VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Ataskaita

Golay C(23,12) kodavimas — praktinė užduotis

Atliko:
Aleksej Krasavcev

Darbo vadovas:
dr. Gintaras Skersys

Vilnius
2025

Turinys

Įvadas

Šioje ataskaitoje aprašoma įgyvendinta programa, kuri skaitydama 24-bitų BMP paveikslėlį suskaido duomenis į 12 bitų blokų, užkoduoja naudodama Golay C(23,12) kodą, siunčia per binarinį simetrišką kanalą (BSC), dešifruoja ir atkartoja paveikslėlį. Pridedamos naudojimo instrukcijos, programos failų aprašymai, laiko sąnaudos, programuoti sprendimai bei eksperimentų gairės su grafiko šablonu.

Santrauka

- Projekto tikslas: realizuoti 12→23 bitų Golay kodo užkodavimą, kanalų simuliaciją ir klaidų taisymą realiuose duomenyse (paveikslėliuose).
- Kalba: Python 3.x
- Reikalingos bibliotekos: `Pillow`

1. Įgyvendintos užduoties dalys

Dalies pavadinimas	**Statusas**
Pagrindiniai moduliai	
Užkodavimas	Atlikta
Siuntimas kanalu	Atlikta
Dekodavimas	Atlikta
1 Scenarijus	
Vektoriaus ilgio patikrinimas	Atlikta
Užkodavimas	Atlikta
Siuntimas kanalu	Atlikta
Parodo klaidų pozicijas	Atlikta
Galimybė naudotojui pakeisti iškraipytą vektorių	Atlikta
Dekodavimas	Atlikta
2 Scenarijus	
Teksto skaidymas į vektorius	Atlikta
Vektorių užkodavimas	Atlikta
Siuntimas neužkoduotų vektorių kanalu	Atlikta
Siuntimas užkoduotų vektorių kanalu	Atlikta
Neužkoduoto teksto atstatymas	Atlikta
Užkoduoto teksto dekodavimas ir atstatymas	Atlikta
3 Scenarijus	
Paveikslėlio skaidymas į vektorius	Atlikta
Vektorių užkodavimas	Atlikta
Siuntimas neužkoduotų vektorių kanalu	Atlikta
Siuntimas užkoduotų vektorių kanalu	Atlikta
Neužkoduoto paveikslėlio atstatymas ir išsaugojimas	Atlikta
Užkoduoto paveikslėlio dekodavimas, atstatymas ir išsaugojimas	Atlikta
Dokumentacija	
Funkcijų aprašymas	Atlikta
Stambesnių kodo dalių aprašymas	Atlikta

1 lentelė. Implementuotos dalys

2. Trečiųjų šalių bibliotekos

- **Pillow** (`PIL`) — vaizdų nuskaitymui ir išsaugojimui (24-bit BMP atidarymas ir rašymas). Instaliacija: `pip install pillow`.
- Standartinės Python bibliotekos:
  - `concurrent.futures` - paraleliam užkodavimo, kanalo ir dekodavimo operacijų vykdymui naudojant `ProcessPoolExecutor`. Leidžia paskirstyti CPU-intensive užduotis keliems procesams ir paspartinti vykdymą daugiabrandžioiuose procesoriuose.
  - `threading` - sinchronizacijai tarp gijų naudojant `Lock()`, siekiant apsaugoti modulinį atsitiktinių skaičių generatorių (`_module_rng`) nuo lenktyniavimo situacijų (race conditions) kanalo simuliacijoje.
  - `multiprocessing` - procesų valdymui Windows aplinkoje. Naudojama `freeze_support()` funkcija užtikrina tinkamą programos veikimą Windows sistemose.
  - `itertools` - efektyviam iteravimui per duomenų struktūras bei chunk'ų formavimui paraleliam apdorojimui.
  - `os` - sisteminėms operacijoms: `os.urandom()` naudojamas kriptografiškai saugiam atsitiktinių skaičių generatoriaus seed'ui sukurti, `os.cpu_count()` - procesų skaičiaus nustatymui pagal CPU branduolių kiekį.
  - `random` - atsitiktinių skaičių generavimui kanalo simuliacijoje. Naudojamas `random.Random()` objektas su seed'u iš `os.urandom()`, kad būtų išvengta seed'o perkartojimo tarp skirtingų procesų.
  - `time` - vykdymo laiko matavimui naudojant `time.perf_counter()`, kuris užfiksuoja kiekvienos pipeline fazės trukmę (bytes blocks konvertavimas, užkodavimas, kanalas, dekodavimas).
  - `functools` - rezultatų kešavimui naudojant `@lru_cache` dekoratorių. Matricos (G, H, B) maskų konvertavimas į integer formato kaukes atliekamas tik kartą ir išsaugomas atmintyje vėlesniam naudojimui.

3. Laiko sąnaudos

- Literatūros skaitymui ir kodo veikimo aiškinimuisi: 2.5 h
- Projektavimui: 1 h
- Programavimui, klaidų ieškojimui ir taisymui: 13.5 h
- Ataskaitos ruošimui: 8 h

Viso (sąlyginai): apie 25 h.

4. Kaip paleisti programą

Programa galima paleisti 2 būdais - per vykdomąjį failą `main.exe` arba per `python` aplinką. Minimalūs žingsniai programos paleidimui su `python` Windows aplinkoje:

1. Įsidiekite Python 3.13.3 arba naujesnę versiją.
2. Įsidiekite priklausomybes:

```
pip install pillow
pip install pyinstaller  # jei norite sukompiliuoti .exe fail
```

3. Paleidimas iš komandų eilutės (PowerShell):

```
python .\main.py
```

Programa paleidžia meniu, pasirinkite:

- `1` - interaktyvus 12 bitų vektorius su galimybe pakeisti jau iškraipytą vektorių
- `2` - tekstinis įvedimas (pavienis string)
- `3` - pasirinkti (parašyti kelią) 24-bit BMP paveikslėlį
- `4` - išeiti

4. Jei norite sukurti vieną vykdomąjį failą: (parinktinai)

```
pyinstaller --onefile main.py
```

Pastabos apie parametrus (interaktyviai per meniu):

- Klaidos tikimybė `p`: iš intervalo [0,1] (pvz. 0.01 = 1% bito iškraipymo tikimybė).
- Vaizdo kelias: įveskite pilną arba santykinį kelią iki 24-bit BMP failo.

5. Programos failų aprašymas

`main.py`. — interaktyvus meniu: atidarymas, bytes → 12-bit blokai, bitmapų kūrimas, iškviečiami `functions.py` helper'ai. Čia matomi trys scenarijai (1: vektorius, 2: tekstas, 3: paveikslėlis). Taip pat čia registruojami laiko skaitikliai ir įrašomi rekonstruoti failai: `*_reconstructed.bmp` ir `*_reconstructed_encoded.bmp`.

`functions.py`. — visos žemų lygių funkcijos:

- Golay matricos `G()`, `H()`, `B()` ir jų mask'ų konvertavimas į integer kaukes (`G_masks()`, `H_masks()`, `B_masks()`).
- `encode_int` ir `decode_int`, `IMLD_int`, taip pat vidiniai pagalbiniai `_syndrome_w12/24_int`, `_add_w24_int` funkcijos.
- Kanalų modeliai: `canal` (bitų sąrašui, `list[int]`), `canal_int12`, `canal_int23`. Moduliniam RNG naudojama `os.urandom` seeda ir `Lock()` apsauga siekiant saugumo per procesus.
- Blokų pakavimas: `bytes_to_12bit_ints` ir `blocks_ints_to_bytes`, bei aukšto lygio paralelūs wrapperiai: `bytes_to_blocks`, `encode_blocks`, `canal_blocks12/23`, `decode_blocks`, `blocks_to_bytes`.
- Išsaugojimas: `save_to_file` funkcija (naudoja `Pillow.Image.frombytes` ir `save`).

6. Vartotojo sąsaja ir naudojimo pavyzdžiai

Programa turi tekstinį meniu (komandų eilutė).

- Pavyzdys darbiniam scenarijui (1 — vartotojo vektorius):

```
Golay (C23) Code Implementation
------------------------------
Possible scenarios to test implementation:
1. Enter 12-bit vector
2. Enter text
3. Chose (write path) image file to encode/decode
4. Exit
------------------------------
Enter your choice (1-4): 1
Enter a 12-bit binary vector (e.g., 101010101010): 111000111000
Enter error probability (e.g., 0.01 for 1%): 0.01
```

Išvestys: originalus vektorius, užkoduotas vektorius, užkoduotas vektorius išsiųstas per kanalą, klaidų vektorius, klaidų kiekis.

```
Do you want to change the noisy vector? (y/n): n
```

Išvestys: originalus vektorius, dekoduotas vektorius, klaidų vektorius.

- Pavyzdys darbiniam scenarijui (2 — tekstas):

```
Golay (C23) Code Implementation
------------------------------
Possible scenarios to test implementation:
1. Enter 12-bit vector
2. Enter text
3. Chose (write path) image file to encode/decode
4. Exit
------------------------------
Enter your choice (1-4): 2
Enter text to encode: hello, world!
Enter error probability (e.g., 0.01 for 1%): 0.01
```

Išvestys: originalus tekstas, atstatytas neužkoduotas tekstas išsiųstas per kanalą, atstatytas užkoduotas tekstas išsiųstas per kanalą, užkoduotų vektorių kiekis, klaidų kiekis.

- Pavyzdys darbiniam scenarijui (3 — paveikslėlis):

```
Golay (C23) Code Implementation
------------------------------
Possible scenarios to test implementation:
1. Enter 12-bit vector
2. Enter text
3. Chose (write path) image file to encode/decode
4. Exit
------------------------------
Enter your choice (1-4): 3
Enter the path to the 24-bit BMP image file: test.bmp
Enter error probability (e.g., 0.01 for 1%): 0.01
```

Išvestys: laiko matavimai, sugeneruoti failai pavadinimu `test_reconstructed.bmp` ir `test_reconstructed_encoded.bmp`, keletas statistinių verčių (bendras bitų skaičius, vektorių skaičius, užkoduotų vektorių skaičius, kiek bitų buvo apversta kanale, klaidų kiekis po dekodavimo ir paveikslėlio atstatymo).

7. Programiniai sprendimai

- Duomenų suskaidymas: baitai skaitomi MSB-first, susikaupia į bitinį akumuliatorių ir iš jo traukiami 12-bit blokai. Jei paskutinis blokas pilnai neužsipildo 12 bitų, jis užpildomas nuliais (LSB pusėje), kad būtų 12 bitų ilgis.
- Kodavimas: kiekvienas 12-bit blokas konvertuojamas į 23-bit kodinį žodį su `encode_int` naudojant išankstines `G_masks()`. Prie dekodavimo atliekama 24-to bitų pariteto papildymas.
- Dekodavimas: `IMLD_int` įgyvendina B-matrix paiešką, kad išspręstų iki 3 klaidų per 24-bit žodį.
- Kanalas: BSC modelis taikomas kiekvienam bitui nepriklausomai su tikimybe `p`. Kanalo funkcijos yra `canal` (operacijoms su int'ų sąrašu: `list[int]`), `canal_int12` ir `canal_int23` (operacijos su integeriais). Moduliniam RNG naudojamas `random.Random(os.urandom())` per-proceso seed, kad būtų išvengta deterministinių kartojimų tarp procesų; prieiga apsaugota `Lock()`.
- Paralelizacija: CPU-bound užduotys (kodavimas, kanalas, dekodavimas, pack/unpack) atliktos per `ProcessPoolExecutor`. Siekiant išvengti tarpo užpildymo sukeltų vizualių artefaktų (pvz., horizontalių juostų), įvesta chunk'ų dalijimo logika: bytes→blocks dalijami pagal 3-baitų ribas (24 bitai = 2 12 bitų blokai), o blocks→bytes dalijami pagal 2-blokų ribas, taip išvengiama tarpo vidinių užpildymų.
- Windows multiprocessing: projektas naudoja `multiprocessing.freeze_support()` ir picklable top-level funkcijas, kad būtų suderinamas su Windows proceso paleidimo režimu.

8. Atliktų eksperimentų aprašymas ir gairės

Šiame darbe atlikti eksperimentai parodo Golay C(23,12) kodo poveikį bitų klaidų lygiui esant skirtingoms kanalo klaidos tikimybėms. Eksperimentai atlikti lokaliai naudojant 1 MB atsitiktinius duomenis kaip testinį įvesties srautą; kiekvienai kanalo tikimybei p paleista po 10 nepriklausomų bandymų (”runs”) ir gauti vidurkiai užrašyti.

Produkuoti failai ir parametrai:

- `dummy_1MB.bin` — sugeneruotas 1 048 576 baitų atsitiktinis failas (jeigu nebuvo rastas, sukurtas automatiniu būdu).
- `experiments_results.csv` — vienas įrašas kiekvienam p ir kiekvienam bandymui (stulpeliai: ‘p‘, ‘run‘, klaidų skaičiai, santykiai, ‘timings‘ ir kt.).
- `experiments_results_avg.csv` — apibendrinti vidurkiai per 10 bandymų kiekvienai p (naudota grafike).
- Naudoti p reikšmių rinkinys: 0.0001, 0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5; taip pat atliktas patikros paleidimas su p=0.
- `max_workers` pradinis nustatymas eksperimentui: 8 (galima keisti norint palyginti greitį su mažesniu/ didesniu procesu skaičiumi).
- `num_runs` = 10 (kiekvienai p pakartojimų skaičius).

Metodika: duomenys sugrupuojami į 12-bit blokus, užkoduojami su Golay C(23,12), kodiniai žodžiai siunčiami per kanalą su bitų apvertimo tikimybe p, sugeneruotas triukšmas taikomas tiek neužkoduotiems 12-bit blokams, tiek užkoduotiems 23-bit žodžiams. Užkoduotai grupei atliekamas dekodavimas (`IMLD_int`), po to rezultatai virsta baitais ir lyginami su originalu. Kiekviename paleidime fiksuojamos laiko atkarpos (`bytes_to_blocks, encode, channel, decode, blocks_to_bytes`) ir bitų klaidų skaičiai prieš ir po dekodavimo.

Trumpa rezultatų santrauka (pagal `experiments_results_avg.csv`):

- $p = 0$ — rekonstrukcija be klaidų (lyginant baitais), t. y. tiek neužkoduota, tiek užkoduota ir atstatyta versija sutampa su originalu.
- Mažos tikimybės ($p \leq 0.001$) — neužkoduotas srautas turi matomą bitų klaidų koeficientą (BER), tačiau Golay kodas beveik eliminuoja klaidas (vidutinės reikšmės artimos nuliui).
- Vidutinės tikimybės (pvz., $p = 0.01$–0.05) — Golay ženkliai sumažina klaidų skaičių, bet dalis klaidų gali išlikti; tai matoma `mean_errs_coded` stulpelyje `experiments_results_avg.csv`.

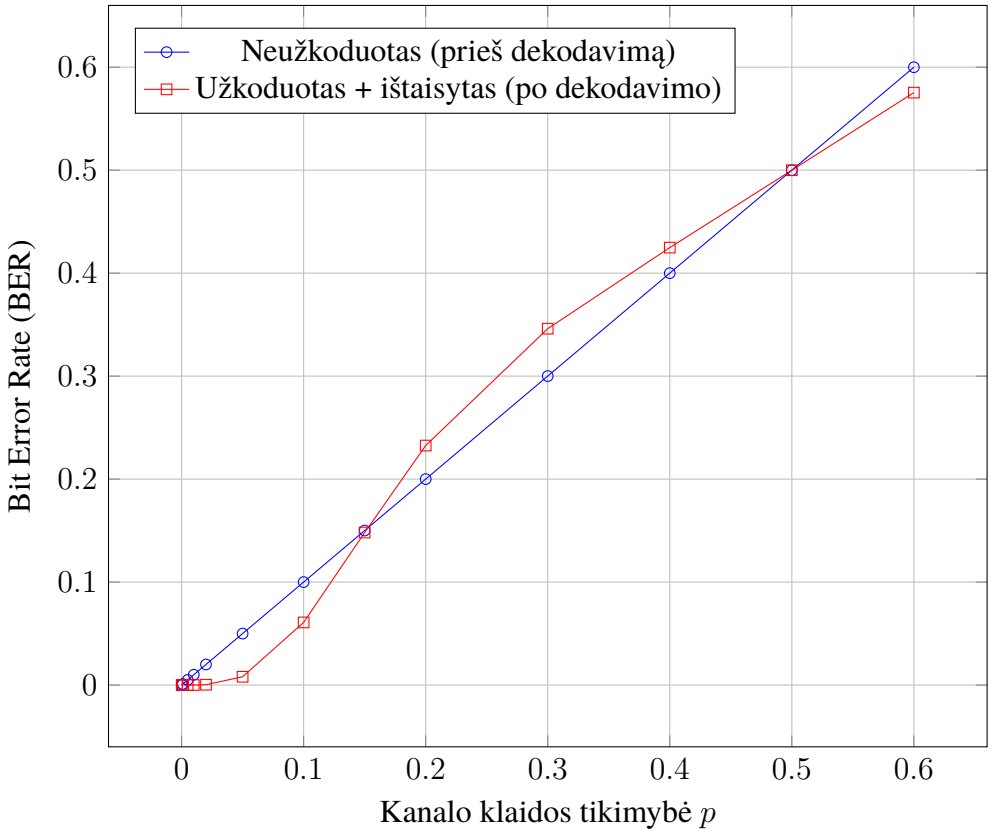
- Didelės tikimybės ($\texttt{p} \geq 0.2$) — kanalas sugeneruoja daug klaidų, Golay kodas nebegali jas pilnai ištaisyti (klaidų skaičius po dekodavimo artėja prie neužkoduoto lygio arba jį viršija priklausomai nuo konfigūracijos).

1 ir 2 grafikai nubrėžia vidutinius BER prieš ir po dekodavimo kaip funkcijas $p(x)$ ir $p(log_{10}x)$. 3 grafikas nubrėžia kai kurių operacijų trukmių priklausomybes nuo klaidų tikimybės p kaip funkcijas $p(log_{10}x)$.

Visas eksperimentų įsigijimo kodas yra faile `experiments.py`; rezultatai išsaugomi CSV formatu projekto šaknyje.

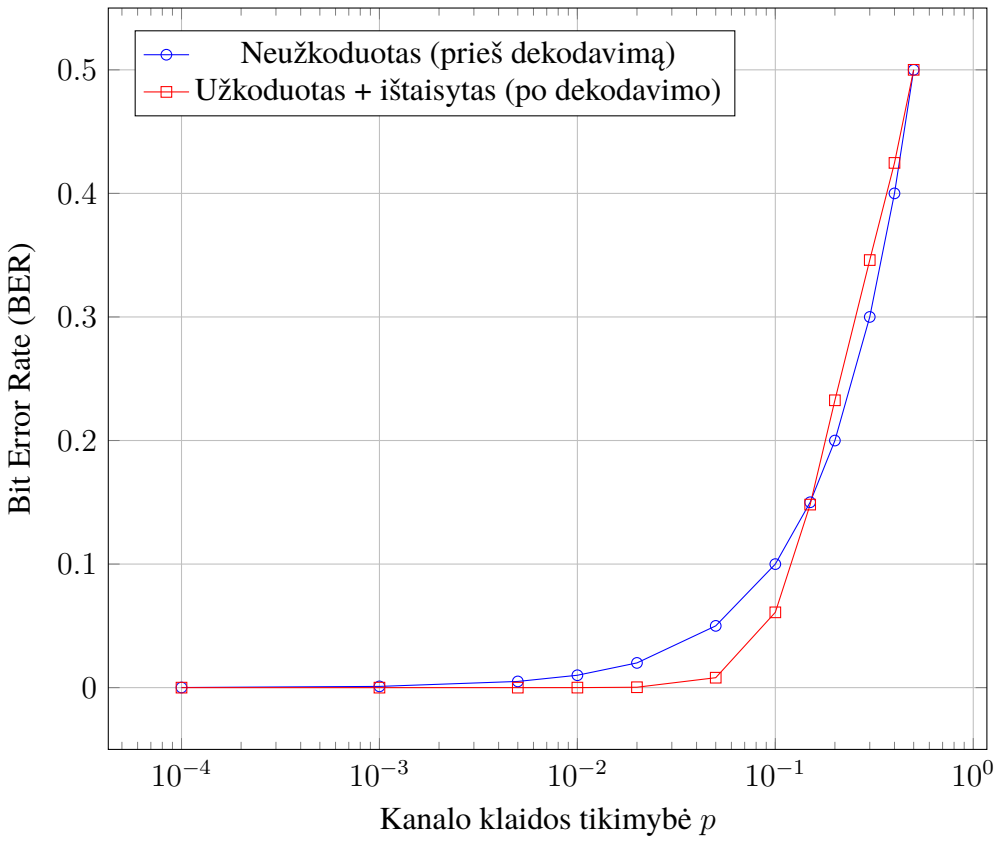
8.1. Eksperimento rezultatai

Toliau pateikti eksperimento rezultatai.



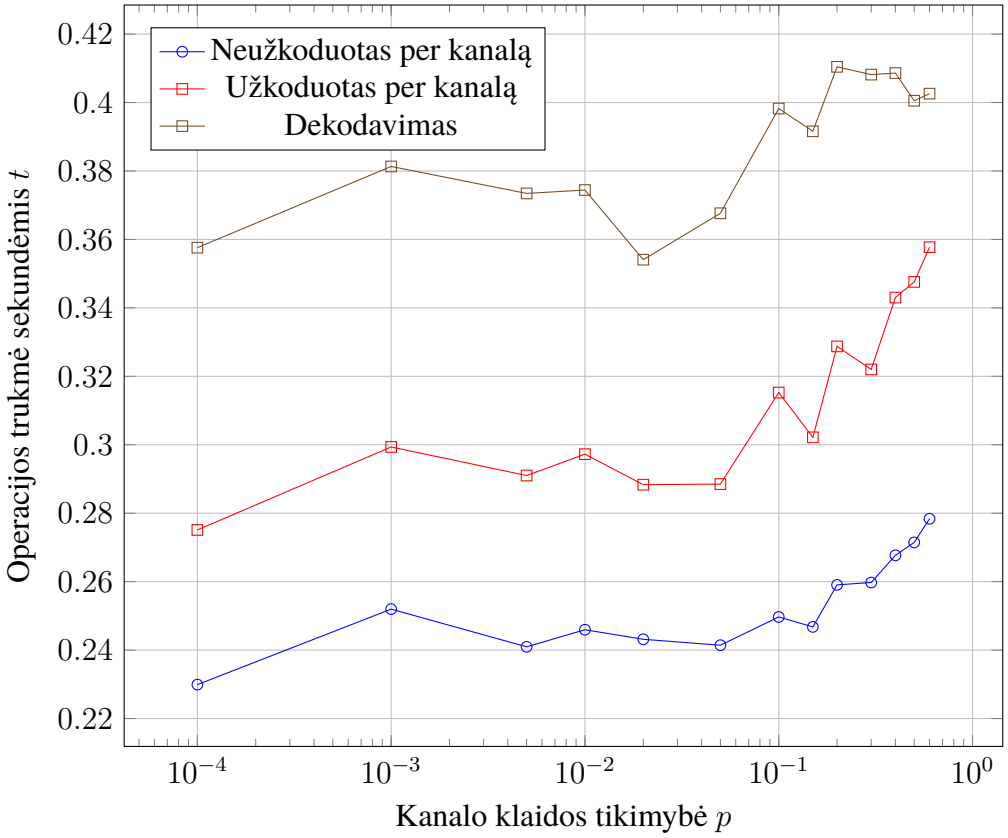
1 pav. BER priklausomai nuo kanalo klaidos tikimybės — linijinis grafikas.

1 pav. parodytos dvi kreivės: „Neužkoduotas (prieš dekodavimą)" — klaidų santykis tiesiogiai po kanalo, ir „Užkoduotas + ištaisytas (po dekodavimo)" — klaidų santykis po dekodavimo. Duomenys yra vidurkiai iš `num_runs`=10 bandymų su 1 MB atsitiktiniais baitais; tai parodo Golay kodo efektyvumą mažoms klaidų tikimybėms.



2 pav. BER priklausomai nuo kanalo klaidos tikimybės — logaritminis grafikas.

2 pav. pateikiamas tas pats duomenų rinkinys kaip linijiniame grafike, tačiau x ašis yra logaritminė. Logaritminė skalė leidžia geriau įvertinti elgseną mažose klaidų tikimybėse.



3 pav. Operacijos trukmė priklausomai nuo kanalo klaidos tikimybės — logaritminis grafikas.

Grafike 3 pav. pateiktos trys matavimo eilutės: „Neužkoduotas per kanalą” — laikas neužkoduotų blokų siuntimui per kanalą; „Užkoduotas per kanalą” — bendras užkodavimo ir siuntimo laikas; „Dekodavimas” — atskiras dekodavimo etapas. Matavimai yra vidurkiai iš `num_runs`=10 paleidimų su 1 MB duomenimis, naudojant `max_workers`=8 pagal numatytuosius eksperimentų nustatymus. Pastebima, kad užkodavimas ir dekodavimas didina bendrą apdorojimo laiką, o dekodavimo trukmė linkusi didėti su didesne klaidų tikimybe `p` (dėl dažnesnių klaidų taisymo operacijų).

Priedai

- Projekto failai:
  - `main.py`,
  - `functions.py`,
  - `experiments.py`,
  - `README.MD`.
  - `main.exe`
- Eksperimento rezultatai:
  - `experiments_results.csv`,
  - `experiments_results_avg.csv`.