МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**СОРНЯКОВЫЙ АЛГОРИТМ РЕШЕНИЯ КВАДРАТИЧНОЙ ЗАДАЧИ О НАЗНАЧЕНИЯХ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. С. Фисун

(подпись)

Направление подготовки 02.03.02 — «Фундаментальная информатика и\_\_\_\_\_

(код, наименование)

информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_курс\_\_\_\_\_\_\_\_3\_\_\_\_\_\_\_\_\_

Направленность (профиль) Математическое и программное обеспечение компьютерных технологий\_\_\_\_\_\_\_\_\_

Научный руководитель

канд. техн. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. Е. Полупанова

(подпись)

Нормоконтролер

канд. техн. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. Е. Полупанова

(подпись)

Краснодар

2021

**РЕФЕРАТ**

В курсовой работе 23 стр., 3 ч., 5 рис., 3 табл., 6 источников, 1 приложение.

СОРНЯКОВЫЙ АЛГОРИМ, РЕШЕНИЕ КВАДРАТИЧНОЙ ЗАДАЧИ О НАЗНАЧЕНИЯХ.

Объектом исследования в данной работе является сорняковый алгоритм решения квадратичной задачи о назначениях на произвольных параметрах.

Цель курсовой работы – реализовать сорняковый алгоритм, решающий квадратичную задачу о назначении, определение значения свободных параметров, которые позволят оптимизировать время и качество работы.

Методологическая основа исследования включает в себя эмпирический метод (многократный запуск программы на разных параметрах и получение результатов её работы), анализ полученных зависимостей, аналогия (механизм работы программы сравнивается с механизмом распространения живых организмов).

При подготовке курсовой работы изучены принципы работы алгоритма сорняковой оптимизации.

В практической части курсовой реализовано приложение на языке программирования Python, позволяющее решить квадратичную задачу о назначениях с помощью сорнякового алгоритма на определенных параметрах и выявить их зависимость.

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc90408903)

[1 Задача о назначениях 6](#_Toc90408904)

[1.1 Теоретические основы 6](#_Toc90408905)

[1.2 Математическая постановка задачи 6](#_Toc90408906)

[2.3 Венгерский алгоритм 7](#_Toc90408907)

[2 Сорняковый алгоритм 10](#_Toc90408908)

[2.1 Сведения 10](#_Toc90408909)

[2.2 Биологические основы 10](#_Toc90408910)

[2.3 Схема алгоритма 11](#_Toc90408911)

[3 Реализация алгоритма IWO на задачу о назначениях 17](#_Toc90408912)

[3.1 Алгоритм 17](#_Toc90408913)

[3.2 Основные сведения о программе 18](#_Toc90408914)

[4 Анализ алгоритма 20](#_Toc90408915)

[4.1 Сравнение работы алгоритма на начальных параметрах IWO 20](#_Toc90408916)

[4.2 Сравнение времени работы алгоритма 21](#_Toc90408917)

[Заключение 23](#_Toc90408918)

[Список использованных источников 24](#_Toc90408919)

[Приложение А Основная программа 25](#_Toc90408920)

# ВВЕДЕНИЕ

В последние годы интенсивно развиваются алгоритмы поисковой оптимизации, которые называют поведенческими, интеллектуальными, вдохновленными природой, роевыми, многоагентными, популяционными и т.д. Эффективность таких алгоритмов соизмерима, а часто превосходит эффективность ставших уже классическими эволюционных алгоритмов, среди которых наиболее известен генетический алгоритм. С помощью популяционных алгоритмов успешно решаются сложные оптимизационные задачи.

Основная цель работы – разработка сорнякового алгоритма решения квадратичной задачи о назначениях, определение значения свободных параметров, которые позволят оптимизировать время и качество работы.

Для реализации поставленной цели предполагается решить следующие задачи:

* изучить алгоритм сорняковой оптимизации для решения поставленной задачи;
* разработать программу, предназначенную для решения квадратичной задачи о назначениях с помощью сорнякового алгоритма;
* провести ряд экспериментов для исследования зависимости времени и качества работы сорнякового алгоритма решения квадратичной задачи о назначениях на произвольных размерах.

Объектом исследования в данной работе является сорняковый алгоритм решения квадратичной задачи о назначениях на произвольных размерах.

Предметом исследования является время и качество работы сорнякового алгоритм решения квадратичной задачи о назначениях на произвольных размерах.

Информационной базой исследования являются результаты работы программы. Методологическая основа исследования включает в себя эмпирический метод (многократный запуск программы на разных параметрах и получение результатов её работы), анализ полученных зависимостей, аналогия (механизм работы программы сравнивается с механизмом распространения живых организмов).

Научная новизна работы заключается в том, что предлагается новый приближенный алгоритм решения квадратичной задачи о назначениях.

Теоретическая и практическая значимость работы состоит в широком применении сорнякового алгоритма для решения квадратичной задачи о назначении в различных областях. Например, есть множество n предприятий, которые могут быть расположены в n местах. Для каждой пары мест задано расстояние и для каждой пары производств задан вес или поток (т. e. количество материала (сырья или продукции), перевозимого между двумя производствами). Требуется расставить производства по местам (два производства нельзя размещать в одном месте) таким образом, что сумма расстояний, умноженных на соответствующие потоки, будет минимальной. Интуитивно понятно, что предприятия с большим потоком следует размещать ближе друг к другу.

# 1 Квадратичная задача о назначениях

## 1.1 Теоретические основы

Квадратичная задача о назначениях (КЗН, англ. Quadratic assignment problem, QAP) — одна из фундаментальных задач комбинаторной оптимизации в области оптимизации или исследования операций, принадлежащая категории задач размещения объектов.

Задача моделирует следующую задачу из реальной жизни: есть множество  предприятий, которые могут быть расположены в  местах. Для каждой пары мест задано расстояние и для каждой пары производств задан вес или поток (т. e. количество материала (сырья или продукции), перевозимого между двумя производствами). Требуется расставить производства по местам (два производства нельзя размещать в одном месте) таким образом, что сумма расстояний, умноженных на соответствующие потоки, будет минимальной.

Интуитивно понятно, что предприятия с большим потоком следует размещать ближе друг к другу.

Формулировка задачи похожа на формулировку задачи о назначениях, различаются они целевой функцией — в квадратичной задаче она квадратичная, что и отражает название.

## 1.2 Общая постановка

В соответствии с формулировкой, представленной Купмансом и Бекманом [1], имеется набор из N объектов и N локаций для их размещения, для этого набора определены:

* Матрица , состоящая из элементов ­– стоимостей расположения –ого объекта в ой локации;
* Матрица , состоящая из элементов , характеризующих количество ресурсов к транспортировке из –ого в ый объект;
* Матрица , состоящая из элементов , характеризующих стоимость транспортировки единицы ресурса из –ой в ую локацию.

Тогда полная стоимость транспортировки ресурса из -ого объекта в -й выглядит как произведение , где – расположение i-ого объекта в пространстве локаций, а 𝑘– расположение j-ого объекта в пространстве локаций.

Зададим некоторую перестановку в виде вектора − соответствует назначению в ‐ую локацию некоторого объекта.

Просуммируем все транспортировки между объектами, расположенными в локациях соответственно перестановке, и добавим к этому стоимость расположения объектов в этих локациях при перестановке . Получим полную стоимость для работы всех объектов при перестановке . Необходимо найти такое назначение всех объектов на локации, чтобы эта сумма была минимальной (формула 1.1).

|  |  |
| --- | --- |
|  | (1.1) |

На основе постановки задачи построим математическую модель [2]. Мы можем представить варьируемый параметр, являющийся перестановкой , как некоторую матрицу , состоящую из элементов , при этом она должна гарантировать, что каждый объект назначен только на одну локацию (формула 1.2) и на каждую локацию назначен только один объект (формула 1.3).

|  |  |
| --- | --- |
|  | (1.2) |
|  | (1.3) |
|  | (1.4) |
|  | (1.5) |

Тогда наименьшее значение критерия (формула 1.5) при выполнении ограничений (формула 1.2, 1.3, 1.4) будет достигаться с некоторой матрицей 𝑋, которая соответствует оптимальной перестановке . При данной постановке явно видно, что критерий зависит как от расположения истока, так и от расположения стока потока ресурсов.

## 2.3 Вычислительная сложность

В 1976 г. Sahniand и Gonzalez показали NP-полноту [2]. Определение сложности задачи показало несостоятельность поиска алгоритма для нахождения оптимального решения задачи за полиномиальное время. Легко убедиться в NP-полноте проблемы, поскольку к ней можно свести другие известные NP-полные проблемы [3].

1. Задача о коммивояжере: матрица F строится в соответствии с расстояниями между пунктами задачи коммивояжера; матрица D строится как матрица смежности задачи коммивояжера. Тогда оптимальная перестановка КЗН будет являться оптимальным циклом обхода для исходной задачи о коммивояжере.

2. Задача о максимальной клике: для поиска клики размера k необходимо чтоб матрица F формировалась как матрица смежности графа исходной задачи; матрица D строится как матрица смежности клики размера k. Тогда максимальная клика может быть найдена решением набора из КЗН, каждое из этих решений будет являться кликой размера , если она существует в исходном графе.

Существуют примеры входных данных, когда возможно получение решения за полиномиальное время. Если матрицы и являются взвешенными матрицами смежности, описывающие граф типа-дерево, то проблема может быть разрешена с помощью динамического программирования за полиномиальное время. Условие является необходимым, и, если хотя бы одна из матриц под него не подходит, то задача превращается вновь в NP-полную.

## 2.4 Обзор точных методов решения

Как известно, для NP-трудных задач найти полиномиальный алгоритм нахождения оптимального решения не представляется возможным. Поэтому для нахождения решения используют разнообразные методики динамического программирования и наиболее эффективной техники «ветвей и границ». Для решения КЗН наиболее используемыми являются три алгоритма [2].

1. Алгоритм одиночного назначения (на каждом листе дерева поиска «ветвей и границ» объект назначается на одну локацию).

2. Алгоритм парного назначения (на каждом листе дерева поиска «ветвей и границ» фиксированная пара объектов назначается на пару локаций).

3. Алгоритм взаимной позиции (уровень дерева поиска «ветвей и границ» не соответствует назначению объекта на локацию. Частичная перестановка на каждом уровне определена с точки зрения дистанции между объектами).

Все три алгоритма объединяет то, что они начинаются с пустой перестановки и в конце гарантированно получается некоторая перестановка, являющаяся решением задачи. На практике же алгоритм одиночного назначения оказался наилучшим. Алгоритм парного назначения показал себя неэффективным в вычислении, а алгоритм взаимной позиции подходит в основном для проблем с разреженными матрицами. В методе «ветвей и границ» главным критерием для применения к задачам комбинационной оптимизации является нижняя граница. Она будет отчетливо видна поэтому ее легко вычислить.

# 2 Сорняковый алгоритм

## 2.1 Сведения

Алгоритм сорняковый оптимизации (Invasive Weed Optimization, IWO) вдохновлен таким явлением, как колонизация сельскохозяйственных угодий сорняками. Алгоритм был предложен в 2006 году иранскими учеными Мехрабианом и Лукасом и основан на моделировании таких свойств, как посев, рост и конкуренция в колонии сорняка.

## 2.2 Биологические основы

Говоря простыми словами, сорняк – это любое растение, находящееся там, где оно мешает другим. В общем случае, все растения можно назвать сорняками. Они причиняют вред культивируемым растениям, что приводит к снижению урожайности и качеству сельскохозяйственной продукции.

Одной из интересных особенностей сорняков является то, что они всегда побеждают. И чем больше люди стараются, тем лучше они приспосабливаются. Сорняки вторгаются в земледелие с помощью рассеивания семян, колонизации и оккупации полей. Высокая эффективность указанных процессов обеспечивается благодаря их высокой адаптации к местным условиям, биоразнообразию.

Основным механизмом, определяющим динамику любых растений, является естественный отбор, из которого можно выделить два крайних типа: **r-отбор** и **К-отбор**. Реальные стратегии отбора лежат между этими предельными типами.

Можно сказать, что девизом r-отбора является «живи быстро, размножайся быстро, умирай молодым»*.* Данный тип отбора необходим для успеха в нестабильной, непредсказуемой окружающей среде. При r-отборе предпочтительны такие качества, как высокая плодовитость, маленький размер семян и приспособленность к рассеиванию их на большое расстояние.

К-отбор использует принцип «живи медленно, размножайся медленно, умирай в старости»*.* Этот тип отбора необходим для успеха в стабильной, предсказуемой окружающей среде, когда вероятно тяжелое соперничество за ограниченные ресурсы между конкурентоспособными индивидуумами. Ситуация имеет место, если размер популяции в ареале обитания близок к максимуму, который он способен вместить. При К-отборе предпочтительны такие качества индивидов, как большой размер семян, длинная жизнь, небольшое потомство, за которым требуется интенсивный уход [3].

## 2.3 Схема алгоритма

Блок-схема алгоритма представлена на рисунке 2.

В алгоритме IWO модель поведения сорняков при колонизации учитывает следующие базовые свойства процесса:

1) Распределение конечного числа семян по всей области поиска (инициализация популяции).

2) Производство выросшими растениями семян в зависимости от приспособленности растений (воспроизводство).

3) Распределение произведенных семян в случайном порядке по области поиска (пространственное распределение).

4) Повторение шагов 2, 3 до тех пор, пока не будет достигнут заданный максимум числа растений.

5) Отбор растений с более высокой приспособленностью, их воспроизводство и пространственное распределение (конкурентное исключение).

6) Повторение шага 5 до выполнения условия окончания процесса [3].

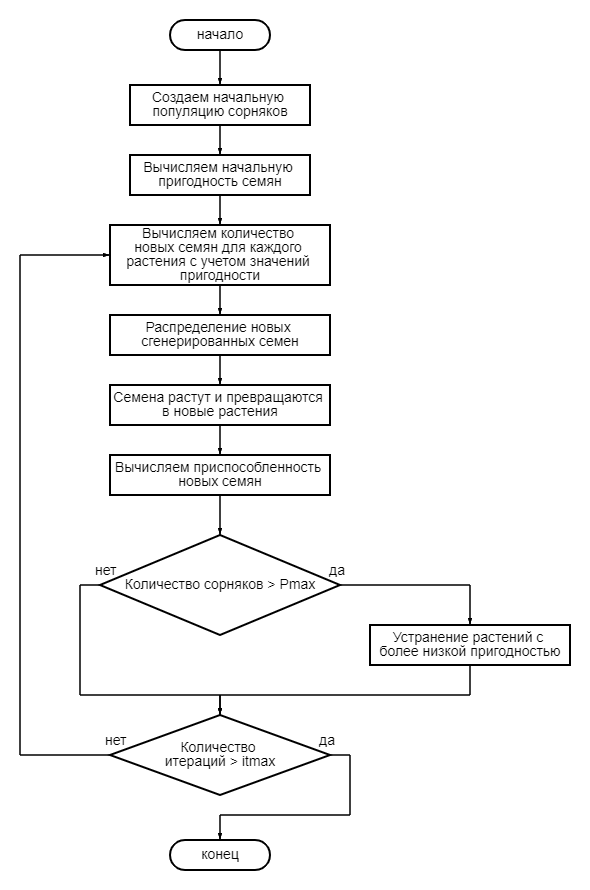


Рисунок 2 – Блок схема алгоритма IWO

Рассмотрим каждый шаг алгоритма.

Инициализация популяции: начальное положение сорняков принимаем случайным, равномерно распределенным в некотором гиперпараллепипеде ) (формула (2.1))

|  |  |
| --- | --- |
|  | (2.1) |

где

­­­– векторы свободных параметров алгоритма;

–начальный размер популяции.

Воспроизводство: в оригинальном алгоритме IWO число семян , произведенных сорняком , линейно зависит от его текущей приспособленности и определяется формулой (формула (2.2)) (рисунок 3)

|  |  |
| --- | --- |
|  | (2.2) |

где

­­­, – заданные константы, представляющие собой минимальное и максимальное число семян, которые могут быть произведены каждым из сорняков на текущей итерации (формула (2.3)).

|  |  |
| --- | --- |
|  | (2.3) |

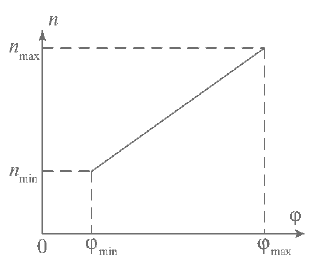


Рисунок 3 – График воспроизводства сорняков

Стоит отметить, что во многих популяционных алгоритмах агентам, имеющим низкую приспособленность, не позволяют размножаться, хотя существуют вероятность того, что некоторые из них несут более полезную информацию, чем их конкуренты. Схема воспроизводства, используемая алгоритмом IWO, дает шанс менее приспособленным агентам выжить и размножиться подобно тому, как это происходит в природе [4].

Пространственное распределение: произведенные сорняком семена распределяем в окрестности родительского растения в соответствии с нормальным законом распределения, имеющим нулевое математическое ожидание и стандартное отклонение (формула (2.4)).

|  |  |
| --- | --- |
|  | (2.4) |

Стандартное отклонение в этом выражении зависит от текущего номера поколения , уменьшаясь с ростом этого номера по формуле (2.5)

|  |  |
| --- | --- |
|  | (2.5) |

где

­­­ – начальное и конечное значение стандартного отклонения ­­­ ;

– свободный параметр модуляции, определяющий характер функции .

Данная формула обеспечивает уменьшение вероятности распространения семян вдали от родительского растения с ростом номера итераций, повышая локализационные свойства алгоритма. Можно сказать, что данная схема изменения стандартного отклонения реализует механизм перехода от r-отбора к K-отбору в процессе эволюции популяции.

Конкурентное исключение: обозначим через субпопуляцию сорняков, являющихся потомками растения [5]. Тогда до достижения всей популяцией сорняков своего максимального размера, равного , новую популяцию формируем путем объединения текущей популяции со всеми субпопуляциями (формула (2.6)).

|  |  |
| --- | --- |
|  | (2.6) |

Конкурентное исключение начинает функционировать после превышения популяцией размера и заключается в уничтожении сорняков с меньшей приспособленностью до достижения популяцией этого размера.

Таким образом, растения и их потомки оцениваются вместе, и тем из них, которые обладают лучшей приспособленностью, позволяют размножаться. Данный механизм позволяет растениям с меньшей приспособленностью воспроизводиться, и, если их потомки обладают хорошей приспособленностью, они могут выжить.

Свободными параметрами алгоритма являются начальная и максимальная численность популяции , минимальное и максимальное число семян ­­­,, которые могут быть произведены каждым из сорняков на данной итерации; начальное и конечное значение стандартного отклонения , параметр модуляции .

# 3 Реализация алгоритма IWO на квадратичную задачу о назначениях

## 3.1 Алгоритм

Алгоритм IWO можно разбить на следующие шаги:

1. на вход поступает две квадратные матрицы (матрица стоимости перевозки ресурса) и (матрица количества перевозимого ресурса), состоящая из вещественных чисел, а также их размерность;
2. определяются параметры IWO:

* максимальное количество итераций;
* начальный и конечный размер популяции;
* минимальное и максимальное количество семян;
* показатель уменьшения дисперсии;
* начальное и конечное значение стандартного отклонения.

1. генерируется начальная популяция:

* генерируется вектор с заданной размерность, где каждый элемент равномерно распределенное случайное число от до со средним значение равным нулю;
* каждому число ставится в соответствие его индекс;
* сортируется полученный вектор по возрастанию;
* вычисляется целевая функция.

1. вычисляется минимальное и максимальное значения целевой функции из популяции;
2. фаза воспроизводства потомства для каждого сорняка:

* вычисляется число семян, которые может произвести данный сорняк;
* каждое семя располагаем в окрестности родителя;
* вычисляется целевая функцию для данного семени;

1. объединяем полученное потомство с потомством, полученным на предыдущей итерации;
2. сортируем по значению целевой функции и отбираем сорняки до размера максимальной популяции;
3. выполняем шаги 4 – 8 до заданного числа итераций.

## 3.2 Основные сведения о программе

Для реализации приложения используется язык программирования Python и среда разработки Pycharm.

Программа представляет собой приложение с вводом и выводом данных через консоль. Пользователю предлагается ввести название входного файла с матрицами, начальный и максимальный размер популяции, минимальное и максимальное число семян, показатель уменьшения дисперсии, начальное и конечное значение стандартного отклонения и количество итераций.

В результате работы программы на выходе пользователь получает найденное решение, значение целевой функции.

Пример результата работы программы показан на рисунке 4-5.

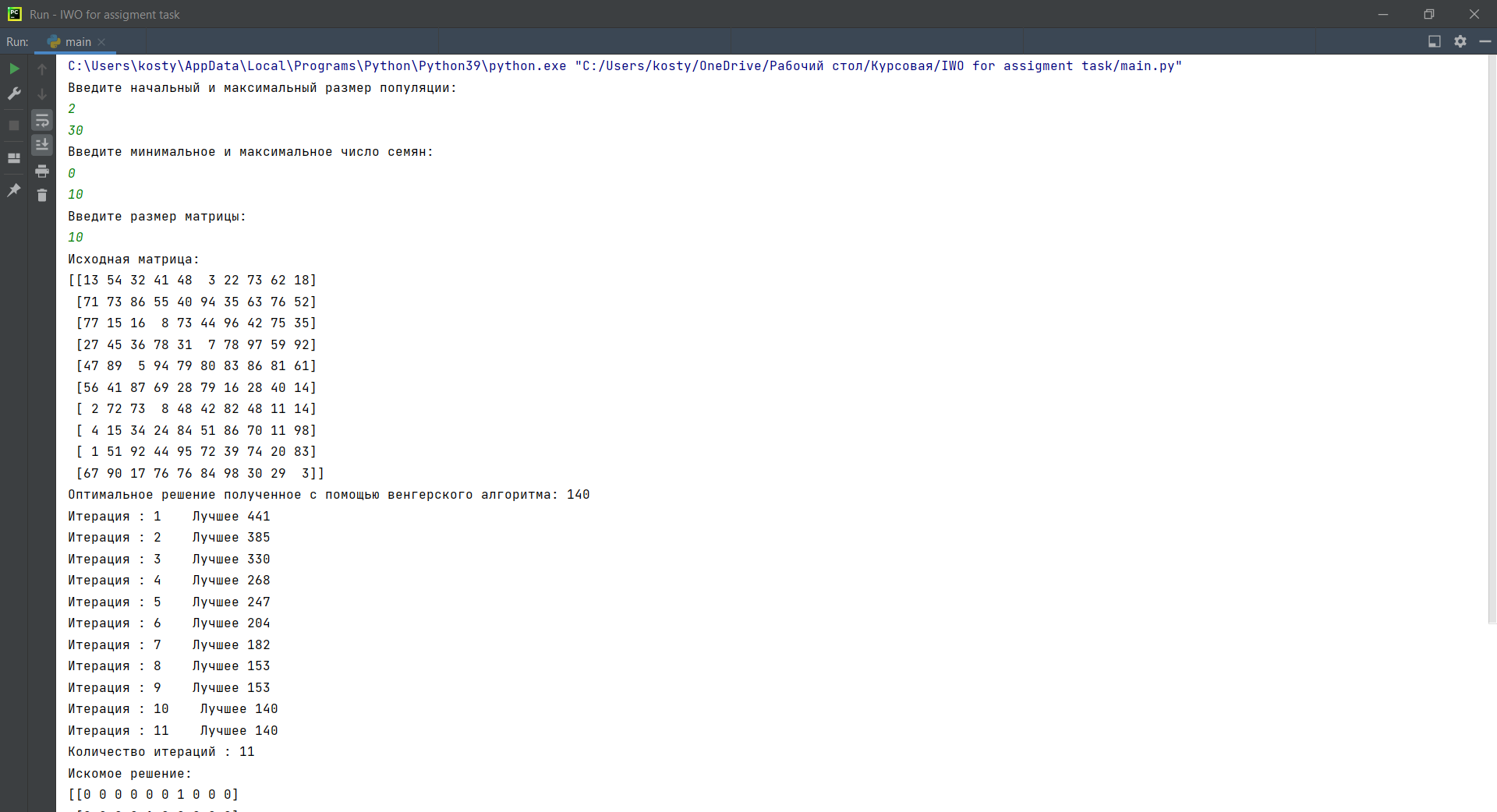


Рисунок 4 – Результат работы программы

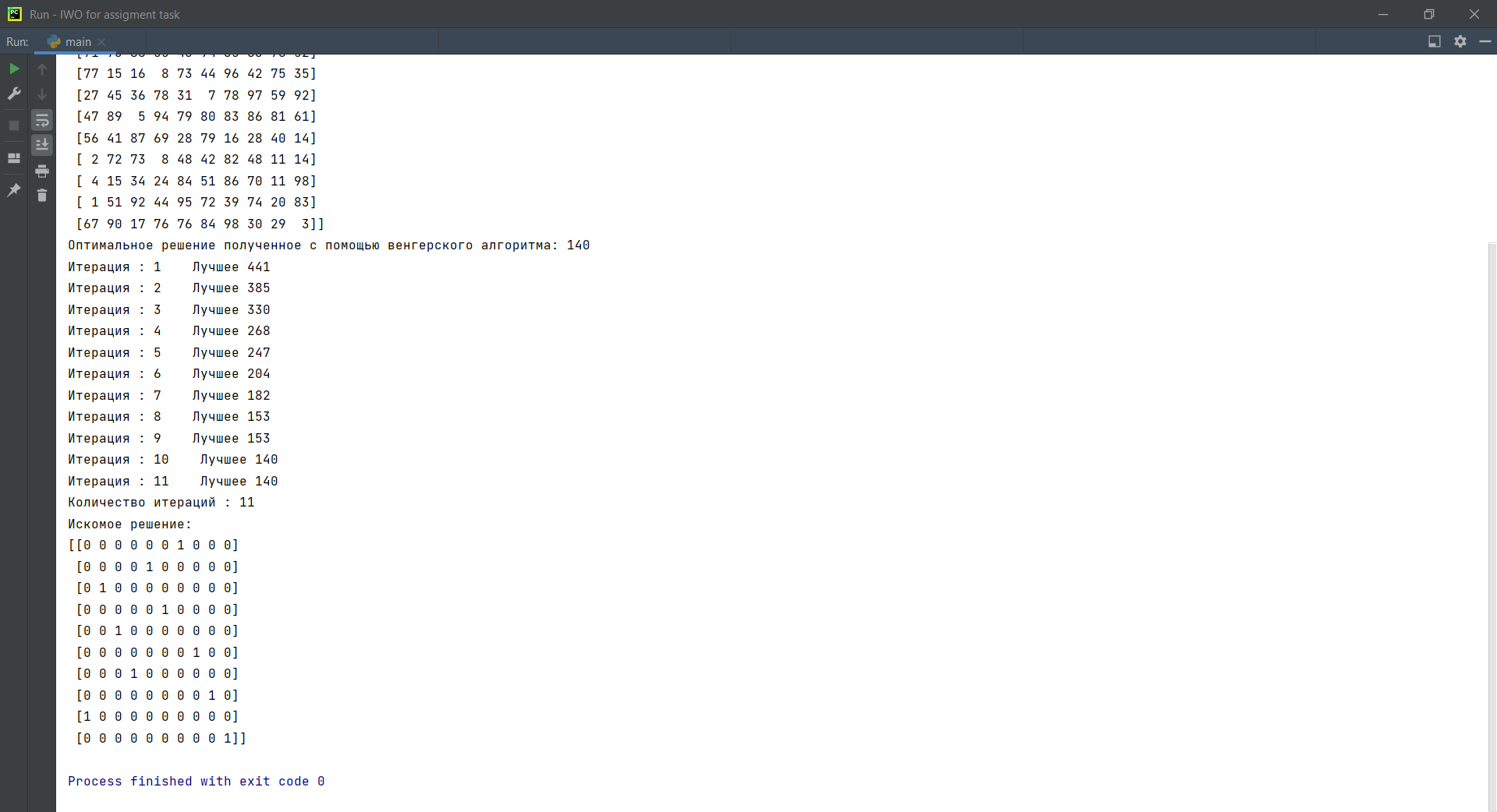


Рисунок 5 – Результат работы программы

# 4 Анализ алгоритма

## 4.1 Сравнение работы алгоритма на начальных параметрах IWO

Каждый раз на вход программе подается матрица, размерностью 100x100. Начальная популяция состоит из 2 матриц, заполненных единицами по главной и побочной диагонали.

В таблице 1-2 представлены результаты программы на различных параметрах IWO – количество итераций, потребовавшихся на нахождение оптимального решения. В данной задаче оптимальное решение находится с помощью венгерского алгоритма. По достижению заданной точности алгоритм прекращается.

Таблица 1 – Сравнение результатов программы при изменении максимального размера популяции

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Параметры | Res1 | Res2 | Res3 | Res4 | Res5 |
| minSeed | 0 | 0 | 0 | 0 | 0 |
| maxSeed | 5 | 5 | 5 | 5 | 5 |
| maxPopSize | 30 | 50 | 70 | 90 | 110 |
| m | 2 | 2 | 2 | 2 | 2 |
| Количество итераций | 85 | 40 | 30 | 28 | 23 |

Таблица 2 – Сравнение результатов программы при изменении максимального количества семян

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Параметры | Res1 | Res2 | Res3 | Res4 |
| minSeed | 0 | 0 | 0 | 0 |
| maxSeed | 25 | 50 | 75 | 100 |
| maxPopSize | 50 | 50 | 50 | 50 |
| m | 2 | 2 | 2 | 2 |
| Количество итераций | 56 | 33 | 30 | 24 |

По полученным данным можно сделать вывод, что эффективность сорнякового алгоритма существенно зависит от выбора свободных параметров.

## 4.2 Сравнение времени работы алгоритма

Для анализа эффективности работы разработанного сорнякового алгоритма использовалось два параметра – время, затраченное программой на решение задачи и размерность матрицы.

На рисунке 6 показана зависимость времени работы сорнякового алгоритма от размерности матрицы стоимости.



Рисунок 6 – Зависимость времени работы сорнякового алгоритма от размерности матрицы стоимости

Также в таблице 3 представлены сравнительные значения работы алгоритма.

Таблица 3 – Сравнительные значения работы алгоритма

|  |  |
| --- | --- |
| Размерность матрицы | Сорняковый алгоритм |
| 100 | 0,1563 c |
| 250 | 6,9300 с |
| 500 | 22,9000 с |
| 1000 | 93,5600 с |

По полученным данным можно сделать вывод, что сложность данного алгоритма , где – время жизни сорняков, т.е. число итераций, – максимальная популяции, – максимальное количество семян, которые может произвести данный сорняк, – размер исходной матрицы.

# ЗАКЛЮЧЕНИЕ

Цель курсовой работы – реализовать сорняковый алгоритм решения задачи о назначениях – достигнута. Определены значения свободных параметров при которых алгоритм работает эффективнее и качественнее.

В теоретической части курсовой работы изучены принципы алгоритма сорняковый оптимизации, рассмотрена задача о назначениях и венгерский алгоритм для её решениях.

В практической части работы разработано приложение, написано на языке программирования Python, которые решает задачу о назначениях с помощью венгерского и сорнякового алгоритма.

Разработанный сорняковый алгоритм решения задачи о назначении на произвольных размерах матрицы можно использовать для наилучшего распределения некоторого числа работ между таким же числом исполнителей.

В перспективе данный алгоритм можно модернизировать для решения квадратичной задачи о назначениях. Например, есть множество предприятий, которые могут быть расположены в мест. Для каждой пары мест задано расстояние и для каждой пары производств задан вес, т. е. материала, перевозимого между двумя производствами. Требуется расставить производство по местам, чтобы сумма расстояний была минимальной.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. **Koopmans, T.C.** Assignment problems and the location of economic activities / T.C. Koopmans, M.: Beckman // Econometric. – 1957. – № 25. – Р. 53-76. – Текст : электронный URL : researchgate.net/ publication/ 330464076\_Improved\_Invasive\_ weed\_optimization\_Algorithm\_IWO\_ Based\_ on\_Chaos \_Theory\_for\_Optimal\_design\_of\_PID\_controller tps:// (дата обращения 14.11.2021)
2. Panos, M.P. The Quadratic Assignment Problem: A Survey and Recent Developments / M.P. Panos, Franz Rendl. Henry Wolkwicz, 1994.
3. **Sahni, S.** P-complete Approximation Problems / S. Sahni, T. Gonzalez // Journal of the ACM 23, 1976. – Р. 555-565 Текст: непосредственный. (дата обращения 15.11.2021).
4. Clayton, W. Commander. Survey of the Quadratic Assignment Problem, with Applications / W.Clayton. – Darmstadt University of North Carolina at Chapel Hill, 2005.
5. **Карпенко, А.П.** Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой. Издательство: МГТУ Год: 2017 Страниц: 447 ISBN: 978-5-7038-4634-6 Текст: непосредственный (дата обращения 12.11.2021)
6. **Misaghi, M.** Improved invasive weed optimization algorithm (IWO) based on chaos theory for optimal design of PID controller Текст : электронный URL: https://[www.researchgate.net/](http://www.researchgate.net/)publication/ 330464076\_ Improved \_Invasive \_weed\_optimization\_Algorithm\_IWO (дата обращения 12.11.2021)
7. **Yazdani, M.** Invasive weed optimization algorithm for minimizing total weighted earliness and tardiness penalties on a single machine under aging effect. – Текст : электронный URL : https:// www.semanticscholar .org/paper /Assignment-Problems-and-the-Location-of-Economic-Koopmans-Beckmann (дата обращения 15.11.2021)
8. **Бэрри, П.** Изучаем программирование на Python / П. Бэрри; [пер. с англ. М.А. Райтман]. – Москва: Издательство «Э», 2017. -624с. – Текст: непосредственный (дата обращения 20.11.2021)

# ПРИЛОЖЕНИЕ А

**Основная программа**

*# Главная функция***from** math **import** log  
**from** random **import** randint  
  
**from** numpy **import** random  
**import** numpy **as** np  
  
**import** array **as** arr  
  
**from** scipy.optimize **import** linear\_sum\_assignment  
  
*# Функция***def** function(x):  
 **return** pow(x, 2)  
  
*# Целевая функция***def** target\_function(c,x):  
 result = 0 *# Результат* **for** i **in** range(len(c)):  
 **for** j **in** range(len(c)):  
 result += c[i][j] \* x[i][j]  
  
 **return** result  
  
*# Вывод матриц***def** print\_matrix(matrix):  
 **for** i **in** range(len(matrix)):  
 **for** j **in** range(len(matrix)):  
 print(**"{:4d}"**.format(matrix[i][j]), end=**""**)  
 print()  
  
**def** mixing\_matrix(matrix, row\_or\_column, pos1, pos2):  
 *# Меняем строки* **if** (isinstance(matrix, tuple)):  
 matrix = tuple\_in\_matrix(matrix)  
  
 size\_matrix = size\_matrix\_task  
  
 **if** row\_or\_column == 0:  
 **for** i **in** range(size\_matrix):  
 t = matrix[pos1][i]  
 matrix[pos1][i] = matrix[pos2][i]  
 matrix[pos2][i] = t  
  
 *# Меняем столбцы* **else**:  
 **for** i **in** range(size\_matrix):  
 t = matrix[i][pos1]  
 matrix[i][pos1] = matrix[i][pos2]  
 matrix[i][pos2] = t  
  
 **return** matrix  
  
*# Из матрицы делаем кортеж***def** matrix\_in\_tuple(matrix):  
 a = []  
 **for** i **in** range(len(matrix)):  
 a.append([])  
 **for** j **in** range(len(matrix)):  
 a[i].append(matrix[i][j])  
  
 lis = list()  
 **for** i **in** range(0, len(a)):  
 **for** j **in** range(0, len(a)):  
 lis.append(a[i][j])  
  
 **return** tuple(lis)  
  
*# Из картежа делаем матрицу***def** tuple\_in\_matrix(tuple):  
 matrix\_list = list(tuple) *# Получаем список* k = 0  
 matrix = []  
 **for** i **in** range(size\_matrix\_task):  
 a = []  
 **for** j **in** range(size\_matrix\_task):  
 a.append(matrix\_list[k])  
 k += 1  
 matrix.append(a)  
  
 **return** np.array(matrix)  
  
  
**def** iwo(eps):  
 *# Размер матрицы задачи  
  
 # IWO параметры* maxIt = 20 *# Максимальное количество итераций* population\_Size\_Initial = 2 *# Начальный размер популяции* maximum\_Population\_Size = 50 *# Максимальная размер популяции* min\_Seed = 0 *# Минимальное количество семян* max\_Seed = 100 *# Максимальное количество семян* m = 2 *# Показатель уменьшения дисперсии(m)* sigma\_initial = 0.5 *# Начальное значение стандартного отклонения* sigma\_final = 0.001 *# Конечное значение стандартного отклонения  
  
 # Инициализируем случайным образом минимальное количество сорняков для старта алгоритма  
 # Заполняем 0 гиперпараллелепипед* initial\_Population = np.zeros((population\_Size\_Initial,  
 size\_matrix\_task,  
 size\_matrix\_task),  
 dtype=int)  
  
 *# Случайно заполняем 1(В данном случае по диагонали)* **for** i **in** range(size\_matrix\_task):  
 initial\_Population[0][i][i] = 1  
 **for** i **in** range(size\_matrix\_task):  
 initial\_Population[1][i][size\_matrix\_task - i - 1] = 1  
  
 initial = dict() *# Словарь для сопоставления матриц и целевой функции* initial\_Fitness = list()  
 **for** i **in** range(population\_Size\_Initial):  
 initial\_Fitness.append(target\_function(matrix\_task,initial\_Population[i])) *# Добавляем целефую функцию в список* initial[matrix\_in\_tuple(initial\_Population[i])] = target\_function(matrix\_task,initial\_Population[i]) *# Заполняем словарь начальной популяцией  
  
 # print(initial\_Fitness1)* best\_Solution = min(initial\_Fitness)  
 *# Основной цикл* t = 0 *# Количество итераций* **while**(best\_Solution != eps):  
  
 *# обновить стандартное отклонение по формуле* sigma = (pow(((maxIt - t) / maxIt), m) \* (sigma\_initial - sigma\_final)) + sigma\_final  
 *#print(sigma)* best\_Solution = min(initial\_Fitness)  
 worst\_Solution = max(initial\_Fitness)  
  
 new\_Initial\_Population = list()  
 new\_Fitness\_Population = list()  
  
 new\_Initial = dict()  
  
 *# фаза воспроизводства* **for** i **in** range(0, len(initial)):  
 *# Вычисляем число семян, которые может произвести данный сорняк* ratio = (initial\_Fitness[i] - worst\_Solution) / (best\_Solution - worst\_Solution)  
 s = (min\_Seed + ((max\_Seed - min\_Seed) \* ratio))  
 **if** s == 0:  
 s += 1  
 *# print(s)* **for** j **in** range(0, round(s)):  
  
 gene = sigma \* population\_Size\_Initial  
 *#new\_Solution\_Position = initial\_Population[i]  
 #for i in range(gene):  
 # Выбираем номера строк(столбцов) которые собираемся менять* t1 = randint(0,size\_matrix\_task - 1)  
 t2 = randint(0,size\_matrix\_task - 1)  
  
 *# Чтобы значения не совпадали* **while**(t1 == t2):  
 t2 = randint(0, size\_matrix\_task - 1)  
  
 *# Меняем строки или столбцы* row\_or\_column = randint(0,1) *# Строка - 0, столбец - 1  
  
 # Генерация потомка  
 #if(isinstance(initial\_Population,tuple)):  
 # initial\_Population[i] = tuple\_in\_matrix(initial\_Population[i]) # При необходимости меняем* new\_Solution\_Position = mixing\_matrix(initial\_Population[i], row\_or\_column, t1, t2)  
  
  
 new\_Solution\_Cost = target\_function(matrix\_task, new\_Solution\_Position) *# Вычисление целевой функции* new\_Initial\_Population.append(new\_Solution\_Position)  
 new\_Fitness\_Population.append(new\_Solution\_Cost)  
   
 new\_Initial[matrix\_in\_tuple(new\_Solution\_Position)] = new\_Solution\_Cost  
  
 *# Объядинение популяции* ini = initial | new\_Initial  
  
  
 *# Сортировка* res = list(ini.items())  
 res.sort(key=**lambda** x: x[1], reverse=**False**)  
  
 initial\_Population = list()  
 initial\_Fitness = list()  
 initial = dict()  
 j = 1  
  
 *# Исключаем слабых* **for** i **in** res:  
 **if** (j <= maximum\_Population\_Size):  
 initial\_Population.append(i[0])  
 initial\_Fitness.append(i[1])  
 initial[i[0]] = i[1]  
 **else**:  
 **break** j += 1  
 t += 1  
 print(**"Итерация : "** + str(t) + **" Лучшее "** + str(initial\_Fitness[0]))  
 print(t)  
  
 **return  
  
def** Hungary(matrix):  
 b = matrix.copy()  
 *# Строка и столбец минус 0* **for** i **in** range(len(b)):  
 row\_min = np.min(b[i])  
 **for** j **in** range(len(b[i])):  
 b[i][j] -= row\_min  
 **for** i **in** range(len(b[0])):  
 col\_min = np.min(b[:, i])  
 **for** j **in** range(len(b)):  
 b[j][i] -= col\_min  
 line\_count = 0  
  
 *# Когда количество строк меньше длины матрицы, цикл* **while** (line\_count < len(b)):  
 line\_count = 0  
 row\_zero\_count = []  
 col\_zero\_count = []  
 **for** i **in** range(len(b)):  
 row\_zero\_count.append(np.sum(b[i] == 0))  
 **for** i **in** range(len(b[0])):  
 col\_zero\_count.append((np.sum(b[:, i] == 0)))  
 *# Нажать порядок (ветка или столбец)* line\_order = []  
 row\_or\_col = []  
 **for** i **in** range(len(b[0]), 0, -1):  
 **while** (i **in** row\_zero\_count):  
 line\_order.append(row\_zero\_count.index(i))  
 row\_or\_col.append(0)  
 row\_zero\_count[row\_zero\_count.index(i)] = 0  
 **while** (i **in** col\_zero\_count):  
 line\_order.append(col\_zero\_count.index(i))  
 row\_or\_col.append(1)  
 col\_zero\_count[col\_zero\_count.index(i)] = 0  
 *# Нарисуйте линию, покрывающую 0, и получите матрицу после строки минус минимальное значение и столбец плюс минимальное значение* delete\_count\_of\_row = []  
 delete\_count\_of\_rol = []  
 row\_and\_col = [i **for** i **in** range(len(b))]  
 **for** i **in** range(len(line\_order)):  
 **if** row\_or\_col[i] == 0:  
 delete\_count\_of\_row.append(line\_order[i])  
 **else**:  
 delete\_count\_of\_rol.append(line\_order[i])  
 c = np.delete(b, delete\_count\_of\_row, axis=0)  
 c = np.delete(c, delete\_count\_of\_rol, axis=1)  
 line\_count = len(delete\_count\_of\_row) + len(delete\_count\_of\_rol)  
 *# Когда количество строк равно длине матрицы, выскакиваем* **if** line\_count == len(b):  
 **break** *# Определяем, нужно ли рисовать линию, чтобы покрыть все нули, если она покрывает, операции сложения и вычитания* **if** 0 **not in** c:  
 row\_sub = list(set(row\_and\_col) - set(delete\_count\_of\_row))  
 min\_value = np.min(c)  
 **for** i **in** row\_sub:  
 b[i] = b[i] - min\_value  
 **for** i **in** delete\_count\_of\_rol:  
 b[:, i] = b[:, i] + min\_value  
 **break** row\_ind, col\_ind = linear\_sum\_assignment(b)  
 min\_cost = matrix[row\_ind, col\_ind].sum()  
 best\_solution = list(matrix[row\_ind, col\_ind])  
 print(**"Оптимальное решение полученное с помощью венгерского алгоритма: "** + str(min\_cost))  
 **return** min\_cost  
  
  
  
**def** main():  
 **global** matrix\_task  
 **global** size\_matrix\_task  
  
 size\_matrix\_task = 20  
  
 rd = random.RandomState(10000)  
 *#Генерируем случайную матрицу* matrix\_task = rd.randint(1, 100, size=(20, 20))  
  
 eps = Hungary(matrix\_task)  
  
 iwo(eps)  
**return  
  
  
if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()