ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Кафедра

«Криптология и Кибербезопасность»

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

«Оптимизация размещения информации в сжатых JPEG-изображениях с использованием генетического алгоритма»

подпись, дата

Исполнитель:

студент гр. Б21-515 Грущин И.М.

подпись, дата

Научный

руководитель: Борзунов Г.И.

подпись, дата

Зам. зав. каф. № 42: Когос К. Г.

Москва – 2024

РЕФЕРАТ

Отчет содержит 41 с., 5 рис., 7 источн., 5 прил.

СТЕГАНОГРАФИЯ, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, JPEG, QIM, ОПТИМИЗАЦИЯ, ВСТРАИВАНИЕ

Объект исследования данной работы — коэффициенты ДКП. Предмет исследования — схема встраивания информации в коэффициенты ДКП.

Цель работы состоит в предложении варианта схемы встраивания информации в ДКП-коэффициенты, обеспечивающей с помощью генетического алгоритма повышение качества встраивания, характеризуемого значениями MSE, PSNR по сравнению с известными алгоритмами.

В данной работе были исследованы методы встраивания информации в ДКП-коэффициенты JPEG изображений, посредством генетического алгоритма оптимизирована схема встраивания информации в JPEG изображения методом QIM.

Область применения результатов работы — скрытое встраивание и передача информации.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 4

ВВЕДЕНИЕ 5

1 Алгоритмы встраивания информации в ДКП-коэффициенты сжатого JPEG-изображения 6

1.1 Встраивание битов в выбранные коэффициенты 6

1.2 Метод встраивания QIM 7

1.3 Выбор шага квантования 8

2 Оптимизация генетическим алгоритмом 11

2.1 Оценка качества встраивания 11

2.2 Схема генетического алгоритма 12

2.3 Программная реализация 14

2.4 Вычислительный эксперимент 14

ЗАКЛЮЧЕНИЕ 17

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 18

ПРИЛОЖЕНИЕ А 20

ПРИЛОЖЕНИЕ Б 22

ПРИЛОЖЕНИЕ В 30

ПРИЛОЖЕНИЕ Г 36

ПРИЛОЖЕНИЕ Д 41

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящем отчете применяют следующие термины с соответствующими определениями, обозначениями и сокращениями:

ГА	—	Генетический алгоритм.
ДКП	—	Дискретное косинусное преобразование.
AC-коэффициент	—	Все ДКП-коэффициенты в блоке, за исключением DC-коэффициента.
DC-коэффициент	—	Коэффициент в верхнем левом углу матрицы блока ДКП-коэффициентов.
JPEG	—	Растровый графический формат, применяемый для хранения сжатых изображений.
MSE	—	Среднеквадратичная ошибка (Mean Squared Error).
PSNR	—	Пиковое отношение сигнала к шуму (Peak Signal-to-Noise Ratio)
QIM	—	Quantization Index Modulation – метод встраивания информации в ДКП-коэффициенты.
TIFF	—	Формат хранения растровых графических изображений.

ВВЕДЕНИЕ

В данной работе проводится исследование и оптимизация методов размещения информации в сжатых в формате JPEG цифровых изображениях с использованием генетического алгоритма. Развитие современных техник передачи и хранения данных ведет к необходимости стеганографической обработки изображений с целью эффективного и незаметного встраивания информации. В этом контексте, оптимизация процесса стеганографии в JPEG-изображениях становится актуальной задачей, связанной не только с сохранением качества изображений, но и с обеспечением безопасной передачи конфиденциальной информации.

Применение генетического алгоритма для оптимизации размещения информации в JPEG-изображениях может значительно улучшить эффективность стеганографических методов, обеспечивая высокую степень стойкости и минимальные изменения в качестве изображений.

Целью данной работы является разработка эффективного метода оптимизации размещения информации в сжатых JPEG-изображениях, основанного на использовании генетических алгоритмов.

Целью данной работы является разработка схемы встраивания информации в ДКП-коэффициенты, обеспечивающей с помощью генетического алгоритма повышение качества встраивания, характеризуемого значениями MSE, PSNR по сравнению с известными алгоритмами. Результаты проведенной работы изложены в настоящем отчете, содержащем 2 главы.

В первой главе рассматриваются существующие алгоритмы встраивания информации в сжатого JPEG-изображения а также описание выбранного алгоритма.

Во второй главе приводится разработанный генетический алгоритм и результаты проделанной работы.

1 Алгоритмы встраивания информации в ДКП-коэффициенты сжатого JPEG-изображения

Можно выделить два основных подхода к встраиванию частей секретного сообщения в ДКП коэффициенты JPEG-изображений:

- Непосредственное встраивание битов в выбранные коэффициенты.

- Изменение групп выбранных коэффициентов таким образом, чтобы они удовлетворяли определённым соотношениям в зависимости от встраиваемых битов [1].

Первый подход позволяет обеспечить большую ёмкость встраивания по сравнению со вторым подходом. Алгоритмы, работающие с отдельными ДКП-коэффициентами, могут использоваться как для встраивания в изображения цифровых водяных знаков, так и для встраивания произвольных сообщений, в то время как алгоритмы, работающие с группами ДКП-коэффициентов, преимущественно используются для встраивания цифровых водяных знаков. В связи с большей применимостью генетического алгоритма для выбора отдельных коэффициентов в данной работе исследуется первый подход.

1.1 Встраивание битов в выбранные коэффициенты

Известно достаточно много стеганографических методов и алгоритмов, работающих с JPEG-изображениями. Основное отличие между ними заключается в способах формирования множества ДКП-коэффициентов, непосредственно используемых для записи битов встраиваемого сообщения.

Например, алгоритмы, представленные в [1, 2], используют для встраивания сообщения все ненулевые ДКП-коэффициенты блоков изображения-контейнера. Данные алгоритмы основаны на методе PM1, согласно которому в один коэффициент встраивается один бит сообщения. Встраивание заключается в уменьшении или увеличении коэффициента на единицу в зависимости от значения встраиваемого бита.

В статье [3] для записи битов сообщения используются только ДКП-коэффициенты, равные по модулю заранее заданной величине L. Данная величина является параметром соответствующего алгоритма.

При встраивании единичного бита абсолютное значение коэффициента увеличивается на единицу, при встраивании нулевого бита — остаётся без изменений. При этом все прочие коэффициенты, не включённые в пространство сокрытия, также увеличиваются по модулю на единицу, чтобы при извлечении сообщения не возникло неоднозначности.

Иным подходом является выбор определенной частотной области в блоке ДКП-коэффициентов, одинаковой для различных блоков, вне зависимости от характеристик конкретного изображения. Условно блок коэффициентов можно поделить на три области — низкочастотную, среднечастотную и высокочастотную.

Помимо выбора области встраивания для решения задачи встраивания необходимо выбрать непосредственно алгоритм встраивания — модификации или замены ДКП-коэффициентов. например метод QIM [4, 5] или замена наименьшего значащего бита [6].

В статье [7] рассматривается применение simple-QIM алгоритма в высокочастотной области. В настоящем исследовании произведена попытка оптимизировать данный метод посредством генетического алгоритма.

1.2 Метод встраивания QIM

Основная идея метода QIM [4] заключается в изменении элемента данных изображения в зависимости от значения бита секретного сообщения. Изменяемое число делится на заранее определённый коэффициент, а затем округляется. Этот коэффициент называется шагом квантования q. Формула (1) встраивания бита сообщения b_i при этом имеет вид:

$$c' = q \cdot \lfloor \frac{c}{q} \rfloor + \frac{q}{2} \cdot b_i \,, \qquad (1)$$

где $c`$ - коэффициент ДКП до встраивания;

c - коэффициент ДКП после встраивания;

b_i -бит секретного сообщения;

$\lfloor \dots \rfloor$ - целая часть от деления.

Извлечение выполняется по формуле (2):

$$b_i' = arg \min_{p \in [0,1]} \left| c'' - c_p' \right|, \quad (2)$$

где c`` - коэффициент ДКП, содержащий бит сообщения;

c_0` вычисляется по формуле (3);

c_1` вычисляется по формуле (4);

$$c_0'' = q \cdot \lfloor \frac{c''}{q} \rfloor, \quad (3)$$

$$c_1'' = q \cdot \lfloor \frac{c''}{q} \rfloor + \frac{q}{2}, \quad (4)$$

Эффективность встраивания в значительной степени зависит от величины шага квантования.

1.3 Выбор шага квантования

В вышеупомянутой статье [4] в качестве шага квантования используется наименьший ДКП-коэффициент из самых редких по области невстраивания. На рисунке 1 показано разбиение блока ДКП-коэффициентов на DC-коэффициент, области встраивания и невстраивания. В данном случае в область невстраивания входят со 2 по 54 коэффициент, а в область встраивания с 55 по 64 коэффициент.

Рисунок 1 — Разбиение блока ДКП-коэффициентов на DC-коэффициент, области встраивания и невстраивания

В статье [5] исследователи также показывают нецелесообразность использования шага квантования за пределами отрезка [3; 20], поэтому введём данное ограничение при подсчёте шага квантования.

Итоговая блок-схема алгоритма приведена на рисунке 2.

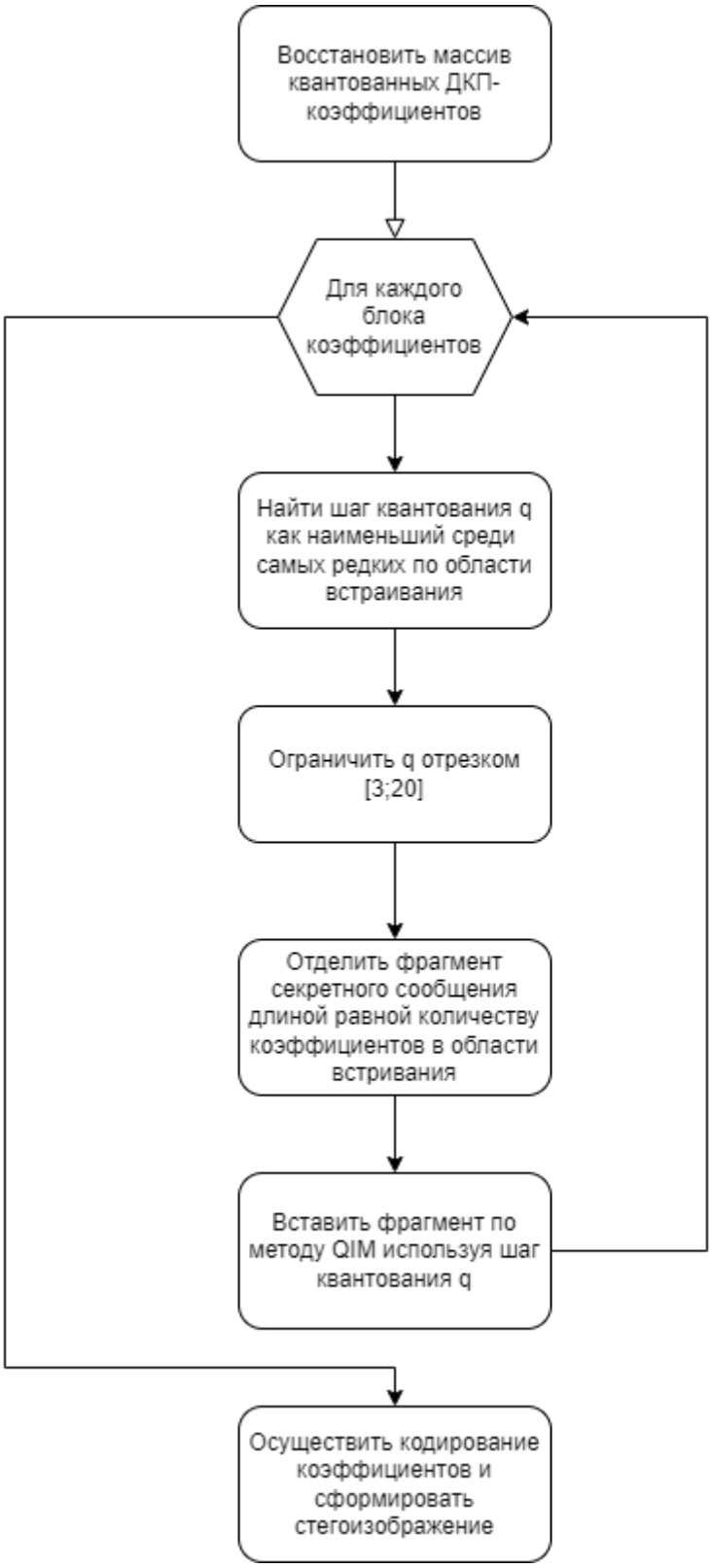

Рисунок 2 — Блок-схема QIM алгоритма, используемого в работе

2 Оптимизация генетическим алгоритмом

Использование всех коэффициентов может быть не целесообразно, вероятно при использовании лишь части коэффициентов возможно сохранить тот же уровень качества встраивания. Задача генетического алгоритма состоит в том, чтобы выбрать среди 10 коэффициентов области невстраивания те, встраивание в которые позволит сохранить емкость стегоконтейнера при той же оценке качества встраивания, либо же повысить качество встраивания при уменьшении объема встраиваемой информации

2.1 Оценка качества встраивания

Как оценку качества встраивания зачастую используют оценки MSE (среднеквадратичная ошибка) и PSNR (пиковое отношение сигнала к шуму.

MSE для двух монохромных изображений I и K размера m×n, одно из которых считается зашумленным приближением другого, вычисляется по формуле (5).

$$MSE = \frac{1}{m \cdot n} \cdot \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I(i,j) - K(i,j)|^2 \quad (5)$$

где I(i, j) — пиксель изображения I с координатами i и j,
K(i, j) — пиксель изображения K с координатами i и j,

Тогда PSNR считается по формуле (6).

$$PSNR = 10 \cdot log_{10}\left(\frac{MAX_I}{MSE}\right) \quad (6)$$

где MAX_I — это максимальное значение, принимаемое пикселем изображения. В нашем случае $MAX_I = 255$.

Типичным значением PSNR для сжатых изображений считается 30-40 дБ.

2.2 Схема генетического алгоритма

В качестве хромосом используется вектор длиной 10, каждое значение которого обозначает используется ли данный коэффициент для встраивания информации: 1 — используется, 0 — не используется. Скрещивание происходит посредством кроссинговера с 10% вероятностью мутации. На рисунке 5 изображена блок-схема ГА.

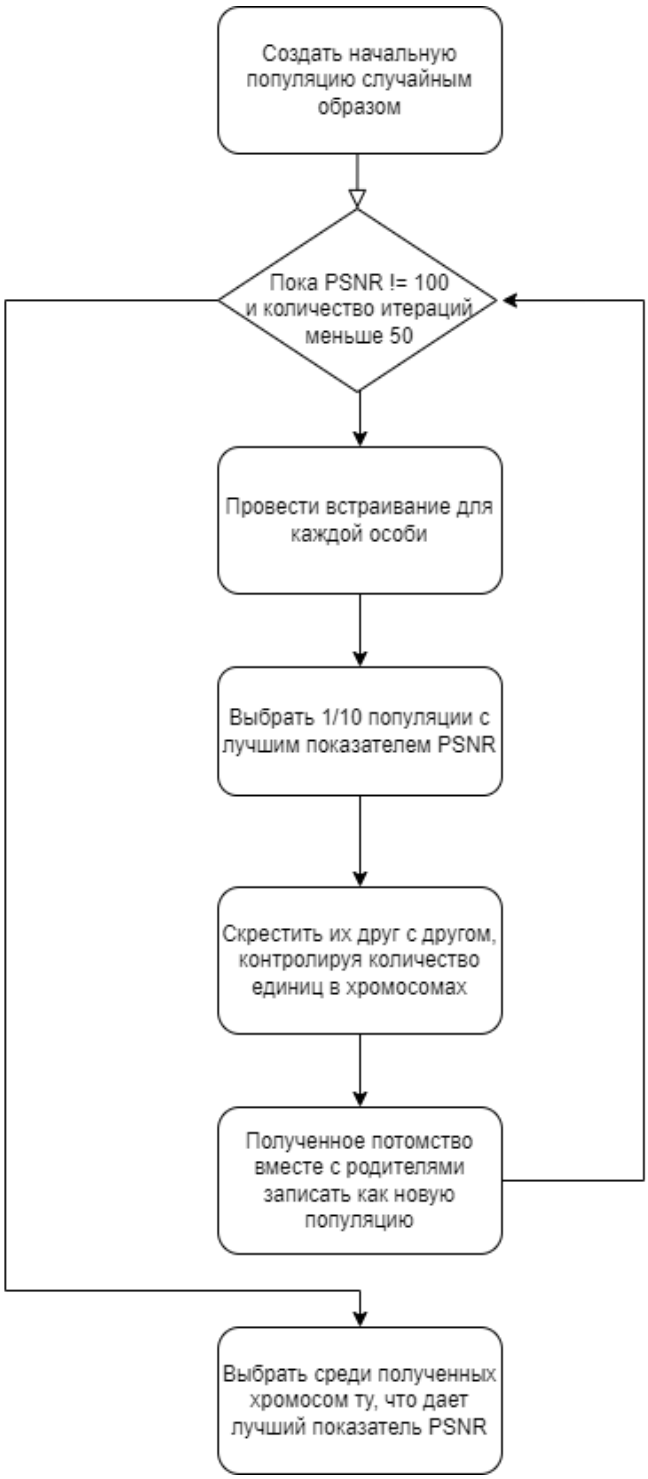

Рисунок 3 — Блок-схема ГА

В работе рассматривается кодирование 6 коэффициентами на блок, в связи с чем возникает необходимость дополнительного воздействия на хромосомы для получения хромосом с необходимым количеством единиц. Для этого был выбран путь повторения скрещивания с мутацией, пока не будет достигнуто необходимое количество единиц в векторе хромосомы потомка.

2.3 Программная реализация

Для написания программы, реализующих встраивание, извлечение информации и эволюций хромосом был выбран язык C++. Для обработки JPEG изображений использовалась библиотека libjpeg-turbo. Также был написан скрипт на языке Python для вычисления оценки PSNR. Весь исходный код доступен на странице сервиса Github [8], а также в приложении.

2.4 Вычислительный эксперимент

В рамках вычислительного эксперимента было проведено встраивание текста на английском языке объёмом в 5767 символов (46136 бит) в изображение размером 800×553 для тренировки ГА. По результатам, представленным на рисунке 4, была отобрана хромосома [1 1 0 0 1 0 1 1 0 1] оценка PSNR для которой составила ~33,983 дБ.

PSNR	Chromosome
32,934	0 1 1 1 0 1 0 1 1 0
33,015	0 1 1 1 1 1 0 1 0 0
33,063	0 1 1 1 0 1 1 1 0 0
33,165	0 1 0 1 1 1 0 1 1 0
33,204	0 1 1 0 0 1 1 1 1 0
33,256	0 0 1 1 1 1 1 0 0 1
33,259	0 0 1 1 0 1 1 1 1 0
33,293	0 1 1 1 0 0 1 1 1 0
33,313	0 1 0 1 0 1 1 1 1 0
33,314	0 1 1 1 1 0 1 1 0 0
33,387	1 1 1 0 0 1 1 1 0 0
33,544	0 1 0 1 0 0 1 1 1 1
33,545	1 0 0 1 1 1 0 1 1 0
33,704	1 0 0 1 1 1 1 1 0 0
33,714	1 0 1 1 1 0 1 0 0 1
33,780	0 0 0 1 1 0 1 1 1 1
33,834	1 0 1 0 1 0 1 0 1 1
33,910	1 0 0 0 1 1 1 0 1 1
33,922	1 1 0 0 1 0 1 0 1 1
33,983	1 1 0 0 1 0 1 1 0 1

Рисунок 4 — Результаты тренировки ГА

Полученная схема вставки информации в ДКП коэффициенты использовалась для встраивания того же текста в 5 стандартных изображений из базы USC-SIPI (раздел Miscallenous) с разрешениями 512×512, таких как «Airplane», и др, конвертированных в формат JPEG.

На рисунке 5 показаны результаты сравнительного эксперимента на полученной с помощью ГА хромосоме и хромосоме [0 0 0 0 1 1 1 1 1 1], которая соответствует запуску детерминированного алгоритма со встраиванием в 6 последних (в зигзаг-порядке) коэффициентов. Запуск производился на 6 изображениях: kok (тренировочное для ГА), splash (4.2.01 из базы USC-SIPI), babuin (4.2.05 из базы USC-SIPI), landscape (4.2.07 из базы USC-SIPI), pepper (4.2.06 из базы USC-SIPI), airplane (4.2.03 из базы USC-SIPI).

		chromosome		
		0 0 0 0 1 1 1 1 1 1	1 1 0 0 1 0 1 1 0 1	Advantage
PSNR For Image	kok	33,624	33,983	1,07%
	splash	27,195	27,435	0,88%
	babuin	25,758	26,068	1,20%
	landscape	26,704	26,975	1,02%
	pepper	26,276	26,518	0,92%
	airplane	27,138	27,366	0,84%
			Average advantage For test images	0,97%

Рисунок 5 — Сравнительный анализ хромосом

Из таблицы видно, что во всех случаях использование схемы, полученной с помощью ГА дает преимущество в среднем на 1%, значит цель в виде получения схемы оптимального размещения скрытой информации достигнута.

ЗАКЛЮЧЕНИЕ

В настоящей работе были рассмотрены основы кодирования сжатых JPEG-изображений, основные алгоритмы встраивания скрытой информации и способы оценки качества встраивания.

Основным результатом работы является схема оптимизированного встраивания скрытой информации в сжатые JPEG изображения.

Иными результатами работы являются разработанный алгоритм встраивания, а также программа, реализующая ГА встраивания информации в JPEG-изображениях, и результаты эксперимента, указанные в пункте 4 главы 2.

Цель данной работы выполнена: в главе 2 описана найденная с помощью ГА схема встраивания информации в изображение с наименьшим искажением, дающая на ~1% большую величину PSNR.

Перспективы продолжения работы состоят в исследовании возможности расширении области встраивания, в которой происходит выбор коэффициентов посредством ГА, а также в определении индивидуального выбора хромосомы для каждого изображения-контейнера.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Yu, L. PM1 steganography in JPEG images using genetic algorithm / L. Yu, Y. Zhao, R. Ni, Z. Zhu // Soft Computing. – 2009. – Vol. 13(4). – P. 393-400. – DOI: 10.1007/s00500-008-0327-7.

2. Евсютин, О.О. Улучшенный алгоритм встраивания информации в сжатые цифровые изображения на основе метода PM1 / О.О. Евсютин, А.С. Кокурина, А.А. Шелупанов, И.И. Шепелев // Компьютерная оптика. – 2015.– Т. 39, № 4. – С. 572-581. – DOI: 10.18287/0134-2452-2015-39-4-572-581.

3. Nikolaidis, A. Low overhead reversible data hiding for color JPEG images / A. Nikolaidis // Multimedia Tools and Applications. – 2016. – Vol. 75(4). – P. 1869-1881. – DOI: 10.1007/s11042-014-2377-4.

4. Митекин, В.А. Алгоритмы встраивания информации на основе QIM, стойкие к статистической атаке / В.А. Митекин, В.А. Федосеев // Компьютерная оптика. – 2018. – Т. 42, № 1. – С. 118-127. – DOI: 10.18287/2412-6179-2018-42-1-118-127.

5. Евсютин О.О., Мельман (Кокурина) А.С., Мещеряков Р.В., Исхакова А.О. A new approach to reducing the distortion of the digital image natural model in the DCT domain when embedding information according to the QIM method // Proceedings of the 29th International Conference on Computer Graphics and Vision (GraphiCon 2019; Bryansk; Russian). Aachen: CEUR-WS, 2019. С. 268-272.

6. Chang, C.-C. A steganographic method based upon JPEG and quantization table modification / C.-C. Chang, T.-S. Chen, L.-Z. Chung // Information Sciences. – 2002. – Vol. 141(1-2). – P. 123-138. – DOI: 10.1016/S0020-0255(01)00194-3.

7. Мельман Анна Сергеевна, Петров Павел Олегович, Шелупанов Александр Александрович, Аристов Анатолий Владимирович, Похолков Юрий Петрович ВСТРАИВАНИЕ ИНФОРМАЦИИ В JPEG-ИЗОБРАЖЕНИЯ С МАСКИРОВКОЙ ИСКАЖЕНИЙ В ЧАСТОТНОЙ ОБЛАСТИ // Доклады ТУСУР. 2020. №4. URL: https://cyberleninka.ru/article/n/vstraivanie-informatsii-

v-jpeg-izobrazheniya-s-maskirovkoy-iskazheniy-v-chastotnoy-oblasti (дата обращения: 28.01.2024).

8. GIM_srw_fifth_term. URL: https://github.com/Krollbotid/GIM_srw_fifth_term (дата обращения: 29.01.2024).

ПРИЛОЖЕНИЕ А

Объявления функций, используемых встраивающей (coder), извлекающей (decoder) и реализующей ГА (evolver) программами, и классов используемых для реализации ГА.

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <jpeglib.h>
#include <unordered_map>
#include <vector>
#include <string>
#include <bitset>
#include <algorithm>
#include <cmath>
#include <fstream>

#define MAXPOP 100
#define DESIRED_FITNESS 100

void to_zigzag(const JCOEFPTR in);
void from_zigzag(const JCOEFPTR in);
JCOEF find_quant_step(const JCOEFPTR arr, const size_t begin, const size_t
end);

namespace evolution {
   struct individ {
      static const int genLen = 10;
      int gene[genLen];
      double fitness;
      individ() : gene{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, fitness{0} {};
```

```
        individ(const int (&ch1)[genLen]);
    };

    bool operator== (const individ &id1, const individ &id2);
    individ breed(const individ &p1, const individ &p2);
    bool IndividComparator(const individ& a, const individ& b);



    class Evolution {
    private:
        individ population[MAXPOP];
        inline static const char genStorFilename[] = "geneticStorage.txt";
    public:
        int popSave();
        int popLoad();
        int CreateFitnesses(const std::string &filename);
        int CreateNewPopulation();
        individ getGene(const int &i) { return population[i]; }
        int evolve(const std::string &filename);
    };
}
```

ПРИЛОЖЕНИЕ Б

Реализация объявленных в приложении А функций и классов.

```
#include "common.h"
// Tables below exist in jutils.c, but I dont want to waste my time looking for way to include them - copy-paste is faster
const int my_jpeg_zigzag_order[DCTSIZE2] = {
   0,  1,  5,  6, 14, 15, 27, 28,
   2,  4,  7, 13, 16, 26, 29, 42,
   3,  8, 12, 17, 25, 30, 41, 43,
   9, 11, 18, 24, 31, 40, 44, 53,
  10, 19, 23, 32, 39, 45, 52, 54,
  20, 22, 33, 38, 46, 51, 55, 60,
  21, 34, 37, 47, 50, 56, 59, 61,
  35, 36, 48, 49, 57, 58, 62, 63
};

const int my_jpeg_natural_order[DCTSIZE2] = {
   0,  1,  8, 16,  9,  2,  3, 10,
  17, 24, 32, 25, 18, 11,  4,  5,
  12, 19, 26, 33, 40, 48, 41, 34,
  27, 20, 13,  6,  7, 14, 21, 28,
  35, 42, 49, 56, 57, 50, 43, 36,
  29, 22, 15, 23, 30, 37, 44, 51,
  58, 59, 52, 45, 38, 31, 39, 46,
  53, 60, 61, 54, 47, 55, 62, 63
};

void to_zigzag(const JCOEFPTR in)
{
   JBLOCK buf;
```

```
    for (int i = 0; i < DCTSIZE2; ++i) {
        buf[my_jpeg_zigzag_order[i]] = in[i];
    }
    for (int i = 0; i < DCTSIZE2; ++i) {
        in[i] = buf[i];
    }
    return;
}

void from_zigzag(const JCOEFPTR in)
{
    JBLOCK buf;
    for (int i = 0; i < DCTSIZE2; ++i) {
        buf[my_jpeg_natural_order[i]] = in[i];
    }
    for (int i = 0; i < DCTSIZE2; ++i) {
        in[i] = buf[i];
    }
    return;
}

JCOEF find_quant_step(const JCOEFPTR arr, const size_t begin, const size_t end)
// [begin, end)
{
    std::unordered_map<JCOEF, int> frequencyMap;

    // Count the frequency of each element in the array
    for (int i = begin; i < end; ++i) {
        frequencyMap[arr[i]]++;
    }
```

```
    // Find the minimum frequency value
    int minFrequency = std::min_element(frequencyMap.begin(),
frequencyMap.end(),
        [](const auto& a, const auto& b) {
            return a.second < b.second;
        })->second;

    // Find all elements with the minimum frequency value
    std::vector<JCOEF> leastFrequentElements;
    for (const auto& pair : frequencyMap) {
        if (pair.second == minFrequency) {
            leastFrequentElements.push_back(pair.first);
        }
    }
    std::transform(leastFrequentElements.begin(), leastFrequentElements.end(),
leastFrequentElements.begin(), [](JCOEF n) { return std::abs(n); });
    JCOEF ans = *std::min_element(leastFrequentElements.begin(),
leastFrequentElements.end());
    if (ans < 3)
        ans = 3;
    if (ans > 20)
        ans = 20;
    return ans;
}

namespace evolution {
    individ::individ(const int (&ch1)[genLen])
    {
        for (int i = 0; i < genLen; ++i) {
```

```
        gene[i] = ch1[i];
    }
}

bool operator==(const individ &id1, const individ &id2)
{
    for (int i = 0; i < id1.genLen; ++i) {
        if (id1.gene[i] != id2.gene[i])
            return false;
    }
    return true;
}

individ breed(const individ &p1, const individ &p2)
{
    int crossover = rand() % p1.genLen; // Create the crossover point (not first).
        int first = rand() % 100;// Which parent comes first?

        individ child = p1; // Child is all first parent initially.

        int initial = 0, final = p1.genLen; // The crossover boundaries.
        if (first < 50)
       initial = crossover;
        else
       final = crossover + 1;

        for(int i = initial; i < final; ++i) { // Crossover!
             child.gene[i] = p2.gene[i];
        }
    for (int i = 0; i < child.genLen; ++i) {
```

```
            if (rand() % 100 < 10) {
          child.gene[i] = rand() % 2;
        }
    }

        return child; // Return the kid...
}

int Evolution::popSave()
{
    std::ofstream myfile;
    myfile.open(genStorFilename);
    for (int i = 0; i < MAXPOP; ++i) {
        for (int j = 0; j < population[i].genLen; ++j)
            myfile << population[i].gene[j] << " ";
        myfile << std::endl;
    }
    return 0;
}

int Evolution::popLoad()
{
    std::ifstream myfile;
    myfile.open(genStorFilename);
    for (int i = 0; i < MAXPOP; ++i) {
        for (int j = 0; j < population[i].genLen; ++j)
            myfile >> population[i].gene[j];
    }
    return 0;
}
```

```
int Evolution::CreateFitnesses(const std::string &filename)
{
    std::string baseCom("./coder ");
    std::string comEnd(".jpg"), comEnd2(".csv");
    system((baseCom + filename).c_str());
    system("python PSNRCalc.py");
    std::ifstream myfile;
    myfile.open((filename + comEnd2).c_str());
    int fitness = -1;
    for (int i = 0; i < MAXPOP; ++i) {
        myfile >> population[i].fitness;
        if (population[i].fitness > fitness) {
            fitness = population[i].fitness;
            if (fitness >= DESIRED_FITNESS)
                return i;
        }
    }
    return -1;
}

bool IndividComparator(const individ& a, const individ& b)
{
    // Compare based on Fitness value in descending order
    return a.fitness > b.fitness;
}

int Evolution::CreateNewPopulation()
{
    std::vector<individ> arr(std::begin(population), std::end(population));
```

```
    // Use std::partial_sort to get the n largest elements
    std::partial_sort(arr.begin(), arr.begin() + MAXPOP / 10, arr.end(),
IndividComparator);

          for(int i = 0; i < MAXPOP / 10; ++i) {
               for (int j = i + 1; j < MAXPOP / 10; ++j) {
           int sum = 0;
           while (sum != 6) {
              sum = 0;
              population[i * 10 + j] = breed(arr[i], arr[j]);
              for (int k = 0; k < population[0].genLen; ++k) {
                 sum += population[i * 10 + j].gene[k];
              }
           }
        }
         }

    for (int i = MAXPOP - MAXPOP / 10; i < MAXPOP; ++i) {
       population[i] = arr[i % 10];
    }

    return 0;
}

int Evolution::evolve(const std::string &filename)
{

     // Generate initial population.
     srand((unsigned)time(NULL));
```

```
        for (int i = 0; i < MAXPOP; ++i) { // Fill the population with numbers between
                for (int j = 0; j < population[i].genLen; ++j) {// 0 and the result.
                        population[i].gene[j] = rand() % 2;
                }
        }

      int index = CreateFitnesses(filename);
      if (index >= 0) {
                return index;
        }

        int iterations = 0; // Keep record of the iterations.
      char del[50] = {'/'};
        while (index < 0 && iterations < 50) { // Repeat until solution found and until 50 iterations.
          std::cout << del << " " << iterations << std::endl;
                CreateNewPopulation();
          popSave();
          index = CreateFitnesses(filename);
          if (index >= 0) {
                  return index;
            }
                ++iterations;
        }
        return -1;
    }
}
```

ПРИЛОЖЕНИЕ В

Реализация встраивающей программы.

```
#include "common.h"

void insert_by_qim(const JCOEFPTR block, const size_t len, size_t
*bits_not_encoded, const std::string msg, const evolution::individ &ind) {
    JCOEF q = find_quant_step(block, 1, DCTSIZE2 - len);
    for (int i = DCTSIZE2 - len; i < DCTSIZE2; ++i) {
        if (!(ind.gene[i + len - DCTSIZE2]))
            continue;
        char c = msg[msg.size() - *bits_not_encoded] - '0';
        JCOEF sec_bit = (JCOEF) c;
        //std::cout << i << " " << std::endl;
        //std::cout << block[i] << " " << q << " " << sec_bit << std::endl;
        block[i] = q * (int) floor( (float) block[i] / q) + q * sec_bit / 2;
        //std::cout << block[i] << " " << q << " " << sec_bit << std::endl;
        *bits_not_encoded -= 1;
        if (*bits_not_encoded == 0)
            break;
    }
    return;
}

int write_jpeg_file(std::string outname, jpeg_decompress_struct in_cinfo,
jvirt_barray_ptr *coeffs_array)
{

    struct jpeg_compress_struct cinfo;
    struct jpeg_error_mgr jerr;
    FILE * infile;
```

```
    if ((infile = fopen(outname.c_str(), "wb")) == NULL) {
        fprintf(stderr, "can't open %s\n", outname.c_str());
        return 1;
    }

    cinfo.err = jpeg_std_error(&jerr);
    jpeg_create_compress(&cinfo);
    jpeg_stdio_dest(&cinfo, infile);

    j_compress_ptr cinfo_ptr = &cinfo;
    jpeg_copy_critical_parameters((j_decompress_ptr)&in_cinfo, cinfo_ptr);
    jpeg_write_coefficients(cinfo_ptr, coeffs_array);

    jpeg_finish_compress( &cinfo );
    jpeg_destroy_compress( &cinfo );
    fclose( infile );

    return 0;
}

int readnChange_jpeg_file(const std::string filename, const std::string outname,
const size_t len, size_t *bits_not_encoded, const std::string msg, const
evolution::individ &ind)
{
    // setup for decompressing
    struct jpeg_decompress_struct cinfo;
    struct jpeg_error_mgr jerr;
    FILE * infile;
    if ((infile = fopen(filename.c_str(), "rb")) == NULL) {
```

```
    fprintf(stderr, "can't open %s\n", filename.c_str());
    return 1;
}
cinfo.err = jpeg_std_error(&jerr);
jpeg_create_decompress(&cinfo);
jpeg_stdio_src(&cinfo, infile);
jpeg_read_header(&cinfo, TRUE);

jvirt_barray_ptr *coeffs_array = jpeg_read_coefficients(&cinfo);
JBLOCKARRAY buffer_one;
/*
   Note about types:
   JCOEPTR is defined as JCOEF*
   JBLOCK is defined as JCOEF[DCTSIZE2]
   So in general they're equal
*/
JCOEFPTR blockptr_one;
jpeg_component_info* compptr_one;
    /*
//change one dct:
int ci = 0; // between 0 and number of image component
int by = 0; // between 0 and compptr_one->height_in_blocks
int bx = 0; // between 0 and compptr_one->width_in_blocks
int bi = 0; // between 0 and 64 (8x8)
    */
    for (int color_comp = 0; color_comp < 3; ++color_comp) { // ci
    compptr_one = cinfo.comp_info + color_comp;
            for (int i = 0; i < compptr_one->height_in_blocks; i++) { //by
```

```
                buffer_one = (cinfo.mem-
>access_virt_barray)((j_common_ptr)&cinfo, coeffs_array[color_comp], i,
(JDIMENSION)1, FALSE);
                for (int j = 0; j < compptr_one->width_in_blocks; ++j) { //bx
                        blockptr_one = buffer_one[0][j]; // YES, left index must be 0
otherwise it gets SIGSEGV after half of rows. Idk why.
                                to_zigzag(blockptr_one);
            insert_by_qim(blockptr_one, len, bits_not_encoded, msg, ind);
            from_zigzag(blockptr_one);
            if (!(*bits_not_encoded)) {
               goto out_of_cycles;
            }
               }
        }
        }
   out_of_cycles:

   write_jpeg_file(outname, cinfo, coeffs_array);

   jpeg_finish_decompress( &cinfo );
   jpeg_destroy_decompress( &cinfo );
   fclose( infile );

   return 0;
}

int main(int argc, char* argv[])
{
        if (argc != 2) {
```

```
        fprintf(stderr, "Usage: %s <input_file_base>\ne.g. %s examplein\n", argv[0],
argv[0]);
        exit(EXIT_FAILURE);
    }
    std::string infilename("./source/"), outfilename("./encoded/");
    infilename += argv[1];
    outfilename += argv[1];

    // secret message setup
    std::ifstream file("info.txt");
    std::string msg;

    if (file.is_open()) {
        std::string line;
        while (file >> line) {
            msg += line + " ";
        }
        file.close();
    } else {
        std::cerr << "Error: Unable to open the file." << std::endl;
    }

    std::string bmsg;
    for (char c : msg) {
        std::bitset<8> binary(c);
        bmsg += binary.to_string();
    }
    size_t bits_not_encoded;

    size_t lens[] = {10}; // amount of coefficients for inserting
```

```
    /* uses for training
    evolution::Evolution model;
    model.popLoad(); */
    evolution::individ bestind;
    int bestgene[] = {1, 1, 0, 0, 1, 0, 1, 1, 0, 1};
    for (int i = 0; i < 10; ++i) {
        bestind.gene[i] = bestgene[i];
    }
    for (int k = MAXPOP; k < MAXPOP + 1; ++k) {
        // Try reading and changing a jpeg
        bits_not_encoded = bmsg.size();
        if (readnChange_jpeg_file(infilename + std::string(".jpg"), outfilename +
std::to_string(k) + std::string(".jpg"), lens[0], &bits_not_encoded, bmsg, bestind)
== 0)
        {
            //std::cout << bmsg << std::endl
            std::cout << "It's Okay... Gene #" << k << " " << bits_not_encoded << "bits
left not encoded." << std::endl;
        }
        else return 1;
    }
    return 0;
}
```

ПРИЛОЖЕНИЕ Г

Реализация извлекающей программы.

```
#include "common.h"

void extract_by_qim(const JCOEFPTR block, const size_t len, size_t
*bits_decoded, std::string *msg, const evolution::individ &ind) {
    JCOEF q = find_quant_step(block, 1, DCTSIZE2 - len);
    for (int i = DCTSIZE2 - len; i < DCTSIZE2; ++i) {
        if (!(ind.gene[i + len - DCTSIZE2]))
            continue;
        int b = block[i] - q * (int) floor( (float) block[i] / q); // intermediate term
        if (abs(b) < abs(b - q / 2)) {
            *msg += '0';
        }
        else {
            *msg += '1';
        }
        //std::cout << i << " " << std::endl;
        //std::cout << block[i] << " " << q << " " << msg->back() << std::endl;
        *bits_decoded += 1;
        if (*bits_decoded == 48000)
            break;
    }
    return;
}

int readnChange_jpeg_file(const std::string filename, const size_t len, size_t
*bits_decoded, std::string *msg, const evolution::individ &ind)
{
    // setup for decompressing
```

```
struct jpeg_decompress_struct cinfo;
struct jpeg_error_mgr jerr;
FILE * infile;
if ((infile = fopen(filename.c_str(), "rb")) == NULL) {
    fprintf(stderr, "can't open %s\n", filename.c_str());
    return 1;
}
cinfo.err = jpeg_std_error(&jerr);
jpeg_create_decompress(&cinfo);
jpeg_stdio_src(&cinfo, infile);
jpeg_read_header(&cinfo, TRUE);

jvirt_barray_ptr *coeffs_array = jpeg_read_coefficients(&cinfo);
JBLOCKARRAY buffer_one;
/*
   Note about types:
   JCOEPTR is defined as JCOEF*
   JBLOCK is defined as JCOEF[DCTSIZE2]
   So in general they're equal
*/
JCOEFPTR blockptr_one;
jpeg_component_info* compptr_one;
    /*
//change one dct:
int ci = 0; // between 0 and number of image component
int by = 0; // between 0 and compptr_one->height_in_blocks
int bx = 0; // between 0 and compptr_one->width_in_blocks
int bi = 0; // between 0 and 64 (8x8)
    */
    for (int color_comp = 0; color_comp < 3; ++color_comp) { // ci
```

```
        compptr_one = cinfo.comp_info + color_comp;
                for (int i = 0; i < compptr_one->height_in_blocks; i++) { //by
                buffer_one = (cinfo.mem-
>access_virt_barray)((j_common_ptr)&cinfo, coeffs_array[color_comp], i,
(JDIMENSION)1, FALSE);
                for (int j = 0; j < compptr_one->width_in_blocks; ++j) { //bx
                        blockptr_one = buffer_one[0][j]; // YES, left index must be 0
otherwise it gets SIGSEGV after half of rows. Idk why.
            /*std::cout << "block " << color_comp << " " << i << " " << j <<
std::endl;
            for (int l = 0; l < DCTSIZE2; ++l)
               std::cout << blockptr_one[l] << " ";*/
            std::cout << std::endl;
                                to_zigzag(blockptr_one);
            extract_by_qim(blockptr_one, len, bits_decoded, msg, ind);
            from_zigzag(blockptr_one);
            if (*bits_decoded >= 48000) {
               goto out_of_cycles;
            }
               }
        }
        }
   out_of_cycles:

   jpeg_finish_decompress( &cinfo );
   jpeg_destroy_decompress( &cinfo );
   fclose( infile );

   return 0;
}
```

```
int main(int argc, char* argv[])
{
        if (argc != 2) {
      fprintf(stderr, "Usage: %s <input_file_base>\ne.g. %s examplein\n", argv[0],
argv[0]);
      exit(EXIT_FAILURE);
   }
   std::string infilename("./encoded/");
   infilename += argv[1];

   size_t lens[] = {10}; // amount of coefficients for inserting
   evolution::Evolution model;
   model.popLoad();
   for (int k = 1; k < MAXPOP; ++k) {
      // secret message setup
      std::string msg;
      std::string bmsg;
      size_t bits_decoded = 0;
      // Try reading and changing a jpeg
      if (readnChange_jpeg_file(infilename + std::to_string(k) + std::string(".jpg"),
lens[0], &bits_decoded, &bmsg, model.getGene(k)) == 0)
      {
         std::cout << "It's Okay... " << bits_decoded << "bits decoded." <<
std::endl;
         /*for (int i = 0; i < bits_decoded % 8; ++i) {
            bmsg.pop_back();
         }
         bits_decoded -= bits_decoded % 8;
         if (bits_decoded != bmsg.size() ) {
```

```
            std::cout << bits_decoded << bmsg.size() << std::endl;
        }*/
        //std::cout << bmsg.substr(0, 128) << std::endl;
        for (int i = 0; i < bits_decoded; i += 8) {
            std::bitset<8> byte(bmsg.substr(i, 8));
            char c = static_cast<char>(byte.to_ulong());
            msg += c;
        }
        std::cout << "Message:" << msg << std::endl;
      }
      else return 1;
   }
   return 0;
}
```

ПРИЛОЖЕНИЕ Д

Реализация запуска генетического алгоритма

```
#include "common.h"

int main(int argc, char* argv[])
{

   if (argc != 2) {
      fprintf(stderr, "Usage: %s <input_file_base>\ne.g. %s examplein\n", argv[0],
argv[0]);
      exit(EXIT_FAILURE);
   }
   std::string filename(argv[1]);
   evolution::Evolution model;
   int ans = model.evolve(filename);
   if (ans == -1) {
      std::cout << "No solution found." << std::endl;
   } else {
      evolution::individ gn = model.getGene(ans);
      for (int i = 0; i < gn.genLen; ++i)
         std::cout << gn.gene[i] << " ";
      std::cout << std::endl;
   }
   return 0;
}
```