Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 13

по курсу «Методы оптимизации»

«Реализация генетического алгоритма для восстановления фотографии»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Реализовать генетический алгоритм для восстановления реального изображения 100 на 100 пикселей, визуализировать процесс поиска на графиках.

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```
1
2  using Random, Plots, Images, FileIO
3
4  #
5  target_image = load("input.jpg")
6  target_image = Float64.(channelview(RGB.(target_image)))
7
8  #

9  original_image = colorview(RGB, target_image)
10
11 function image_to_vector(img)
12     return vec(reshape(img, :))
13 end
14
15 function vector_to_image(vec, height, width)
16     reshaped = reshape(vec, 3, height, width)
17     return colorview(RGB, reshaped)
18 end
19
20 #

21 function plot_comparison!(gen_vec, anim, height, width, gen, fintess)
22     generated_img = vector_to_image(gen_vec, height, width)
23
24     p = plot(
25         plot(original_image, title="                    ", axis=false,
       ticks=false, border=false),
26         plot(generated_img, title="                     $gen -
                          $fintess", axis=false, ticks=false, border=
       false),
27         layout = (1, 2),
28         size = (800, 400)
29     )
```

```
30
31      frame(anim, p)
32  end
33
34  #
35  function plot_final_comparison(best_vec, height, width)
36      generated_img = vector_to_image(best_vec, height, width)
37
38      p = plot(
39          plot(original_image, title="Original", axis=false, ticks=false,
        border=false),
40          plot(generated_img, title="Final Result", axis=false, ticks=
        false, border=false),
41          layout = (1, 2),
42          size = (800, 400)
43      )
44      display(p)
45  end
46
47  #

48  function initialize_population(size, dims)
49      return [rand(Float64, dims) for _ in 1:size]
50  end
51
52  function fitness(candidate, target)
53      return -sum((candidate .- target).^2)
54  end
55
56  function crossover(p1, p2)
57      point = rand(1:length(p1)-1)
58      return vcat(p1[1:point], p2[point+1:end])
59  end
60
61  function mutate(child, mutation_rate)
62      for i in eachindex(child)
63          if rand() < mutation_rate
64              child[i] += randn() * 0.05
65              child[i] = clamp(child[i], 0.0, 1.0)
66          end
67      end
68      return child
69  end
70
71  function select_parents(population, target, num_parents)
72      sorted = sort(population, by = x -> fitness(x, target), rev = true)
```

```
    return sorted[1:num_parents]
end

function genetic_algorithm(target_vec, height, width; pop_size=50,
    generations=500, mutation_rate=0.05, save_path="evolution.gif")
    population = initialize_population(pop_size, length(target_vec))
    anim = Animation()

    best_match_per_pixel = copy(population[1])
    best_error_per_pixel = abs.(best_match_per_pixel .- target_vec)

    for gen in 1:generations
        #

        plot_comparison!(best_match_per_pixel, anim, height, width, gen,
     fitness(best_match_per_pixel,target_vec))

        parents = select_parents(population, target_vec, 4)
        offspring = []
        while length(offspring) < pop_size
            p1, p2 = rand(parents, 2)
            child = crossover(p1, p2)
            push!(offspring, mutate(child, mutation_rate))
        end

        population = offspring
        current_best = select_parents(population, target_vec, 1)[1]

        current_error = abs.(current_best .- target_vec)
        for i in eachindex(current_error)
            if current_error[i] < best_error_per_pixel[i]
                best_match_per_pixel[i] = current_best[i]
                best_error_per_pixel[i] = current_error[i]
            end
        end
    end

    gif(anim, save_path, fps=20)
    return best_match_per_pixel
end

#
height, width = size(target_image)[2], size(target_image)[3]
target_vec = image_to_vector(target_image)

best_match = genetic_algorithm(target_vec, height, width)
```

3 Результаты

Результаты запуска представлены на рисунках 1.

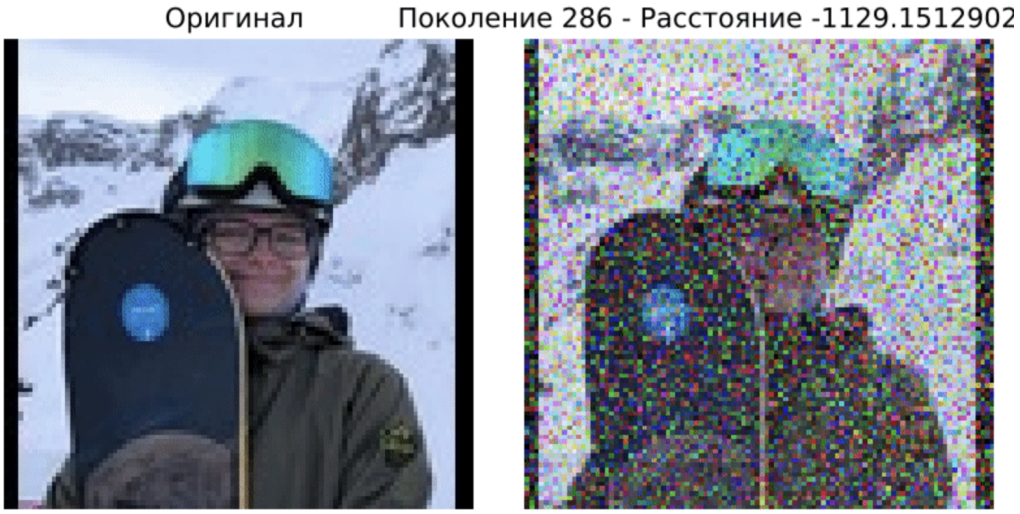

Рис. 1 — Визуализация

4 Выводы

В результате данной лабораторной работы был реализован генетический алгоритм для восстановления реального изображения из случайной последовательности, алгоритм из прошлой лабораторной работы был модифицирован и были получены отличные результаты сходимости (изображение четко идентифицируется).