

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное автономное образовательное учреждение
высшего образования

«Национальный исследовательский

Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий, математики и механики

Направление подготовки:

02.03.02 Фундаментальная информатика и информационные технологии

ОТЧЕТ

по лабораторной работе

на тему:

«Оптимизация производства сахара: анализ и сравнение алгоритмов»

Выполнили: студенты группы

3822Б1ФИ1 Д. И. Суворов, А. В.

Баранов, А. П. Коробейников

Проверил: к.ф.-м.н. А.И. Эгамов

Нижний Новгород

2024

Оглавление

| | |
|---------------------------------------|-----------|
| ВВЕДЕНИЕ | 3 |
| 1. ПОСТАНОВКА ЗАДАЧИ | 4 |
| 2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ | 5 |
| 3. РЕАЛИЗОВАННЫЕ МЕТОДЫ | 7 |
| 3.1. Венгерский алгоритм | 7 |
| 3.2. Жадный алгоритм | 8 |
| 3.3. Бережливый алгоритм | 8 |
| 3.4. Гибридные алгоритмы | 9 |
| 3.5. Расчет выхода сахара..... | 10 |
| 4. ИНТЕРФЕЙС ПРОГРАММЫ | 13 |
| 5. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ | 14 |
| 5.1. Эксперимент №1 | 14 |
| 5.2. Эксперимент №2 | 16 |
| 5.3. Эксперимент №3 | 18 |
| 6. ЗАКЛЮЧЕНИЕ | 20 |
| 7. СПИСОК ЛИТЕРАТУРЫ | 21 |

ВВЕДЕНИЕ

В современной промышленности ключевым фактором успеха является грамотное планирование производственных процессов и использование имеющихся ресурсов. Особенно это актуально для предприятий пищевой промышленности, где оптимизация производственного цикла напрямую влияет на обеспечение населения страны продуктами питания и качество этих продуктов.

В данной работе будет проводиться исследование эвристических стратегий по решению задачи оптимизации производственного процесса на примере сахарного производства. В качестве данных использовать производственные показатели Сергачского сахарного завода.

В рамках работы рассматривается комплексный подход к оптимизации графика переработки сахарной свёклы с применением различных эвристических стратегий. Среди них:

- Венгерский алгоритм
- Жадный алгоритм
- Бережливый алгоритм
- Бережливо-жадный
- Жадно-бережливый алгоритмы

Особое внимание уделяется учёту таких производных факторов как, деградация сырья, дозаривания и содержание неорганических веществ, довольно сильно влияющих на эффективность переработки сырья.

1. ПОСТАНОВКА ЗАДАЧИ

Цель:

Цель лабораторной работы — разработать пользовательскую программу, с удобным интерфейсом, которая по входным данным сможет посчитать итоги каждой стратегии, построить гистограмму для наглядности и выбрать наилучшую стратегию для данного случая.

Задачи:

- ❖ Ознакомиться с необходимыми нам эвристическими стратегиями: Венгерский алгоритм, Жадный алгоритм, Бережливый алгоритм, Жадно-бережливый и Бережливо-жадный алгоритмы.
- ❖ Изучить необходимые библиотеки языка Python: NumPy для матричных вычислений, Munkres для использования венгерского алгоритма, Tkinter для создания удобного интерфейса и Matplotlib для красивой гистограммы
- ❖ Написать программу, способную взаимодействовать с пользователем и получать от него входные данные, реализующую все эвристические стратегии для полученных данных, сравнение результатов работы этих стратегий и их красивое графическое представление.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Задача о назначениях представляет собой фундаментальную проблему комбинаторной оптимизации, где требуется найти оптимальное распределение n работ между n исполнителями при заданной матрице эффективности или стоимости выполнения каждой работы каждым исполнителем.

Задача о назначениях - задача о наилучшем распределении некоторого числа работ между таким же числом исполнителей. Наиболее эффективным методом её решения является венгерский алгоритм.

Венгерский алгоритм - это эффективный метод решения задачи о назначениях, который может быть применён в различных областях, таких как логистика, транспортная логистика, экономика и т.д. Он позволяет найти оптимальное сочетание между ресурсами и задачами, минимизируя затраты или максимизируя выгоду.

Алгоритм был разработан и впервые опубликован в 1955 году венгерским математиком Джорджем Кёнигом.

Венгерский алгоритм основан на построении так называемой матрицы стоимостей, где каждый элемент матрицы представляет собой стоимость соответствующего сочетания элементов из двух наборов, и последовательном выборе оптимальных соответствий.

Алгоритм имеет сложность $O(n^3)$, что делает его эффективным для больших наборов данных. Описание работы алгоритма будет представлено в практической части при разборе конкретных примеров.

Жадный алгоритм – это метод решения оптимизационных задач, при котором на каждом шаге выбирается локально оптимальное решение

в надежде, что такие выборы приведут к глобально оптимальному решению.

Основная идея жадных алгоритмов заключается в том, что они принимают локально оптимальное решение на каждом шаге, без учёта последствий этого решения на будущие шаги. Жадные алгоритмы обычно применяются в задачах оптимизации, когда требуется найти максимальное или минимальное значение некоторой функции.

Следует отметить, что жадные алгоритмы не всегда дают оптимальное решение, в некоторых случаях они лишь дают приближённое оптимальное решение.

Бережливый алгоритм является фактически жадным для нахождения минимума.

Бережливо-жадный и жадно-бережливый алгоритмы являются, соответственно, комбинациями двух алгоритмов. От пользователя задается период v . Бережливо-жадный алгоритм – с первого по $v - 1$ период используется бережливый алгоритм. С v периода до последнего используется жадный алгоритм. Жадно-бережливый алгоритм – с первого по v период используется жадный алгоритм. С $v + 1$ периода до последнего используется бережливый алгоритм.

На выход сахара из единицы сырья влияет химический состав сырья, не только сахароза, но и так как называемая «неорганика»: калий, натрий и альфа-аминный азот.

3. РЕАЛИЗОВАННЫЕ МЕТОДЫ

3.1. Венгерский алгоритм

Реализация венгерского алгоритма была взята из стандартного программного обеспечения (библиотеки «Munkres»).

Поиск минимума:

```
def Minimum(S):  
    """Венгерский минимум"""  
    S_copy_for_min = S.T.copy()  
    hungarian = Munkres()  
    indexes = hungarian.compute(S_copy_for_min.tolist())  
    min_total_cost = 0  
    for row, column in indexes:  
        value = S_copy_for_min.tolist()[row][column]  
        min_total_cost += value  
    return min_total_cost
```

Поиск максимума:

```
def Maximum(S):  
    """Венгерский максимум"""  
    S_copy_for_max = S.T.copy()  
    num_batches, num_stages = S_copy_for_max.shape  
    for i in range(num_stages):  
        max_in_str = np.argmax(S_copy_for_max[i, :])  
        tmp_max = S_copy_for_max[i][max_in_str]  
        for j in range(num_batches):  
            S_copy_for_max[i][j] = S_copy_for_max[i][j] - tmp_max  
    S_copy_for_max *= -1  
  
    hungarian = Munkres()  
    indexes = hungarian.compute(S_copy_for_max.tolist())  
    max_total_cost = 0  
    for row, column in indexes:  
        value = S.T.tolist()[row][column]  
        max_total_cost += value  
    return max_total_cost
```

3.2. Жадный алгоритм

На каждом шаге выбирается максимальное доступное значение из текущего столбца матрицы:

```
def greedy_strategy(matrix):  
    """Жадная стратегия: на каждом этапе выбираем партию с наибольшей ценностью."""  
    calc_matrix = matrix.copy()  
    num_batches, num_stages = calc_matrix.shape  
    selected_order = []  
  
    for j in range(num_stages):  
        # Choosing the batch with the maximum value at the current stage  
        max_index = np.argmax(calc_matrix[:, j])  
        selected_order.append(max_index)  
        # Removing the selected batch from future stages  
        calc_matrix[max_index, :] = -np.inf  
  
    final_value = calculate_final_value(matrix, selected_order)  
    return final_value
```


3.3. Бережливый алгоритм

На каждом шаге выбирается минимальное доступное значение из текущего столбца матрицы:

```
def thrifty_strategy(matrix):
    """Бережливая стратегия: на каждом этапе выбираем партию с наименьшей ценностью."""
    calc_matrix = matrix.copy()
    num_batches, num_stages = calc_matrix.shape
    selected_order = []

    for j in range(num_stages):
        # Choosing the batch with the minimum value at the current stage
        min_index = np.argmin(calc_matrix[:, j])
        selected_order.append(min_index)
        # Removing the selected batch from future stages
        calc_matrix[min_index, :] = np.inf

    final_value = calculate_final_value(matrix, selected_order)
    return final_value
```

3.4. Гибридные алгоритмы

Гибридные алгоритмы представляют собой комбинацию жадного и бережливого подходов. В данной работе реализованы два варианта:

- Бережливо-жадный алгоритм: с первого по $V-1$ период используется бережливый алгоритм, а с периода V до последнего – жадный алгоритм
- Жадно-бережливый алгоритм: с первого по V период используется жадный алгоритм, а с периода $V + 1$ до последнего – бережливый алгоритм

Ниже представлена реализация бережливо-жадного алгоритма, где точка переключения V вычисляется как одна четвёртая от общего количества периодов:

```
def tg_strategy(matrix, v):
    """Стратегия Thrifty-Greedy: первые v-1 этапов – бережливая, начиная с v – жадная."""
    calc_matrix = matrix.copy()
    num_batches, num_stages = calc_matrix.shape
    selected_order = []

    for j in range(v):
        min_index = np.argmin(calc_matrix[:, j])
        selected_order.append(min_index)
        calc_matrix[min_index, :] = np.inf

    for j in range(num_batches):
        if calc_matrix[j, 0] == np.inf:
            calc_matrix[j, :] = -np.inf

    for j in range(v, num_stages):
        max_index = np.argmax(calc_matrix[:, j])
        selected_order.append(max_index)
        calc_matrix[max_index, :] = -np.inf

    final_value = calculate_final_value(matrix, selected_order)
    return final_value
```

Ниже представлена реализация жадно-бережливого алгоритма, где точка переключения V вычисляется как одна четвёртая от общего количества периодов:

```
def gt_strategy(matrix, v):
    """Стратегия Greedy-Thrifty: первые v-1 этапов - жадная, н
а ч и н а я с v - б е р е ж л и в а я."""
    calc_matrix = matrix.copy()
    num_batches, num_stages = calc_matrix.shape
    selected_order = []

    for j in range(v):
        max_index = np.argmax(calc_matrix[:, j])
        selected_order.append(max_index)
        calc_matrix[max_index, :] = -np.inf

    for j in range(num_batches):
        if calc_matrix[j, 0] == -np.inf:
            calc_matrix[j, :] = np.inf

    for j in range(v, num_stages):
        min_index = np.argmin(calc_matrix[:, j])
        selected_order.append(min_index)
        calc_matrix[min_index, :] = np.inf

    final_value = calculate_final_value(matrix, selected_order)
    return final_value
```

3.5. Расчёт потерь сахара с учётом влияния неорганических веществ

```
def calculate_L_matrix(n, K_range, Na_range, N_range,
                      I_range, num_of_days_for_one_period, organics, ripening=True, v=None):
    """ Функция для вычисления матрицы потерь сахара с учетом влияния
    неорганических веществ.
    n - количество партий
    K_range, Na_range, N_range, I_range - диапазоны для K, Na, N, I
    ripening - флаг дозаривания
    v - количество этапов дозаривания"""

    L = np.zeros((n, n))
    if (not organics):
        return L
    for i in range(n):
        # Случайная генерация значений для каждой партии
        K = np.random.uniform(K_range[0], K_range[1])
        Na = np.random.uniform(Na_range[0], Na_range[1])
        N = np.random.uniform(N_range[0], N_range[1])
        I0 = np.random.uniform(I_range[0], I_range[1])

        # Расчет содержания редуцирующих веществ для каждого этапа
        for j in range(n):
            I = I0 * (1.029)**(num_of_days_for_one_period * (j + 1 -
                num_of_days_for_one_period))
            Cx_M = 0.1541 * (K + Na) + 0.2159 * N + 0.9989 * I + 0.1967

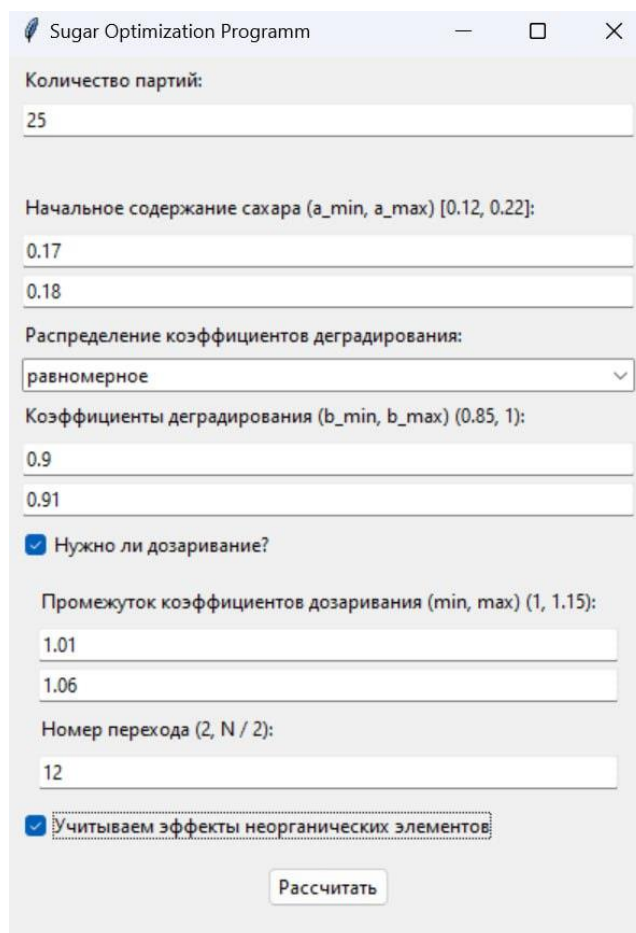
            # Если дозаривание, то учитывать его влияние
            if ripening and j < v:
                L[i, j] = Cx_M # Потери сахара при дозаривании
            else:
                L[i, j] = Cx_M # Потери сахара без дозаривания
        return L

    # Параметры диапазонов для каждого компонента
    K_range = [4.8, 7.05] # Диапазон для калия
    Na_range = [0.21, 0.82] # Диапазон для натрия
    N_range = [1.58, 2.8] # Диапазон для аминного азота
    I_range = [0.62, 0.64] # Диапазон для редуцирующих веществ
```

4. ИНТЕРФЕЙС ПРОГРАММЫ

Для удобства работы с алгоритмами оптимизации был разработан графический пользовательский интерфейс с использованием библиотек tkinter и matplotlib. Интерфейс позволяет:

- Задавать количество партий для обработки
- Настраивать параметры содержания сахара
- Включать и выключать влияние дозаривания и неорганических веществ
- Визуализировать результаты в виде гистограмм.



The screenshot shows a window titled "Sugar Optimization Programm" with standard window controls (minimize, maximize, close). The interface contains several input fields and checkboxes for configuring the optimization process:

- Количество партий:** A text input field containing the value "25".
- Начальное содержание сахара (a_min, a_max) [0.12, 0.22]:** Two stacked text input fields containing "0.17" and "0.18".
- Распределение коэффициентов деградирования:** A dropdown menu with "равномерное" selected.
- Коэффициенты деградирования (b_min, b_max) (0.85, 1):** Two stacked text input fields containing "0.9" and "0.91".
- Нужно ли дозаривание?** A checked checkbox.
- Промежуток коэффициентов дозаривания (min, max) (1, 1.15):** Two stacked text input fields containing "1.01" and "1.06".
- Номер перехода (2, N / 2):** A text input field containing "12".
- Учитываем эффекты неорганических элементов:** A checked checkbox.
- Рассчитать:** A button at the bottom right of the form.

Рис 1. Окно ввода данных

5. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

5.1. Эксперимент №1

Параметры проведения вычислительного эксперимента 1: 20 партий, диапазон сахаристости от 0.15 до 0.17, равномерное распределение деградаций с диапазоном деградации 0.94 до 0.99. Не учитываем дозаривание и влияние неорганических веществ на потерю сахара.

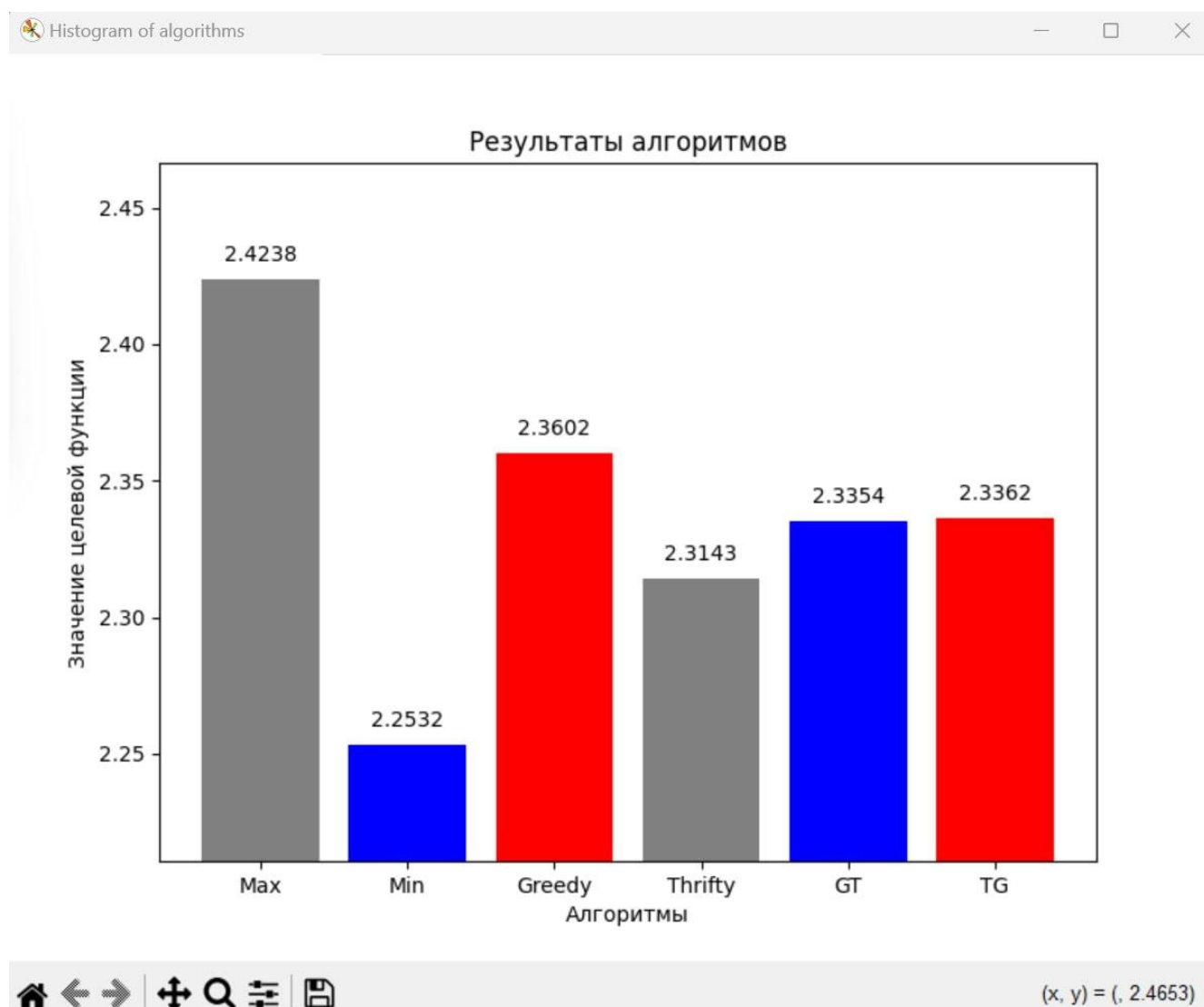


Рис. 2: Гистограмма результатов алгоритмов для эксперимента №1



Рис. 3: Результаты эксперимента №1

5.2. Эксперимент №2

Параметры проведения вычислительного эксперимента 2: 20 партий, диапазон сахаристости от 0.17 до 0.18, концентрированное распределение деградаций с диапазоном деградации 0.94 до 0.99. Не учитываем дозаривание и учитываем влияние неорганических веществ на потерю сахара.

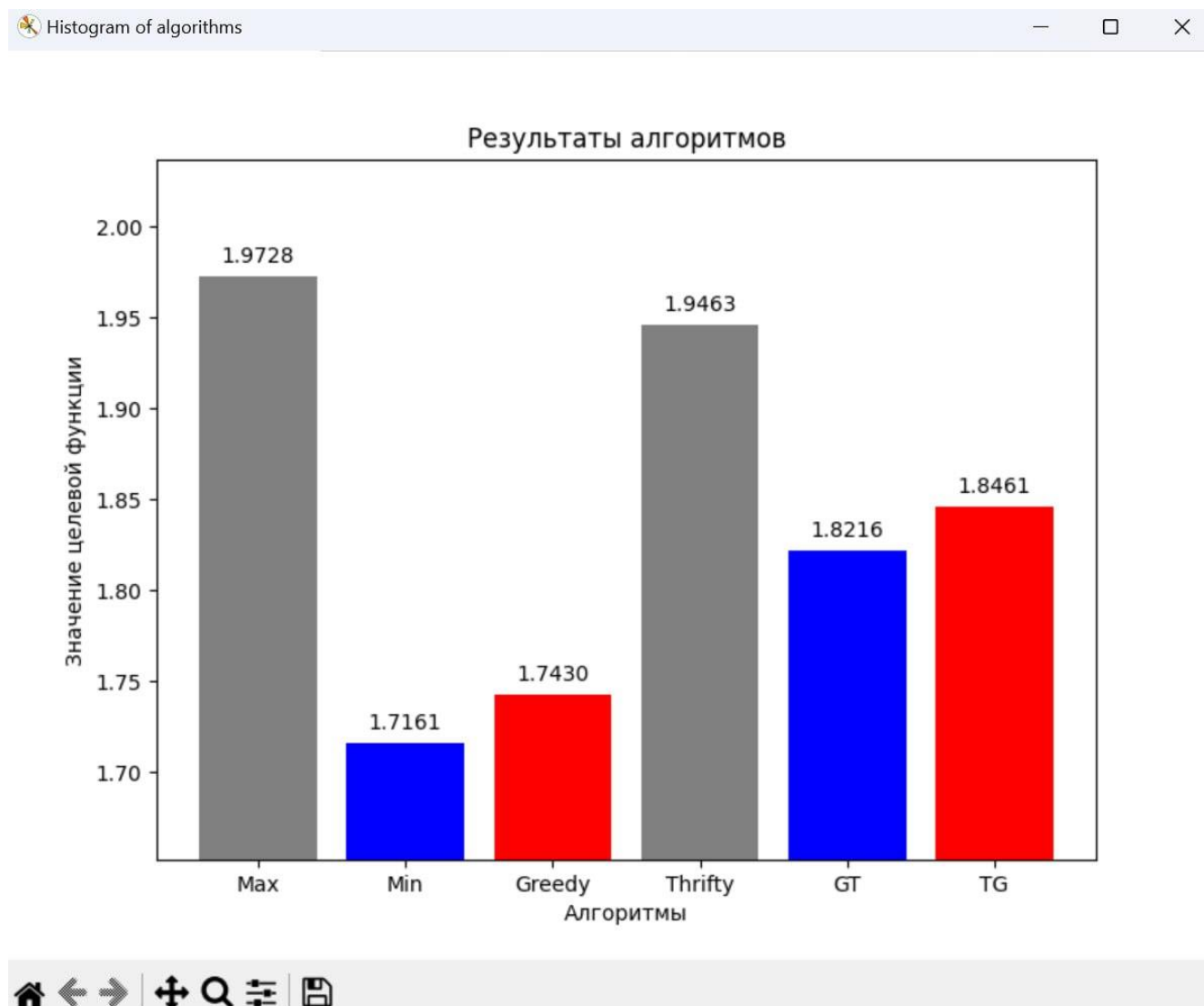


Рис. 4: Гистограмма результатов алгоритмов для эксперимента №2



Рис. 5: Результаты эксперимента №2

5.3. Эксперимент №3

Параметры проведения вычислительного эксперимента 3: 25 партий, диапазон сахаристости от 0.14 до 0.2, концентрированное распределение деградаций с диапазоном деградации 0.96 до 0.99. Учитываем дозаривание с номером перехода 12 и коэффициентами дозаривания $\min = 1.01$, $\max = 1.06$ и влияние неорганических веществ на потерю сахара.

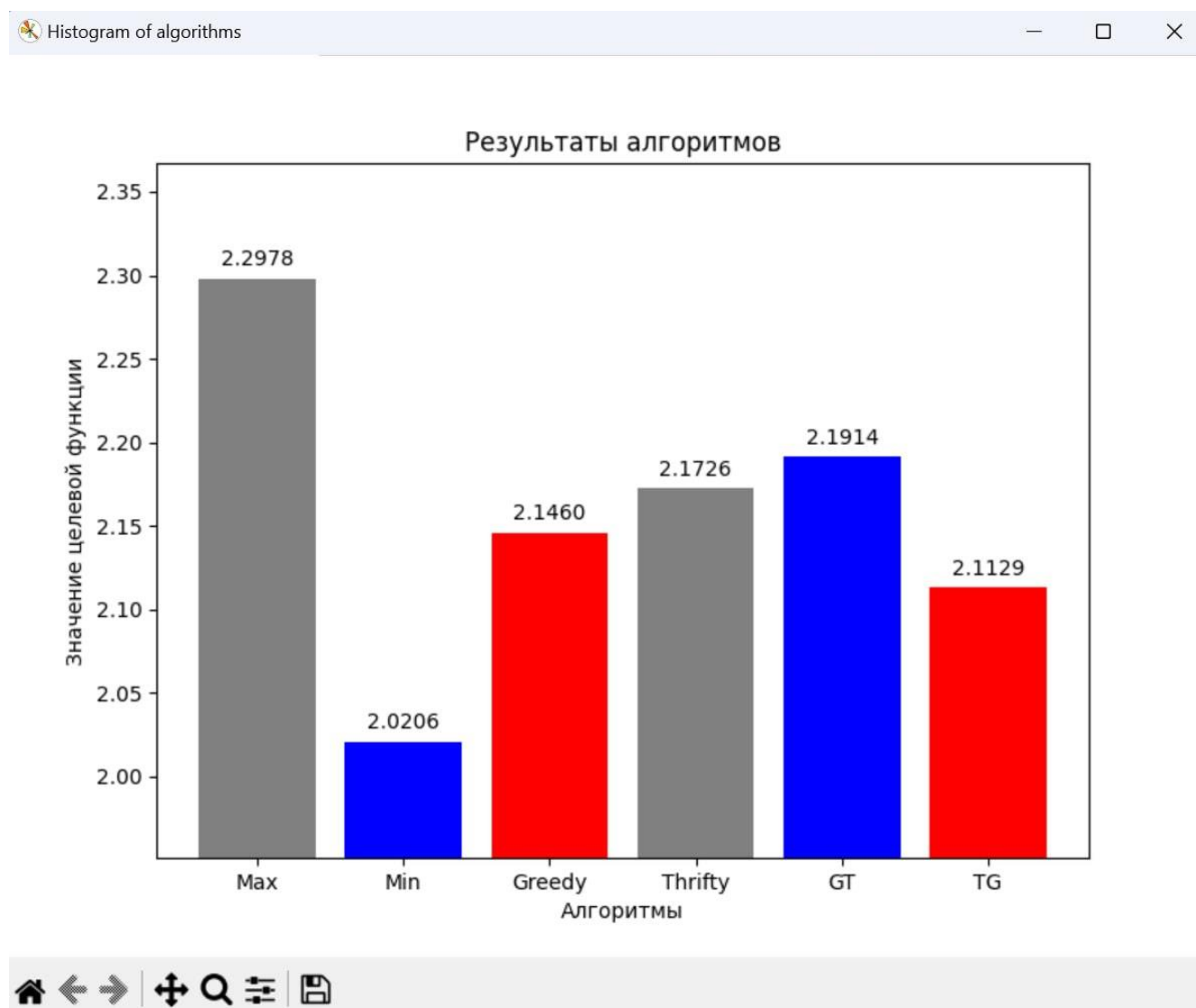


Рис. 6: Динамика накопления сахара в эксперименте №3



Рис. 7: Результаты эксперимента №3

6. ЗАКЛЮЧЕНИЕ

В рамках лабораторной работы были изучены и исследованы следующие алгоритмы: венгерский, жадный, бережливый, бережливо-жадный и жадно-бережливый. После анализа теоретической части была выполнена практическая реализация изученных эвристических методов. Алгоритмы были написаны на языке Python с использованием библиотеки Munkres, а затем протестированы на различных входных данных. В ходе экспериментов удалось получить ожидаемые результаты.

В ходе работы были достигнуты все поставленные цели: разработано программное обеспечение с удобным интерфейсом для оптимизации процесса производства сахара с учётом таких факторов как, влияние неорганических веществ и дозаривание.

7. СПИСОК ЛИТЕРАТУРЫ

1. Решение прикладных задач дискретной оптимизации : учебнометодическое пособие / Д. В. Баландин, О. А. Кузенков, Д. С. Малышев [и др.]. – Нижний Новгород : Нижегородский госуниверситет, 2023. – 23 с. – URL: <https://reader.lanbook.com/book/344675>

2. Язык программирования Python и библиотека tkinter. Практическое руководство / Р. А. Сузи. – Москва : ДМК Пресс, 2022. – 176 с. – ISBN 978-5-97060-751-7

3. Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter / W. McKinney. – 3rd ed. – Sebastopol : O'Reilly Media, 2022. – 550 p. – ISBN 978-1-098-10403-1.

4. Баландин Д.В. Лабораторная работа «Решение прикладных задач дискретной оптимизации» / Д.В. Баландин и др. // Н.Новгород: Издательство ННГУ. – 2023. – 23с. [Электрон. ресурс] <https://e-lib.unn.ru/MegaPro/UserEntry?Action=FindDoc&ids=850257&idb=0> (дата обращения 31.05.24).

5. Пегат А. Нечеткое моделирование и управление. 2-е издание (электронное) / А. Пегат. Под редакцией Ю. В. Тюменцева // Москва: БИНОМ. Лаборатория знаний. – 2013. – 798 с. ISBN 978-5-9963-1495-9. [Электрон. ресурс] https://rusneb.ru/catalog/000199_000009_007552818 (дата обращения 31.05.24).

6. The Hungarian Method for the Assignment Problem / H. W. Kuhn // Naval Research Logistics Quarterly. – 1955. – Vol. 2. – P. 83-97. – DOI: 10.1002/nav.3800020109.

7. Рафгарден Т. Совершенный алгоритм. Жадный алгоритм и динамическое программирование [Текст] / Т. Рафгарден // СПб: Питер. – 2020. – 256 с. ISBN: 978-5-4461-1445-0.