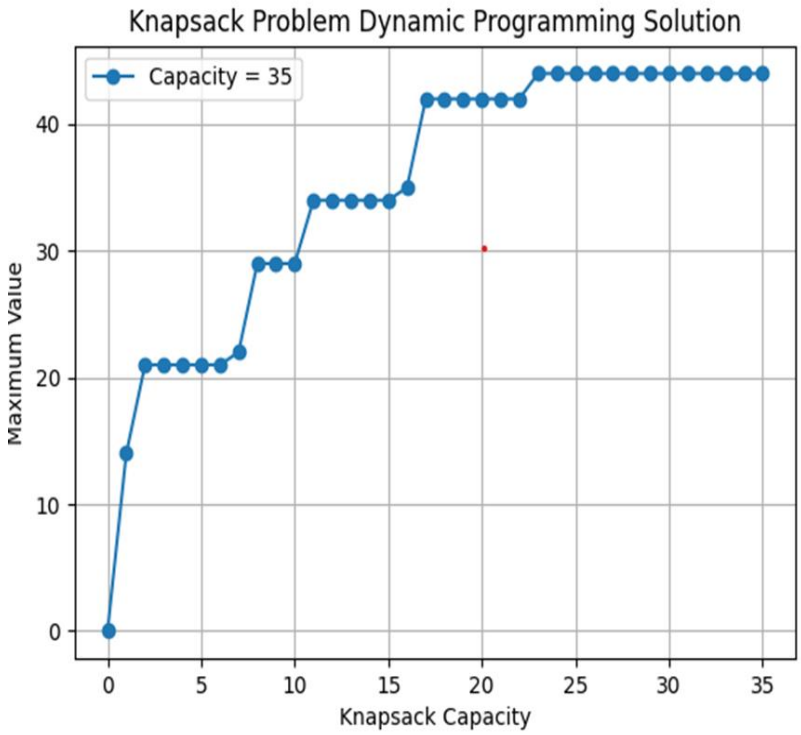

Рисунок 11 – график для вместимости 35

2. capacity = 40, values =[4 7 7 1 9 5 8 1 1 8], weights = [13 16 18 8 17 13 14 12 2 16], результат на рисунке 12;

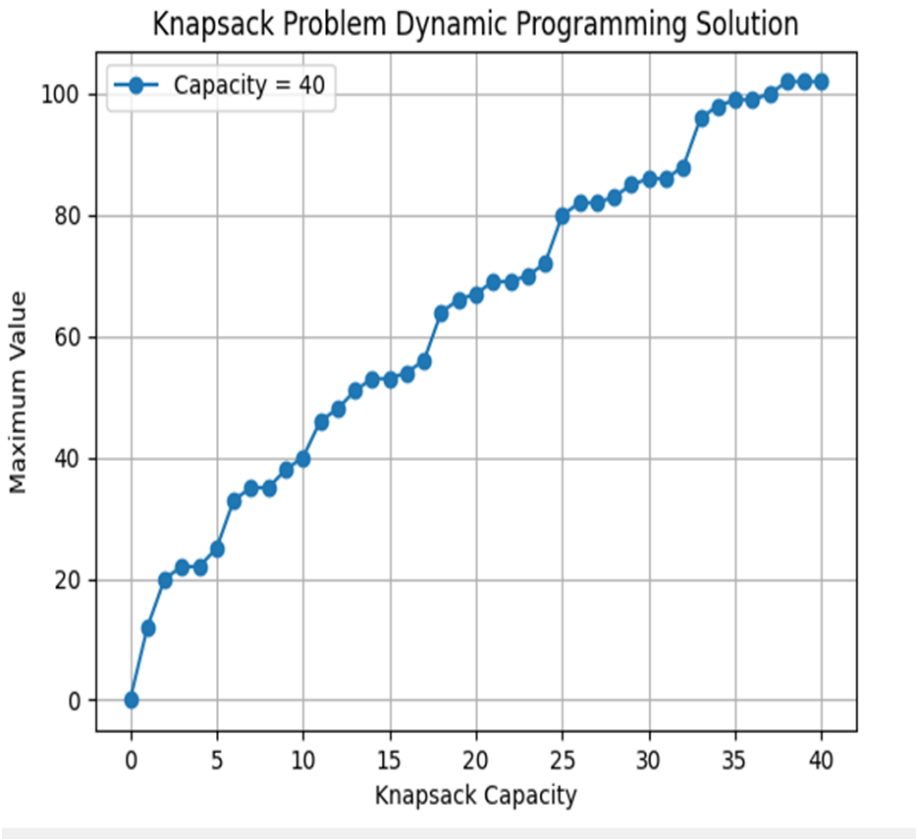

Рисунок 12 – график для вместимости 40

3. capacity = 46, values =[4 7 7 1 9 5 8 1 1 8 2 6 8 1 2], weights = [16 19 10 11 10 10 2 19 8 17 15 6 1 17 5], результат на рисунке 13.

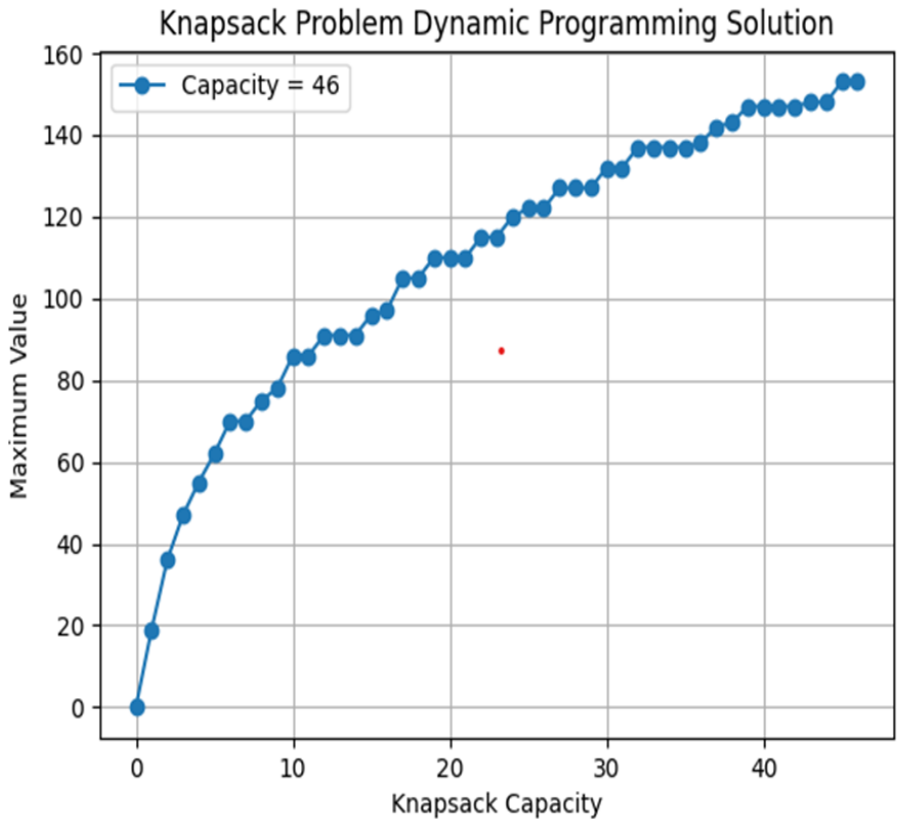

Рисунок 13 – график для вместимости 46

3.4 Задача о распределении инвестиций

Результат работы программы для различных входных данных

На рисунке 13 представлен вывод программы вместе с выходными данными.

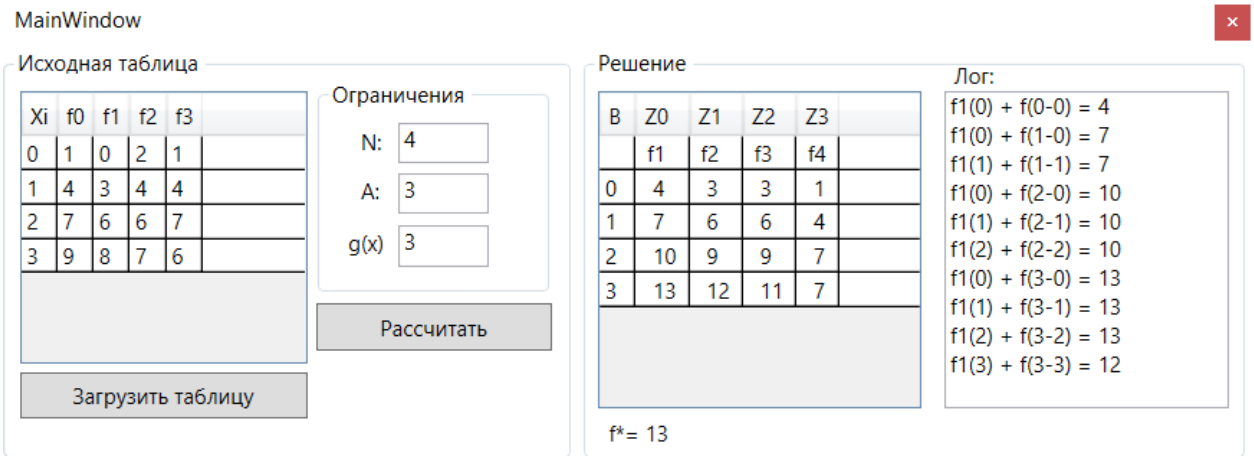

Рисунок 13 – вывод программы

На рисунке 14 представлен вывод программы вместе с выходными данными.

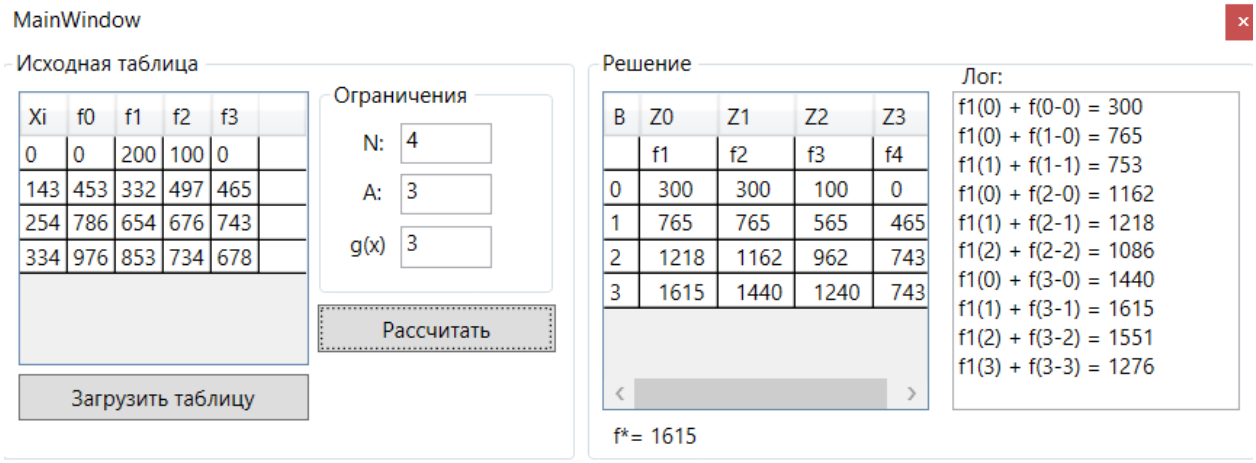

Рисунок 14 – вывод программы

На рисунке 15 представлен вывод программы вместе с выходными данными.

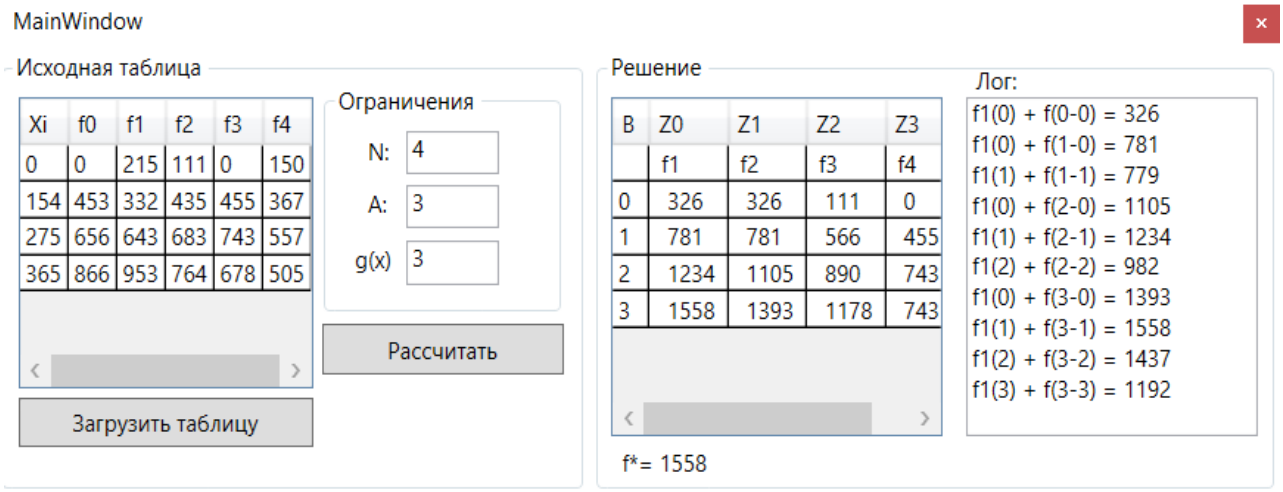

Рисунок 15 – вывод программы

3.5 Задача о распределении ресурсов

Результат работы программы для различных входных данных

Входные данные: 3 года, 3 предприятия, 15 ресурсов.

В третьем столбце прибыль предприятий по годам [1год, 2год, 3год].

Программа выбирает оптимальные предприятия на три года – [1год – 3 предприятие, 2 год – 1 предприятие, 3 год – 2 предприятие]. Результат на рисунке 16

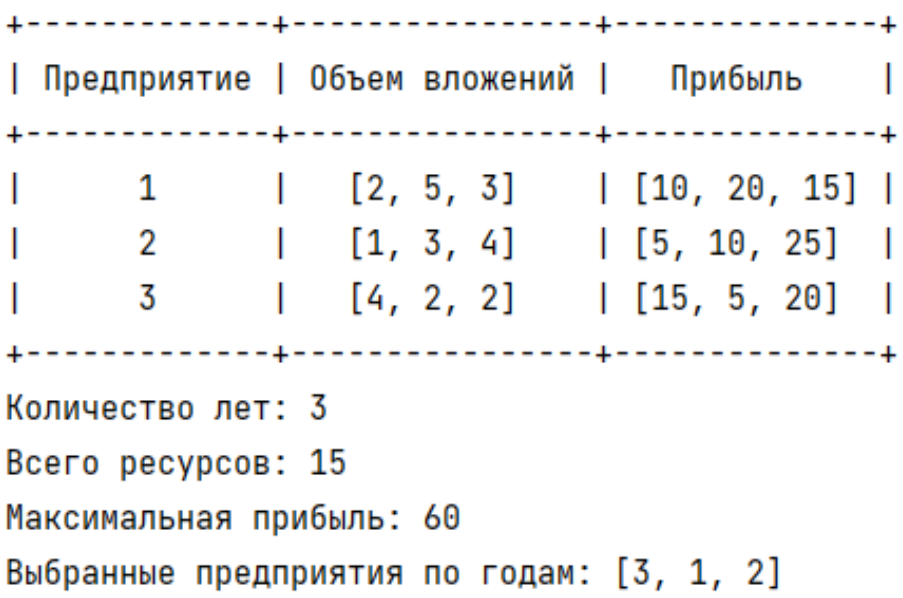

Рисунок 16 – вывод программы

На рисунке 17 вывод программы с другими входными данными. Порядок по аналогии с рисунком 16.

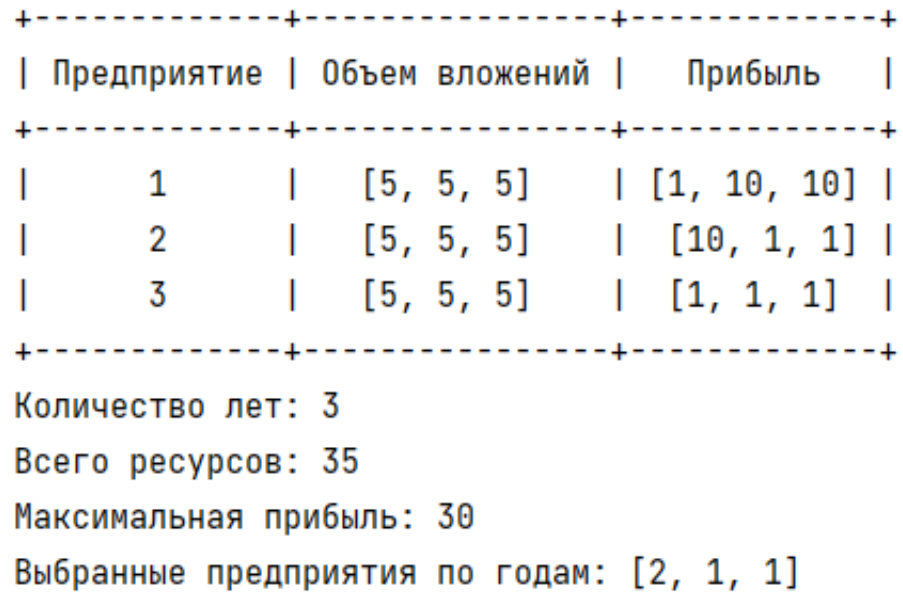

Рисунок 17 – вывод программы

3.6 Выводы по разделу 3

Задача о замене оборудования: создан алгоритм, который позволяет определить оптимальную стратегию замены оборудования, что может быть критично для бизнеса в условиях ограниченных ресурсов и стремления к максимизации прибыли.

Задача о распределении ресурсов: алгоритм позволяет эффективно распределять ресурсы с учетом различных критериев, таких как прибыль, стоимость и другие, что является важным аспектом для управления производственными процессами.

Задача о рюкзаке: предложили алгоритм для решения классической задачи о рюкзаке, которая может быть полезна в различных областях, таких как логистика, управление запасами и другие.

Задача о распределении инвестиций: разработанный код позволяет оптимально распределять инвестиции, учитывая различные факторы и ожидаемые результаты.

Задача об управлении гибридной системой: создан алгоритм для эффективного управления гибридной системой, что имеет значение в контексте решения логистических и транспортных задач.

ЗАКЛЮЧЕНИЕ

В заключении представляются общие выводы и достигнутые результаты. Оценивается степень достижения поставленной цели. Характеризуется производительность команды в целом и отдельных участников. Раскрываются перспективы дальнейшего развития выполненного проекта. (дальнейшее усложнение задачи возможно в виде усложнение алгоритмов движения, например, модель Дубинса или его модификация, или расширение размерности пространствам (модель пространственного движения для авиации/БПЛА))

В целом, исследование позволяет утверждать, что динамическое программирование эффективно применимо в различных областях и способно решать сложные оптимизационные задачи. Полученные результаты могут служить основой для дальнейших исследований и разработок в области оптимизации бизнес-процессов и принятия стратегических решений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Каляев И.А., Гайдук А.Р., Капустян С.Г. Модели и алгоритмы коллективного управления в группах роботов. М.: Физматлит, 2009.

2. Куржанский А.Б. Задача управления групповым движением. Общие соотношения // Докл. РАН. 2009. Т. 426. № 1. С. 20–25.

3. Бортаковский А.С. Быстродействие группы управляемых объектов // Изв. РАН. ТиСУ. 2023. № 5. С. 51–77.

4. Евдокименков В.Н., Красильщиков М.Н., Оркин С.Д. Управление смешанными группами пилотируемых и беспилотных летательных аппаратов в условиях единого информационно-управляющего поля. М.: Изд-во МАИ, 2015.

5. Гончаренко В. И., Желтов С. Ю., Князь В.А., Лебедева Г.Н., Михайлина Д.А., Царева О.Ю. Интеллектуальная система планирования групповых действий беспилотных летательных аппаратов при наблюдении наземных мобильных объектов на заданной территории // Изв. РАН. ТиСУ. 2021. № 3. С. 39–56.

6. Tsourdos A., White B., Shanmugavel M. Cooperative Path Planning of Unmanned Aerial Vehicles. N. Y.: Wiley&Sons, 2011.

7. Jia Zeng, Xiaoke Yang, Lingyu Yang, Gongzhang Shen. Modeling for UAV Resource Scheduling Under Mission Synchronization // J. Systems Engineering and Electronics. 2010. V. 21. № 5. P. 821–826.

8. Babel L. Coordinated Target Assignment and UAV Path Planning with Timing Constraints // J. Intelligent & Robotic Systems. 2019. V. 94 (3-4). P. 857–869.

9. Poudel S., Moh S. Task Assignment Algorithms for Unmanned Aerial Vehicle Networks:
A Comprehensive Survey // Vehicular Communications. 2022. V. 35. P. 100469.

10. Бузиков М.Э., Галяев А.А. Перехват подвижной цели машиной

Дубинса за кратчайшее время // АиТ. 2021. № 5. С. 3–19.

11. Галяев А.А., Рубинович Е.Я. Планирование движения подвижных объектов в конфликтной среде // Аналитическая механика, устойчивость и управление: Тр. XI Междунар. Четаевской конф. (пленарные доклады). Казань: Изд-во КНИТУ-КАИ, 2017. С. 71–90.

12. Mohsan S. A. H., Othman N. Q. H., Li Y. et al. Unmanned Aerial Vehicles (UAVs): Practical Aspects, Fpplications, Open Challenges, Security Issues, and Future Trends. Intel Serv Robotics. 2023. V. 16. P. 109–137.

13. Марков А.А. Несколько примеров решения особого рода задач о наибольших и наименьших величинах // Сообщения Харьк. мат. общества. Сер. 2. Т. I. 1889. С. 250–276.

14. Dubins L.E. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents // American. Mathematics. 1957. V. 79. № 3. P. 497–516.

15. Isaacs R. Games of Pursuit // Scientific Report of the RAND Corporation. Santa Monica, 1951.

16. Reeds J.A., Shepp L.A. Optimal Paths for a Car that Goes Both Forwards and Backwards // Pacific J. Math. 1990. V. 145. № 2. P. 367–393.

17. Бердышев Ю.И. Об оптимальном по быстродействию управлении обобщенной машиной Дубинса // Тр. ИММ УрО РАН. 2016. Т. 22. № 1. С. 26–35.

18. Зеликин М.И., Борисов В.Ф. Синтез оптимальных управлений с накоплением переключений // Итоги науки и техники. Сер. Современная математика и ее приложения. Тематический обзор. 2002. Т. 90. С. 5-189.

19. Бортаковский А.С. Оптимальные по быстродействию траектории плоского движения с неограниченной кривизной // Изв. РАН. ТиСУ. 2022. № 4. С. 38–48.

20. Пацко В.С., Федотов А.А. Трехмерное множество достижимости для машины Дубинса: сведение общего случая ограничений на повороты к

каноническому // Изв. РАН. ТиСУ. 2023. № 4. С. 25–49.

21. Понтрягин Л.С., Болтянский В.Г., Гамкрелидзе Р.В., Мищенко Е.Ф. Математическая теория оптимальных процессов. М.: Физматгиз, 1961.

22. Кларк Ф. Оптимизация и негладкий анализ. М.: Наука, 1988.

23. Бортаковский А.С. Необходимые условия оптимальности гибридных систем переменной размерности // Изв. РАН. ТиСУ. 2022. № 1. С. 28-40.

24. Аграчев А.А., Сачков Ю.Л. Геометрическая теория управления. М.: Физматлит, 2005.

25. Понтрягин Л.С., Болтянский В.Г., Гамкрелидзе Р.В., Мищенко Е.Ф. Математическая теория оптимальных процессов. М.: Физматгиз, 1961. 392 с.

26. Савелов А.А. Плоские кривые. Систематика, свойства, применения (справочное руководство). М.: Физматгиз, 1960. 293 с.

27. Баллман “Динамическое программирование”

28. "Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin //(2008, 1st edition, Prentice Hall)

29. "Code Complete: A Practical Handbook of Software Construction" by Steve McConnell //(1993, 1st edition, Microsoft Press)

30. "The Pragmatic Programmer: From Journeyman to Master" by Andrew Hunt and David Thomas //(1999, 1st edition, Addison-Wesley Professional)

31. "Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides //(1994, 1st edition, Addison-Wesley Professional)

32. Акулич И.Л. Математическое программирование в задачах и упражнениях: учебное пособие / И.Л. Акулич. СПб.: Лань, 2011. − 352 с.

33. Бабаков Н.А. Теория линейных систем автоматического управления / Н.А. Бабаков, А.А. Воронов. Москва: Высшая школа, 1986. − 367 с.

34. Беллман Р.Э., Дрейфус С. Прикладные задачи динамического программирования / Р.Э. Беллман, С. Дрейфус, [пер. с англ. Н.М. Митрофановой]. Москва: Наука, 1965. − 458 с.

35. Ермакова В.И. Сборник задач по высшей математике для экономистов / В.И. Ермакова. Москва: Инфра-М, 2003. − 575 с.

36. Каллихман И.Л., Войтенко М.А. Динамическое программирование в примерах и задачах / И.Л. Каллихман, М.А. Войтенко. Москва: Высшая школа, 1979. − 124 с.

37. Лежнев А.В. Динамическое программирование в экономических задачах / А.В. Лежнев. Москва: БИНОМ. Лаб. знаний, 2010. − 176 с.

ПРИЛОЖЕНИЕ А

Паспорт проекта

ДАТА УТВЕРЖДЕНИЯ

I ПАСПОРТ ПРОЕКТА

Наименование дополнительной профессиональной программы профессиональной переподготовки	Организация процесса разработки программного обеспечения
Наименование проекта	Программная реализация алгоритмов динамического программирования для решения практических задач
Шифр проекта	МАИ.2023.М8О-303Б.Динамическое_программирование
Заказчик проекта	Московский авиационный институт (национальный исследовательский университет)
Руководитель темы от МАИ	Пегачкова Елена Александровна
Рецензент темы	Булакина Мария Борисовна
Целевая аудитория результата проекта (кто потребитель результата проекта)	Логистические компании, центры хранения и сортировки товара
Длительность проекта	01.09.2023 - 31.12.2023
Название команды	#3.303
РОЛИ В ПРОЕКТЕ:	ФИО
TeamLead	Своеволин Иван Сергеевич
Разработчик	Сабурова Серафима Павловна
Тестировщик, архитектор	Сайфуллин Ильдар Камилович
Тестировщик, разработчик	Трофимов Владислав Олегович
Технический писатель, разработчик	Кудрявцев Андрей Георгиевич

Ссылки на ресурсы проекта

Ссылка на GitHub	https://github.com/Svoevolin/DP-DPP-MAI

II ОПИСАНИЕ ПРОЕКТА

Образ результата	Графический вывод решения задачи
Цель проекта	Поиск оптимального времени в рамках решения численных задач управления, не решаемых аналитически
Задачи проекта	
1 Сбор и изучение данных	Поиск информации о существующей проблеме, путях её решения. Также изучаются потребности заказчика.
2 Постановка и формализация задач	Математическая формализация задачи, пригодная для ее алгоритмического разрешения
3 Разработка программного кода, решающего задачу о рюкзаке	Написание программы, решающей поставленную задачу
4 Тестирование проекта	Комплексное тестирование всех созданных модулей, проверка реализованной функциональности, исправление ошибок
5 Написание документации, создание презентации	Создание отчёта о проделанной работе и описание проекта. Создание презентации для защиты
Результат проекта	Программный комплекс, предназначенный для решения поставленных задач
Ограничения и допущения, которые имеют или могут оказать существенное влияние на результат проекта	Предполагается что введенные данные имеют физический смысл

Необходимые ресурсы для выполнения проекта	Знание паттернов разработки на языке Python, комплекс математических методов, достаточный для формализации задачи и её аналитического и численного решения.
Риски проекта	Программная реализация окажется не оптимальной в сравнении с реализацией в другой среде разработки.

III КОМАНДА ПРОЕКТА

ФИО	**Роль**	**Компетенция**	**Задача проекта**
Своеволин Иван Сергеевич	TeamLead	DevOps, Docker, Git, PostgreSQL, организация рабочих процессов, коммуникация внутри команды	1) Тестирование проекта: - тестирование программы Python, - тестирование взаимодействия, - исправление найденных ошибок. 2) Создание презентации и отчёта: - выделение ключевых данных проекта, - подготовка текста, - подготовка фотоматериалов, - заполнение слайдов.
Сайфуллин Ильдар Камилович	Тестировщик, архитектор	Prolog, ReactJS developer, Latex, Python	1) Постановка и формализация задач - моделирование задач, - математическая формализация задач, - поиск алгоритмов решения.

			2) Сбор и изучение данных: - поиск информации об оптимизационных задачах, - выделение необходимой информации.
Трофимов Владислав Олегович	Тестировщик, разработчик	Тестирование, Python, Git, Agile, fault-tolerant & distributed systems, GoLang	1) Разработка программного кода, решающего задачи: - установка Python, - определение основных функций, - описание сценариев использования, - создание набросков кода, - написание функций, - соединение компонент.
Кудрявцев Андрей Георгиевич	Технический писатель, разработчик	GoLang, Prolog, Unreal Engine dev, Power Point, LaTex	1) Написание документации: - написание введения, - обзорной информации для отчета об используемых технологиях и концепциях программирования, - описание реализации библиотеки, - добавление в приложения к отчёту исходного

			кода и диаграмм.
Сабурова Серафима Павловна	Разработчик	1С, Python, Miro, Powerpoint	1) Сбор и изучение данных: - поиск информации об оптимизационных задачах, - выделение необходимой информации, - анализ полученных данных.

IV ЗАДАЧИ ПРОЕКТА

Задача	**Подзадача**	**Время на выполнение (в часах)**
1 Сбор и изучение данных	1.1 Поиск информации об оптимизационных задачах 1.2 Выделение необходимой информации 1.3 Анализ полученных данных	2 часа
2 Постановка и формализация задач	2.1 На основе анализа смоделировать задачу 2.2 Математически формализовать задачу 2.3 Поиск алгоритма решения	10 часов
3 Разработка программного кода, решающего задачи	3.1 Установка Python 3.2 Установка библиотек Python 3.3 Установка компилятора Python 3.4 Определение основных функций 3.5 Описание	20 часов

	сценариев использование 3.6 Описание компонентов программы 3.7 Описание взаимодействия 3.8 Создание набросков кода 3.9 Написание функций 3.10 Соединение компонент	
4 Тестирование проекта	6.1 Тестирование программы Python 6.2 Тестирование взаимодействия 6.3 Исправление найденных ошибок	25 часов
5 Написание документации, создание презентации	7.1 Написание введения 7.2 Обзорной информации для отчета об используемых технологиях и концепциях программирования 7.2 Описание реализации библиотеки 7.3 Добавление в приложения к отчёту исходного кода и диаграмм 7.4 Создание презентации	30 часов
ИТОГО ПЛАНИРУЕМОЕ ВРЕМЯ НА ПРОЕКТ:		87 часа

ПРИЛОЖЕНИЕ Б
QR-код на Github

С полным кодом программ задач можно ознакомиться по QR-коду, ведущему на репозиторий GitHub, на рисунке 18.

Рисунок 18 – QR-код

ПРИЛОЖЕНИЕ В

Код алгоритма задачи о замене оборудования

Этап условной оптимизации

```
def _conditional_optimization(r: list, s: list, p: int, n: int, period: int) -> list:
    print('Условная оптимизация:')
    # Создаем матрицу для максимальных прибылей и выборы замены/сохранения в кортеже: (прибыль, выбор стратегии)
    matrix = [[tuple()] * (n) for _ in range(period)]
    for k in range(period - 1, -1, -1): # итерируемся по k с последнего года эксплуатации к первому
        for t in range(n): # вычисляем прибыль для каждого возможного возраста оборудования
            if k == period - 1: # последний год эксплуатации
                save = r[t] # высчитываем прибыль при сохранении
                replace = r[0] + s[t] - p # высчитываем прибыль при замене
                print(f'F_{k + 1}({t}) = max({r[t]}, {r[0]} + {s[t]} - {p})',
                      f'= {save} (C)' if save >= replace else f'= {replace} (3)')
            else: # все года эксплуатации кроме последнего
                save = r[t] + (matrix[k + 1][t + 1][0]) if t < n - 1 else 0 # высчитываем прибыль при сохранении
                replace = r[0] + s[t] - p + matrix[k + 1][0][0] # рассчитываем прибыль при замене
                print(f'F_{k + 1}({t}) = max({r[t]} + {matrix[k + 1][t + 1][0] if t < n - 1 else 0},'
```

```
                f' {s[t]} - {p} + {r[0]} + {matrix[k + 1][0][0]})',
                f'= {save} (С)' if save >= replace else f'= {replace} (З)')
        matrix[k][t] = (save, 'С') if save >= replace else (replace, 'З') # сохраняем большую прибыль и стратегию
    return matrix
```

Этап безусловной оптимизации

```
def _unconditional_optimization(matrix: list, t_0: int, period: int) -> list:
    print("\nБезусловная оптимизация")
    optimal_strategy = [] # будем сохранять здесь выбор для каждого года
    t_current = t_0  # возраст оборудования на текущий год эксплуатации
    for k in range(period): # годы эксплуатации растут до планового периода
        print(f'{k + 1}-й год эксплуатации')
        print(f'    Возраст оборудования: {t_current}')
        if matrix[k][t_current][1] == 'З':
            optimal_strategy.append(matrix[k][t_current])
            print('    Выгоднее заменить это оборудование')
            t_current = 0 # оборудование заменится на новое с возрастом 0 лет
        else:
            optimal_strategy.append(matrix[k][t_current])
            print('    Выгоднее оставить это оборудование')
            t_current += 1 # оборудование постареет на 1 год
    return optimal_strategy
```

Основная функция

```
def optimal_replacement(r: list, u: list, s: list, p: int, t_0: int, period: int) -> (list, list):
    n = len(r) # вычислим о скольких годах у нас есть данные, n = max{t}
    r = list(x - y for x, y in zip(r, u)) # вычтем из доходов от оборудования издержки на него для каждого года
    # Этап условная оптимизация
    profit_matrix = _conditional_optimization(r=r, s=s, p=p, n=n, period=period)
    # Этап безусловной оптимизации
    optimal_strategy = _unconditional_optimization(matrix=profit_matrix, t_0=t_0, period=period)
    return profit_matrix, optimal_strategy
```

ПРИЛОЖЕНИЕ Д

Код алгоритма задачи о распределении инвестиций

Инициализация окна:

```
public MainWindow()
    {
        InitializeComponent();
        this.DataContext = this;
    }
```

Загрузка вводных данных:

```
private void btnLoadTable_Click(object sender, RoutedEventArgs e)
    {
        Microsoft.Win32.OpenFileDialog ofd = new
Microsoft.Win32.OpenFileDialog();
        ofd.InitialDirectory = AppDomain.CurrentDomain.BaseDirectory;
        if(ofd.ShowDialog() == true)
        {
            string[] _temp = System.IO.File.ReadAllLines(ofd.FileName);
            InputTable = new int[_temp.Length][]; //InputTable = new
int[_temp.Length][];
            for (int i = 0; i < _temp.Length; i++) //for (int i = 0; i < _temp.Length;
i++)
                InputTable[i] = _temp[i].Split(' ').Select(Int32.Parse).ToArray<int>();
            UpdateMatrix(InputTable);
```

```
        }
        txtN.Text = InputTable.Length.ToString();
        txtA.Text = (InputTable.Length - 1).ToString();
        txtgx.Text = (InputTable.Length - 1).ToString();
        btnStart.Visibility = Visibility.Visible;
        //-------------
    }
```

Вычисление ответа:

```
private void btnStart_Click(object sender, RoutedEventArgs e)
    {
        ResultTable = new int[InputTable.Length][];
        X = new int[InputTable.Length][];
        for (int i = 0; i < InputTable.Length; i++)
        {
            ResultTable[i] = new int[InputTable[i].Length];
            X[i] = new int[InputTable[i].Length];
        }
        ResultTable[0][InputTable.Length] = InputTable[0][InputTable.Length];
        for (int Xi = 1; Xi < InputTable.Length; Xi++)
        {
            int _i = 0;
            SetLOG(String.Format("f{0}({1}) + f({2}-{1}) = {3}", 4, Xi, Xi, Xi));
```

```
            ResultTable[Xi][InputTable.Length] =
max(InputTable[Xi][InputTable.Length], InputTable[Xi-1][InputTable.Length],out
_i);
            if (_i == 0)
            {
                X[Xi][InputTable.Length] = Xi;
            }
            else
            {
                X[Xi][InputTable.Length] = Xi-1;
            }
        }
        for (int fi = InputTable.Length-1; fi > 0 ; fi--)
        {
            for (int Xi = 0; Xi < InputTable.Length; Xi++)
            {
                ResultTable[Xi][fi] = fmax(fi, Xi);
            }
        }
        lblResult.Content = "f*= " + ResultTable[ResultTable.Length-1][1];
        UpdateMatrix(ResultTable, ref DataViewerResult);
    }
```