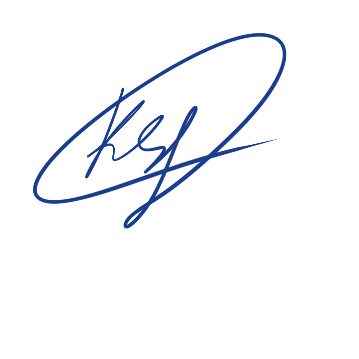
**Федеральное государственное бюджетное образовательное учреждение высшего образования "Белгородский государственный технологический университет им. В.Г. Шухова"**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем.

**Лабораторная работа №1**

Алгоритм отжига.

Выполнил:

Студент группы КБ-211

Коренев Д.Н.

Принял:

Твердохлеб В.В.

Оглавление

[Задание 3](#_Toc177606916)

[Реализация ПО 3](#_Toc177606917)

[Вывод 20](#_Toc177606918)

[Приложения 22](#_Toc177606919)

*Цель работы:* разработка и исследование алгоритма отжига в процессе решения числовой задачи оптимизации.

Задание

1. Изучить теоретическую часть лекционного материала по теме. Ответитьна контрольные вопросы.

2. Используя пример программного кода (исходные файлы в приложениик заданию), а также любой высокоуровневый язык программирования исоответсвующую среду, разработать программное обеспечение сграфическим интерфейсом, позволяющее решить задачу N-ферзей дляN>20. Интерфейс ПО должен обеспечивать ввод следующих параметровалгоритма:

* Максимальная температура
* Минимальная температура
* Коэффициент понижения температуры
* Количество ферзей.
* Количество шагов при постоянном значении температуры.

Необходимо предусмотреть визуализацию лучшего решения в виде шахматной доски, а также необходимо построить график изменения принятых плохих решений, энергии лучшего решения, и температуры.

Реализация ПО

Реализация алгоритма отжига:

Python

class MemberType:

    def \_\_init\_\_(self):

        self.solution = list(range(data\_cb["num\_queens"]))

        self.energy = 0.0

def get\_rand():

    return random.randint(0, data\_cb["num\_queens"] - 1)

def get\_srand():

    return random.random()

def tweak\_solution(member):

    x, y = random.sample(range(data\_cb["num\_queens"]), 2)

    member.solution[x], member.solution[y] \

        = member.solution[y], member.solution[x]

def initialize\_solution(member):

    random.shuffle(member.solution)

    tweak\_solution(member)

def emit\_solution(member):

    board = [['.' for \_ in range(data\_cb["num\_queens"])]

             for \_ in range(data\_cb["num\_queens"])]

    for x in range(data\_cb["num\_queens"]):

        board[x][member.solution[x]] = 'Q'

    for row in board:

        print(' '.join(row))

    print("\n")

def compute\_energy(member):

    conflicts = 0

    board = [['.' for \_ in range(data\_cb["num\_queens"])]

             for \_ in range(data\_cb["num\_queens"])]

    for i in range(data\_cb["num\_queens"]):

        board[i][member.solution[i]] = 'Q'

    dx = [-1, 1, -1, 1]

    dy = [-1, 1, 1, -1]

    for i in range(data\_cb["num\_queens"]):

        x, y = i, member.solution[i]

        for j in range(4):

            tempx, tempy = x, y

            while True:

                tempx += dx[j]

                tempy += dy[j]

                if tempx < 0 or tempx >= data\_cb["num\_queens"] \

                        or tempy < 0 or tempy >= data\_cb["num\_queens"]:

                    break

                if board[tempx][tempy] == 'Q':

                    conflicts += 1

    member.energy = float(conflicts)

def copy\_solution(dest, src):

    dest.solution = src.solution[:]

    dest.energy = src.energy

def main():

    temperature\_x.clear()

    temperature\_y.clear()

    best\_energy\_x.clear()

    best\_energy\_y.clear()

    bad\_solutions\_x.clear()

    bad\_solutions\_y.clear()

    rejected = 0

    emit\_solution\_image(None)

    width, height, channels, data = dpg.load\_image("solution.png")

    dpg.set\_value("texture\_tag", data)

    temperature = data\_cb["init\_temp"]

    current = MemberType()

    working = MemberType()

    best = MemberType()

    best.energy = 100.0

    start\_time = time.time()

    initialize\_solution(current)

    compute\_energy(current)

    copy\_solution(working, current)

    global is\_running

    while (temperature > data\_cb["fin\_temp"]) and (best.energy > 0.0000000):

        if not is\_running:

            break

        temperature\_x.append(float(time.time() - start\_time))

        temperature\_y.append(float(temperature))

        accepted = 0

        for \_ in range(data\_cb["steps\_per\_change"]):

            tweak\_solution(working)

            compute\_energy(working)

            delta = working.energy - current.energy

            if delta <= 0 or math.exp(-delta / temperature) > get\_srand():

                accepted += 1 if delta > 0 else 0

                copy\_solution(current, working)

                if current.energy < best.energy:

                    copy\_solution(best, current)

                else:

                    rejected += 1

                    if rejected % 10 == 0:

                        bad\_solutions\_x.append(float(time.time() - start\_time))

                        bad\_solutions\_y.append(rejected)

            else:

                copy\_solution(working, current)

        best\_energy\_x.append(float(time.time() - start\_time))

        best\_energy\_y.append(float(best.energy))

        temperature \*= data\_cb["alpha"]

    stop()

Полный код программы см. в приложении 1.

Изображение выглядит как текст, снимок экрана, графический дизайн, Шрифт

Автоматически созданное описание  
Рисунок 1. Блок-схема алгоритма.

Внешний вид программы:

Изображение выглядит как снимок экрана, текст, прямоугольный, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 2. Внешний вид приложения

3. Исследовать влияние параметров алгоритма на качество его работы, провести не менее 20 экспериментов при разных комбинациях параметров.

Будем проводить тестирование при значениях: максимальная температура = 100, минимальная температура = 0.01, коэф. уменьшения температуры = 0.99, количество ферзей = 32, количество шагов = 10, если не указано иное значение.

* Максимальная температура
  + 0.1:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 3. При температуре 0.1 алгоритм не достигает оптимальных значений.

* + 1:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 4. Для данных параметров значение максимальной температуры 1 является оптимальным.

* + 25:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 5. При увеличении максимальной температуры, увеличивается затраченное время для вычисления.

* + 100:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 6. При увеличении максимальной температуры, увеличивается затраченное время для вычисления.

* Минимальная температура
  + 0:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 7. При значении минимальной температуры 0 наблюдается нормальная работа алгоритма.

* + 10:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 8. При значении минимальной температуры 10 алгоритм не успевает найти оптимальное решение.

* + 50:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 9. При значении минимальной температуры 50 алгоритм не успевает найти оптимальное решение.

* + 100:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 10. При значении минимальной температуры 100 алгоритм не успевает найти оптимальное решение, не выполняется не одной итерации.

* Коэффициент понижения температуры
  + -1:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 11. При значении коэффициента понижения температуры -1 выполянется одна итерация алгоритма.

* + 0:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 12. При значении коэффициента понижения температуры 0 выполянется одна итерация алгоритма.

* + 0.2:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 13. При значении коэффициента понижения температуры 0.2 алгоритм не успевает найти оптимальное решение

* + 0.99:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 14. При значении коэффициента понижения температуры 0.99 наблюдается нормальная работа алгоритма.

* + 2:

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 15. При значении коэффициента понижения температуры 2 алгоритм не смог найти оптимальное решение за отведенное время.

* Количество ферзей
  + 8:

Изображение выглядит как текст, снимок экрана, прямоугольный, Игры

Автоматически созданное описание  
Рисунок 16. При значении количества ферзей 8 алгоритм находит оптимальное решение мгоновенно.

* + 20:

Изображение выглядит как снимок экрана, текст, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание  
Рисунок 17. При значении количества ферзей 20 алгоритм находит оптимальное решение за 10 с.

* + 30:

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание  
Рисунок 18. При значении количества ферзей 30 алгоритм находит оптимальное решение за 20 с.

* + 50:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 19. При значении количества ферзей 50 алгоритм находит оптимальное решение за 60 с.

* + 100:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 20. При значении количества ферзей 100 алгоритм находит оптимальное решение за 260 с.

* Количество шагов при постоянном значении температуры
  + 1:

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание  
Рисунок 21. При значении количества шагов 1 алгоритм не смог найти оптимальное решение.

* + 5:

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание  
Рисунок 22. При значении количества шагов 5 алгоритм находит оптимальное решение за 1,5 с.

* + 10:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 23. При значении количества шагов 10 алгоритм находит оптимальное решение за 3 с.

* + 50:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 24. При значении количества шагов 50 алгоритм находит оптимальное решение за 12 с.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Параметр** | **Максимальная температура** | **Минимальная температура** | **Коэффициент уменьшения температуры** | **Количество ферзей** | **Количество шагов** | **Результат** |
| **Максимальная температура** | 0.1 | 0.01 | 0.99 | 32 | 10 | Алгоритм не достигает оптимальных значений |
| 1 | 0.01 | 0.99 | 32 | 10 | Алгоритм работает оптимально |
| 25 | 0.01 | 0.99 | 32 | 10 | Время вычисления увеличивается |
| 100 | 0.01 | 0.99 | 32 | 10 | Время вычисления значительно увеличивается |
| **Минимальная температура** | 100 | 0 | 0.99 | 32 | 10 | Нормальная работа алгоритма |
| 100 | 10 | 0.99 | 32 | 10 | Алгоритм не успевает найти оптимальное решение |
| 100 | 50 | 0.99 | 32 | 10 | Алгоритм не успевает найти оптимальное решение |
| 100 | 100 | 0.99 | 32 | 10 | Алгоритм не выполняет ни одной итерации |
| **Коэффициент понижения температуры** | 100 | 0.01 | -1 | 32 | 10 | Выполняется одна итерация алгоритма |
| 100 | 0.01 | 0 | 32 | 10 | Выполняется одна итерация алгоритма |
| 100 | 0.01 | 0.2 | 32 | 10 | Алгоритм не успевает найти оптимальное решение |
| 100 | 0.01 | 0.99 | 32 | 10 | Нормальная работа алгоритма |
| 100 | 0.01 | 2 | 32 | 10 | Алгоритм не смог найти решение за отведенное время |
| **Количество ферзей** | 100 | 0.01 | 0.99 | 8 | 10 | Оптимальное решение найдено мгновенно |
| 100 | 0.01 | 0.99 | 20 | 10 | Оптимальное решение найдено за 10 секунд |
| 100 | 0.01 | 0.99 | 30 | 10 | Оптимальное решение найдено за 20 секунд |
| 100 | 0.01 | 0.99 | 50 | 10 | Оптимальное решение найдено за 60 секунд |
| 100 | 0.01 | 0.99 | 100 | 10 | Оптимальное решение найдено за 260 секунд |
| **Количество шагов при постоянной температуре** | 100 | 0.01 | 0.99 | 32 | 1 | Алгоритм не смог найти оптимальное решение |
| 100 | 0.01 | 0.99 | 32 | 5 | Оптимальное решение найдено за 1.5 секунды |
| 100 | 0.01 | 0.99 | 32 | 10 | Оптимальное решение найдено за 3 секунды |
| 100 | 0.01 | 0.99 | 32 | 50 | Оптимальное решение найдено за 12 секунд |

Таблица 1. Результаты исследования.

Вывод

В ходе проведенного исследования было установлено, что на работу алгоритма отжига в значительной степени влияют такие параметры, как максимальная и минимальная температура, коэффициент понижения температуры, количество шагов и количество ферзей.

1. **Максимальная температура**:
   * При низких значениях максимальной температуры, например, 0.1, алгоритм не достигает оптимальных значений.
   * Оптимальная работа наблюдается при максимальной температуре 1.
   * Увеличение максимальной температуры (25 и 100) приводит к увеличению времени, затраченного на вычисления, хотя алгоритм все же находит решение.
2. **Минимальная температура**:
   * Нормальная работа алгоритма наблюдается при минимальной температуре 0.
   * При повышенных значениях (10, 50, 100) алгоритм не успевает найти оптимальное решение, так как температура падает слишком быстро.
3. **Коэффициент понижения температуры**:
   * При отрицательных значениях коэффициента (-1 и 0) алгоритм выполняет всего одну итерацию и не может найти решение.
   * При коэффициенте 0.99 наблюдается нормальная работа, что позволяет алгоритму медленно снижать температуру и достичь оптимального решения.
   * Значения, превышающие 1 (например, 2), приводят к слишком быстрому охлаждению, что не позволяет алгоритму найти решение за отведенное время.
4. **Количество ферзей**:
   * С увеличением количества ферзей время, необходимое для нахождения решения, возрастает. Однако алгоритм справляется с нахождением оптимального решения для всех протестированных значений, от 8 до 100 ферзей.
   * При 100 ферзях алгоритм находит решение за 260 секунд, что указывает на значительное увеличение времени при росте сложности задачи.
5. **Количество шагов при постоянной температуре**:
   * При малом количестве шагов (1) алгоритм не находит решение.
   * С увеличением количества шагов алгоритм находит решение быстрее. Например, при 50 шагах решение находится за 12 секунд.

Таким образом, для эффективной работы алгоритма отжига необходимо тщательно подбирать значения параметров. Оптимальная комбинация параметров зависит от размера задачи, но в общем случае наиболее удачными значениями для данной задачи являются: максимальная температура 1, минимальная температура 0, коэффициент понижения температуры 0.99.

Приложения

Приложение 1. Полный код программы.

Python 3.12

from PIL import Image, ImageDraw, ImageFont

import dearpygui.dearpygui as dpg

import random

import math

import time

import threading

process = None

is\_running = False

# Data for algorithm

data\_cb = {

    'num\_queens': 8,

    'init\_temp': 100.0,

    'fin\_temp': 0.1,

    'alpha': 0.99,

    'steps\_per\_change': 100

}

# Data for plots, x is time when data captured in s, y is value of data

bad\_solutions\_y = []

bad\_solutions\_x = []

best\_energy\_y = []

best\_energy\_x = []

temperature\_y = []

temperature\_x = []

start\_time = 0

class MemberType:

    def \_\_init\_\_(self):

        self.solution = list(range(data\_cb["num\_queens"]))

        self.energy = 0.0

def get\_rand():

    return random.randint(0, data\_cb["num\_queens"] - 1)

def get\_srand():

    return random.random()

def tweak\_solution(member):

    x, y = random.sample(range(data\_cb["num\_queens"]), 2)

    member.solution[x], member.solution[y] \

        = member.solution[y], member.solution[x]

def initialize\_solution(member):

    random.shuffle(member.solution)

    tweak\_solution(member)

def emit\_solution(member):

    board = [['.' for \_ in range(data\_cb["num\_queens"])]

             for \_ in range(data\_cb["num\_queens"])]

    for x in range(data\_cb["num\_queens"]):

        board[x][member.solution[x]] = 'Q'

    for row in board:

        print(' '.join(row))

    print("\n")

def compute\_energy(member):

    conflicts = 0

    board = [['.' for \_ in range(data\_cb["num\_queens"])]

             for \_ in range(data\_cb["num\_queens"])]

    for i in range(data\_cb["num\_queens"]):

        board[i][member.solution[i]] = 'Q'

    dx = [-1, 1, -1, 1]

    dy = [-1, 1, 1, -1]

    for i in range(data\_cb["num\_queens"]):

        x, y = i, member.solution[i]

        for j in range(4):

            tempx, tempy = x, y

            while True:

                tempx += dx[j]

                tempy += dy[j]

                if tempx < 0 or tempx >= data\_cb["num\_queens"] \

                        or tempy < 0 or tempy >= data\_cb["num\_queens"]:

                    break

                if board[tempx][tempy] == 'Q':

                    conflicts += 1

    member.energy = float(conflicts)

def copy\_solution(dest, src):

    dest.solution = src.solution[:]

    dest.energy = src.energy

def emit\_solution\_image(member):

    global data\_cb

    cell\_size = 30  # Size of each cell in pixels

    border\_size = cell\_size  # Border size to accommodate letters and numbers

    board\_size = data\_cb["num\_queens"] \* cell\_size

    # Adjusted image size to include the border

    image\_size = board\_size + 2 \* border\_size

    # Load the SVG file directly

    queen\_image = Image.open('bQ.png').resize((cell\_size, cell\_size))

    # light wood color

    border\_color = '#D2B48C'

    # Create a blank white image

    image = Image.new('RGB', (image\_size, image\_size), border\_color)

    draw = ImageDraw.Draw(image)

    # Define colors for the checkerboard

    color1 = '#D3D3D3'  # Light gray

    color2 = '#A9A9A9'  # Dark gray

    # Draw the checkerboard pattern

    for x in range(data\_cb["num\_queens"]):

        for y in range(data\_cb["num\_queens"]):

            top\_left = (y \* cell\_size + border\_size,

                        x \* cell\_size + border\_size)

            bottom\_right = (top\_left[0] + cell\_size, top\_left[1] + cell\_size)

            fill\_color = color1 if (x + y) % 2 == 0 else color2

            draw.rectangle([top\_left, bottom\_right], fill=fill\_color)

    if member is not None:

        # Draw the queens

        for x in range(data\_cb["num\_queens"]):

            y = member.solution[x]

            # Calculate the top-left corner of the cell

            top\_left = (y \* cell\_size + border\_size,

                        x \* cell\_size + border\_size)

            # Paste the queen image

            image.paste(queen\_image, top\_left, queen\_image)

    # Draw the border with letters and numbers

    font\_path = "arialbd.ttf"  # Path to a bold TTF font file

    font\_size = 12  # Adjust the font size as needed

    font = ImageFont.truetype(font\_path, font\_size)

    def index\_to\_letters(index):

        letters = []

        while index >= 0:

            letters.append(chr(index % 26 + ord('A')))

            index = index // 26 - 1

        return ''.join(reversed(letters))

    for i in range(data\_cb["num\_queens"]):

        # Draw letters on the top and bottom

        letter = index\_to\_letters(i)

        draw.text((border\_size + i \* cell\_size + cell\_size // 2,

                   border\_size // 2),

                  letter, fill='black', anchor='mm', font=font)

        draw.text((border\_size + i \* cell\_size + cell\_size // 2,

                   image\_size - border\_size // 2),

                  letter, fill='black', anchor='mm', font=font)

        # Draw numbers on the left and right

        number = str(i + 1)

        draw.text((border\_size // 2, border\_size + i \* cell\_size +

                  cell\_size // 2), number, fill='black', anchor='mm', font=font)

        draw.text((image\_size - border\_size // 2, border\_size + i \* cell\_size +

                  cell\_size // 2), number, fill='black', anchor='mm', font=font)

    #  resize image to 4k

    image = image.resize((1000, 1000), Image.Resampling.BICUBIC)

    image.save('solution.png')

def main():

    temperature\_x.clear()

    temperature\_y.clear()

    best\_energy\_x.clear()

    best\_energy\_y.clear()

    bad\_solutions\_x.clear()

    bad\_solutions\_y.clear()

    rejected = 0

    emit\_solution\_image(None)

    width, height, channels, data = dpg.load\_image("solution.png")

    dpg.set\_value("texture\_tag", data)

    temperature = data\_cb["init\_temp"]

    current = MemberType()

    working = MemberType()

    best = MemberType()

    best.energy = 100.0

    start\_time = time.time()

    initialize\_solution(current)

    compute\_energy(current)

    copy\_solution(working, current)

    global is\_running

    while (temperature > data\_cb["fin\_temp"]) and (best.energy > 0.0000000):

        if not is\_running:

            break

        # print(f"Temperature : {temperature}")

        temperature\_x.append(float(time.time() - start\_time))

        temperature\_y.append(float(temperature))

        accepted = 0

        for \_ in range(data\_cb["steps\_per\_change"]):

            tweak\_solution(working)

            compute\_energy(working)

            delta = working.energy - current.energy

            if delta <= 0 or math.exp(-delta / temperature) > get\_srand():

                accepted += 1 if delta > 0 else 0

                copy\_solution(current, working)

                if current.energy < best.energy:

                    copy\_solution(best, current)

                else:

                    rejected += 1

                    if rejected % 10 == 0:

                        bad\_solutions\_x.append(float(time.time() - start\_time))

                        bad\_solutions\_y.append(rejected)

            else:

                copy\_solution(working, current)

        # print(f"Best energy = {best.energy}")

        best\_energy\_x.append(float(time.time() - start\_time))

        best\_energy\_y.append(float(best.energy))

        temperature \*= data\_cb["alpha"]

        # Fit the axis data

        if len(temperature\_x) > 0:

            dpg.set\_value("series\_tag\_temp", [temperature\_x, temperature\_y])

            # dpg.fit\_axis\_data("y\_axis\_temp")

            dpg.fit\_axis\_data("x\_axis\_temp")

            dpg.set\_value("temp\_text", f"{temperature:.2f}")

        if len(best\_energy\_x) > 0:

            dpg.set\_value("series\_tag\_energy", [best\_energy\_x, best\_energy\_y])

            # dpg.fit\_axis\_data("y\_axis\_energy")

            dpg.set\_axis\_limits("y\_axis\_energy", -0.5,

                                max(best\_energy\_y) + 0.5)

            dpg.fit\_axis\_data("x\_axis\_energy")

            dpg.set\_value("energy\_text", f"{best.energy:.2f}")

        if len(bad\_solutions\_x) > 0:

            dpg.set\_value("series\_tag\_bad", [bad\_solutions\_x, bad\_solutions\_y])

            dpg.fit\_axis\_data("x\_axis\_bad")

            dpg.set\_axis\_limits("y\_axis\_bad", -0.5, max(bad\_solutions\_y) + 0.5)

            dpg.set\_value("bads\_text", f"{bad\_solutions\_y[-1]}")

        if len(temperature\_x) > 0:

            dpg.set\_value("time\_text", f"{temperature\_x[-1]:.2f}s")

    # emit\_solution(best)

    emit\_solution\_image(best)

    # update texture

    width, height, channels, data = dpg.load\_image("solution.png")

    dpg.set\_value("texture\_tag", data)

    stop()

def run():

    global is\_running

    global process

    if process is None:

        is\_running = True

        process = threading.Thread(target=main)

        process.start()

def stop():

    global is\_running

    global process

    is\_running = False

    if process is not None:

        try:

            process.join()

        except:

            pass

    process = None

def update\_layout():

    port\_width = dpg.get\_viewport\_client\_width()

    port\_height = dpg.get\_viewport\_client\_height()

    inputs\_height = 26

    board\_size = min(port\_width, port\_height - inputs\_height \* 6)

    plots\_width = port\_width - board\_size / 0.97

    plots\_height = board\_size / 3 - 3  # / 3.031

    inputs\_width = min(max(port\_width \* 0.15, 100), 500)

    outputs\_width = min(max(port\_width \* 0.15, 100), 500)

    if dpg.does\_item\_exist("board"):

        dpg.configure\_item("board", width=board\_size, height=board\_size)

    if dpg.does\_item\_exist("max\_temp"):

        dpg.configure\_item("max\_temp", width=inputs\_width)

    if dpg.does\_item\_exist("min\_temp"):

        dpg.configure\_item("min\_temp", width=inputs\_width)

    if dpg.does\_item\_exist("alpha"):

        dpg.configure\_item("alpha", width=inputs\_width)

    if dpg.does\_item\_exist("num\_queens"):

        dpg.configure\_item("num\_queens", width=inputs\_width)

    if dpg.does\_item\_exist("steps"):

        dpg.configure\_item("steps", width=inputs\_width)

    if dpg.does\_item\_exist("temp\_text"):

        dpg.configure\_item("temp\_text", width=outputs\_width)

    if dpg.does\_item\_exist("energy\_text"):

        dpg.configure\_item("energy\_text", width=outputs\_width)

    if dpg.does\_item\_exist("bads\_text"):

        dpg.configure\_item("bads\_text", width=outputs\_width)

    if dpg.does\_item\_exist("time\_text"):

        dpg.configure\_item("time\_text", width=outputs\_width)

    if dpg.does\_item\_exist("plot\_temp"):

        dpg.configure\_item("plot\_temp", width=plots\_width, height=plots\_height)

    if dpg.does\_item\_exist("plot\_energy"):

        dpg.configure\_item("plot\_energy", width=plots\_width,

                           height=plots\_height)

    if dpg.does\_item\_exist("plot\_bads"):

        dpg.configure\_item("plot\_bads", width=plots\_width, height=plots\_height)

    if dpg.does\_item\_exist("y\_axis\_temp"):

        dpg.set\_axis\_limits("y\_axis\_temp", 0, data\_cb["init\_temp"])

if \_\_name\_\_ == "\_\_main\_\_":

    dpg.create\_context()

    dpg.create\_viewport(title='N-Queen Task', width=800, height=600)

    dpg.set\_viewport\_resize\_callback(update\_layout)

    # print(dpg.get\_viewport\_client\_width())

    # print(dpg.get\_viewport\_client\_height())

    width, height, channels, data = dpg.load\_image("solution.png")

    with dpg.texture\_registry(show=False):

        dpg.add\_dynamic\_texture(

            width=width, height=height, default\_value=data, tag="texture\_tag")

    with dpg.window(label="Example Window", tag="fullscreen"):

        with dpg.theme(tag="plot\_theme\_temp"):

            with dpg.theme\_component(dpg.mvLineSeries):

                dpg.add\_theme\_color(

                    dpg.mvPlotCol\_Line, (255, 99, 99),

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_Marker, dpg.mvPlotMarker\_None,

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_MarkerSize, 0,

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_LineWeight, 3.0,

                    category=dpg.mvThemeCat\_Plots)

        with dpg.theme(tag="plot\_theme\_energy"):

            with dpg.theme\_component(dpg.mvLineSeries):

                dpg.add\_theme\_color(

                    dpg.mvPlotCol\_Line, (22, 90, 255),

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_Marker, dpg.mvPlotMarker\_None,

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_MarkerSize, 0,

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_LineWeight, 3.0,

                    category=dpg.mvThemeCat\_Plots)

        with dpg.theme(tag="plot\_theme\_bads"):

            with dpg.theme\_component(dpg.mvLineSeries):

                dpg.add\_theme\_color(

                    dpg.mvPlotCol\_Line, (123, 123, 123),

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_Marker, dpg.mvPlotMarker\_None,

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_MarkerSize, 0,

                    category=dpg.mvThemeCat\_Plots)

                dpg.add\_theme\_style(

                    dpg.mvPlotStyleVar\_LineWeight, 3.0,

                    category=dpg.mvThemeCat\_Plots)

        def on\_value\_change\_max\_temp(sender, app\_data, user\_data):

            stop()

            user\_data['init\_temp'] = float(app\_data)

            dpg.set\_axis\_limits("y\_axis\_temp", 0, data\_cb["init\_temp"])

        def on\_value\_change\_min\_temp(sender, app\_data, user\_data):

            stop()

            user\_data['fin\_temp'] = float(app\_data)

        def on\_value\_change\_alpha(sender, app\_data, user\_data):

            stop()

            user\_data['alpha'] = float(app\_data)

        def on\_value\_change\_steps(sender, app\_data, user\_data):

            stop()

            user\_data['steps\_per\_change'] = int(app\_data)

        def on\_value\_change\_num\_queens(sender, app\_data, user\_data):

            stop()

            user\_data['num\_queens'] = int(app\_data)

            emit\_solution\_image(None)

            width, height, channels, data = dpg.load\_image("solution.png")

            dpg.set\_value("texture\_tag", data)

        # Максимальная температура

        with dpg.group(horizontal=True):

            with dpg.group():

                dpg.add\_input\_float(label="Max temperature",

                                    default\_value=100,

                                    width=200,

                                    callback=on\_value\_change\_max\_temp,

                                    user\_data=data\_cb,

                                    min\_value=0.0,

                                    step=1.0,

                                    tag="max\_temp")

                # Минимальная температура

                dpg.add\_input\_float(label="Min temperature",

                                    default\_value=0.01,

                                    width=200,

                                    callback=on\_value\_change\_min\_temp,

                                    user\_data=data\_cb,

                                    min\_value=0.0,

                                    step=0.01,

                                    tag="min\_temp")

                # Коэффициент понижения температуры

                dpg.add\_input\_float(label="Temperature reduction coefficient",

                                    default\_value=0.99,

                                    width=200,

                                    callback=on\_value\_change\_alpha,

                                    user\_data=data\_cb,

                                    min\_value=0.0,

                                    max\_value=1.0,

                                    step=0.01,

                                    tag="alpha")

                # Количество ферзей

                dpg.add\_input\_int(label="Number of queens",

                                  default\_value=8,

                                  width=200,

                                  callback=on\_value\_change\_num\_queens,

                                  user\_data=data\_cb,

                                  min\_value=1, max\_value=30,

                                  tag="num\_queens")

                # Количество шагов при постоянном значении температуры

                dpg.add\_input\_int(label="Number of steps at constant "

                                  + "temperature",

                                  default\_value=10,

                                  width=200,

                                  callback=on\_value\_change\_steps,

                                  user\_data=data\_cb,

                                  tag="steps")

            with dpg.group():

                # text output

                dpg.add\_input\_text(default\_value="...",

                                   readonly=True,

                                   tag="temp\_text",

                                   width=50)

                dpg.add\_input\_text(default\_value="...",

                                   readonly=True,

                                   tag="energy\_text",

                                   width=50)

                dpg.add\_input\_text(default\_value="...",

                                   readonly=True,

                                   tag="bads\_text",

                                   width=50)

                dpg.add\_input\_text(default\_value="...",

                                   readonly=True,

                                   tag="time\_text",

                                   width=50)

            with dpg.group():

                dpg.add\_text("Temperature")

                dpg.add\_text("Best Energy")

                dpg.add\_text("Bad Solutions")

                dpg.add\_text("Time")

        with dpg.group(horizontal=True):

            dpg.add\_button(label="run", callback=run)

            dpg.add\_button(label="stop", callback=stop)

        text\_width = dpg.get\_viewport\_client\_width() \* 5 / 10

        text\_height = dpg.get\_viewport\_client\_width() \* 5 / 10

        with dpg.group(horizontal=True):

            dpg.add\_image("texture\_tag",

                          width=text\_width,

                          height=text\_height, tag="board")

            emit\_solution\_image(None)

            width, height, channels, data = dpg.load\_image("solution.png")

            dpg.set\_value("texture\_tag", data)

            plots\_width = text\_width / 2

            plots\_height = text\_height / 3.031  # 4.548

            with dpg.group():

                with dpg.plot(label="Temperature",

                              width=plots\_width,

                              height=plots\_height,

                              tag="plot\_temp"):

                    dpg.add\_plot\_legend()

                    dpg.add\_plot\_axis(

                        dpg.mvXAxis, label="time, s", tag="x\_axis\_temp")

                    dpg.add\_plot\_axis(

                        dpg.mvYAxis, label="temperature", tag="y\_axis\_temp")

                    dpg.set\_axis\_limits("y\_axis\_temp", 0, data\_cb["init\_temp"])

                    dpg.add\_line\_series(

                        temperature\_x, temperature\_y, parent="y\_axis\_temp",

                        tag="series\_tag\_temp")

                dpg.bind\_item\_theme("series\_tag\_temp", "plot\_theme\_temp")

                with dpg.plot(label="Best Energy",

                              width=plots\_width,

                              height=plots\_height,

                              tag="plot\_energy"):

                    dpg.add\_plot\_legend()

                    dpg.add\_plot\_axis(

                        dpg.mvXAxis, label="time, s", tag="x\_axis\_energy")

                    dpg.add\_plot\_axis(

                        dpg.mvYAxis, label="energy", tag="y\_axis\_energy")

                    dpg.add\_line\_series(

                        best\_energy\_x, best\_energy\_y, parent="y\_axis\_energy",

                        tag="series\_tag\_energy")

                dpg.bind\_item\_theme("series\_tag\_energy", "plot\_theme\_energy")

                with dpg.plot(label="Bad Solutions",

                              width=plots\_width,

                              height=plots\_height,

                              tag="plot\_bads"):

                    dpg.add\_plot\_legend()

                    dpg.add\_plot\_axis(

                        dpg.mvXAxis, label="time, s", tag="x\_axis\_bad")

                    dpg.add\_plot\_axis(

                        dpg.mvYAxis, label="bad solutions", tag="y\_axis\_bad")

                    dpg.add\_line\_series(

                        bad\_solutions\_x, bad\_solutions\_y, parent="y\_axis\_bad",

                        tag="series\_tag\_bad")

                dpg.bind\_item\_theme("series\_tag\_bad", "plot\_theme\_bads")

    update\_layout()

    dpg.setup\_dearpygui()

    dpg.set\_primary\_window("fullscreen", True)

    dpg.show\_viewport()

    dpg.start\_dearpygui()

    dpg.destroy\_context()