ПЛОВДИВСКИ УНИВЕРСИТЕТ "ПАИСИЙ ХИЛЕНДАРСКИ"
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
КАТЕДРА "КОМПЮТЪРНИ ТЕХНОЛОГИИ"

МАРИЯ ВИКТОРОВА МАЛИНОВА

ИЗСЛЕДВАНЕ И ПРОВЕРКА НА ХИПОТЕЗИ ЗА КРАЙНИ ПОЧТИ-ПРЪСТЕНИ

АВТОРЕФЕРАТ

на дисертационен труд
за присъждане на образователната и научна степен "доктор"
в област на висше образование
4. Природни науки, математика и информатика
професионално направление
4.6 Информатика и компютърни науки
докторска програма Информатика

Научни ръководители:
Проф. д-р Ангел Голев
Проф. д-р Асен Рахнев

Пловдив, 2022 г.

Съдържание

Обща характеристика на дисертационния труд

Актуалност

Теорията на почти-пръстените е съвременен и развиващ се клон на абстрактната алгебра. В този труд се обръща специално внимание на теорията на почти-пръстените над крайни циклични групи. Разработването и прилагането на специализирани софтуерни инструменти може да даде тласък на изследването и получаването на нови резултати в тази област, като подпомогне генерирането, търсенето и анализирането на алгебрични обекти и формулирането на нови хипотези.

Изследванията в областта на почти-пръстените водят началото си от работата на Диксън от 1905 г., в която се доказва съществуването на „поле със само един дистрибутивен закон“, наречено почти-поле [1]. В следващите години редица автори използват почти-полета за да координатизират някои важни класове крайни геометрии, с чиято помощ са получени и първите примери на недезаргови и непаскалеви геометрии.

Сега почти-пръстените имат множество приложения в теория на групите, при изследване на полиномиални изображения и др. Тази теория продължава да се развива и допълва [4]. Планарните почти-пръстени имат и някои важни приложения в комбинаториката, по-точно - в конструирането на нови балансирани непълни блок-дизайни и частично балансирани непълни блок-дизайни. Тези дизайни пораждат нови методологии и в други области - например в статистически методи за подпомагане поставянето на медицински диагнози [5] и др.

Цел и задачи на дисертационния труд

Основната цел на дисертационния труд е:

Да се подобри и ускори изследването и проверката на нови хипотези за почти-пръстени над крайни циклични групи чрез разработката на специализирани софтуерни системи и модули.

За изпълнението на поставената цел формулираме следните задачи:

1. Използване на нови технологии и паралелни изчисления за

ускорение на генерирането на почти-пръстени над крайни циклични групи.

2. Разработване на нови софтуерни инструменти, които да улеснят работата на крайния потребител в изследването и визуалния анализ на почти-пръстени над крайни циклични групи.

3. Създаване на софтуерни модули за улесняване на класифицирането и анализа на различни групи от почти-пръстени над крайни циклични групи.

4. Формулиране на нови хипотези за структурата на почти-пръстени с крайни циклични групи с помощта на разработените специализирани софтуерни инструменти.

Структура и обем на дисертационния труд

Дисертационният труд се състои от увод, четири глави, заключение, публикации, списък с цитати и използвана литература.

В първа глава е направен обзор на основните алгебрични структури, като се започва от най-простите такива, и постепенно се изгражда до достигане на теорията за почти-пръстените над крайни циклични групи и работата с тях чрез компютърни програмни средства.

Във **втора глава** са описани нови подобрения на алгоритъма за генериране на почти-пръстени над крайни циклични групи, които спомагат за многократното забързване на процеса.

В **трета глава** е описано създаването на нов софтуерен модул, който улеснява изучаването на почти-пръстени над крайни циклични групи, като се дава възможност за извеждането на справки, филтриране и маркиране на вече генерирани почти-пръстени над крайни циклични групи по няколко различни начина.

В **четвърта глава** е представена разработката на нов софтуерен модул за работа с групи от почти-пръстени над крайни циклични групи и намиране на връзки между елементите на даден почти-пръстен над крайна и циклична група. С помощта на новите разработени софтуерни инструменти са формулирани две нови хипотези за структурата на почти-пръстени с крайни циклични групи.

В **заключението** е направено обобщение на реализацията на поставената цел и задачи, като се разглеждат и възможностите за бъдещо развитие.

Глава 1. Основни алгебрични обекти

Фокусът на Глава 1 е да даде дефиниции на основните алгебрични структури, използвани при изучаването на почти-пръстените над крайни циклични групи и са представени различни примери.

Пръстени

Определение 1.7.1. Нека R е множество, в което са дефинирани две алгебрични операции – събиране „+“ и умножение „$\cdot$“. Множеството R се нарича *пръстен*, когато са изпълнени следните аксиоми:

- R е Абелева група относно събирането;
- R е полугрупа относно умножението;
- За всеки $a, b, c \in R$ са изпълнени уравненията:

$$a \cdot (b + c) = a \cdot b + a \cdot c \text{ и } (a + b) \cdot c = a \cdot c + b \cdot c$$

Определение 1.7.3. Пръстен, в който умножението е комутативно, т.е. $ab = ba$, за произволни a, b се нарича *комутативен*.

Определение 1.7.7. Даден елемент a на пръстена R се нарича *нилпотент*, когато $a^n = 0$ за някое цяло $n \geq 1$.

Определение 1.7.9. Даден елемент e на пръстена R се нарича *идемпотент*, когато $e^2 = e$. Разбира се, тогава $e^n = e$ за всяко цяло $n \geq 2$.

1.1 Почти-пръстени

Определение 1.8.1. Множеството $(G, +, *)$ се нарича *ляв почти-пръстен*, ако изпълнява следните условия:

- G е група относно събирането;
- G е полугрупа относно умножението;
- от $a, b, c \in G$ следва, че $a * (b + c) = a * b + a * c$.

От левия дистрибутивен закон следва, че $x * 0 = 0$ за всяко $x \in G$ [16].

Определение 1.8.2. Почти-пръстенът G се нарича *нуласиметричен*, ако за всяко $x \in G$ е вярно $0 * x = 0$.

С други думи, почти-пръстените се различават от пръстените по две аксиоми: адитивната група не е непременно абелева и се изисква да е изпълнен само единият дистрибутивен закон [8, 9].

Всяка циклична група от ред n е изоморфна на остатъците по модул n, следователно в настоящия труд, без ограничение на общността се приема, че крайната циклична група G има вида

$$G = Z_n = \{0, 1, ..., n-1\}, 2 \leq n \leq \infty$$

Допълнително се въвеждат и следните означения:

- π – всяка функция, изобразяваща G в себе си;
- $+$ и $\cdot$ – събиране и умножение по модул n.

Нека е дадена група G и $|G| = n$. Следователно върху G съществуват n^{n^n} възможни операции, точно n^n от които са ляво дистрибутивни върху друга такава операция $+$. Съществува взаимно и еднозначно съответствие между тези n^n операции и функциите π, изобразяващи Z_n в себе си.

За да се осигури релация на еквивалентност върху множеството от всички почти-пръстени, имащи една и съща крайна циклична група, е необходимо да се осигурят:

- необходими и достатъчни условия върху π, така че операциите да са асоциативни;
- необходими и достатъчни условия върху множеството от операции върху π, така че те да дефинират изоморфизъм.

Теорема ([2], I)**.** Нека $*$ е бинарна операция над Z_n. Операцията $*$ е ляво дистрибутивна върху $+$ тогава и само тогава, когато съществува функция π, такава че за всички $p, q \in Z_n$ е изпълнено $p*q = \pi(p)\cdot q$.

Следствие. Над Z_n могат да бъдат дефинирани точно n^n ляво дистрибутивни операции.

Теорема ([2], II)**.** Необходимото и достатъчно условие функцията π да дефинира ляво асоциативна операция е

$$\pi(x) \cdot \pi(y) = \pi(x \cdot \pi(y))$$

за всички $x, y \in Z_n$.

Следователно, намирането на почти-пръстени над Z_n се свежда до намиране на функции π, удовлетворяващи горното равенство.

При изследването на почти-пръстените над крайни циклични групи се използва следният запис:

$$\langle k(x_0 x_1 ... x_{n-1}) \rangle$$

където k е поредният номер при генерирането на почти-пръстените над крайни циклични групи, а x_i е стойността на функцията $\pi : x_i = \pi(i), i \in Z_n$. Например $2(0001)$ означава вторият почти-пръстен над Z_4 със стойности на съответната функция $\pi : \pi(0) = \pi(1) = \pi(2) = 0, \pi(3) = 1$.

1.2 Генериране на почти-пръстени над крайни циклични групи

В тази подточка се обсъждат съществуващи алгоритми за генериране на почти-пръстени над крайни циклични групи, в частност алгоритмите от публикациите. [7, 11, 12, 13]. Най-бързият от тях прави пълна проверка на условието от предишната секция за елементите на функцията π. Разяснено е най-общо на какъв принцип се извършва генерирането на почти-пръстени над крайни циклични групи и каква е сложността на алгоритмите спрямо нарастването на $n - 2^n$ [10, 14].

Описаните в [7] алгоритми не могат да генерират почти-пръстените над $\mathbb{Z}_n$ за $n \geq 24$, където $(\mathbb{Z}_n, \cdot)$ има ненулеви нилпотенти [15] от втора степен, понеже броят на тези почти-пръстени е огромен и те не могат да бъдат генерирани в реално време. Поради това, по време на генерацията за $n \geq 24$ се пропускат цели групи почти-пръстени над крайни циклични групи; пресмята се само големината на тези групи от почти-пръстени над крайни циклични групи.

Глава 2. Ускоряване на генерирането на почти-пръстени над крайни циклични адитивни групи чрез паралелни изчисления

Фокусът на Глава 2 е върху преработката и ускорението на алгоритъма за генериране на почти-пръстени над крайни циклични групи, използвайки паралелни изчисления на `C#`, в частност `Parallel.For`. Използвани са също специализирани инструменти за анализ на код с цел намиране на проблемни точки в изпълнението на алгоритъма, обновена е версията на .NET рамката, както и са извършени оптимизации и допълнителни настройки на компилатора. Ускоренията от тези подобрения са измерени и представени на всеки етап чрез библиотеката BenchMark.NET.

2.1 Разделяне на почти-пръстените над крайни циклични групи на независими групи

За да са възможни паралелни изчисления при генерирането на почти пръстени над крайни циклични групи, то трябва общото количество работа да се раздели на по-малки, независими задачи, чиито резултати после да се съединят.

За целта разделяме множеството от почти-пръстени над крайни циклични групи от даден ред на подмножества, които могат да бъдат генерирани независимо едно от друго. Това дава възможност след това модулът за генериране да се преработи, така че да стартира едновременно множество инстанции на алгоритъма за генериране – броят им зависи от броя на независимите подмножества и от способностите на хардуера, който се използва за генериране.

За да генерираме почти-пръстените над дадена $\mathbb{Z}_n$, ги разделяме на поне шест подмножества, за които функцията π има предварително зададени стойности, показани във Фигура 1 на страница 11.

За почти-пръстени над $\mathbb{Z}_n$ със само тривиалните идемпотенти 0 и 1, подмножество 6 съдържа само един почти-пръстен. За $\mathbb{Z}_n$ с нетривиални идемпотенти, съществуват ненулеви симетрични почти-пръстени, за които стойността на $\pi(0)$ е равна на нетривиалния идемпотент. В този случай, разделяме подмножество 6 на допълнителни подмножества – едно за всеки нетривиален идемпотент в $\mathbb{Z}_n$.

subset 1: $\pi(0) = 0, \pi(1) = 0, \pi(2) = 0$
subset 2: $\pi(0) = 0, \pi(1) = 0, \pi(2) \geq 1$
subset 3: $\pi(0) = 0, \pi(1) = 1, \pi(2) = 0$
subset 4: $\pi(0) = 0, \pi(1) = 1, \pi(2) \geq 1$
subset 5: $\pi(0) = 0, \pi(1) \geq 2$
subset 6: $\pi(0) \geq 1$

Фигура 1: Независими подмножества в Zn

2.2 Реализиране на паралелната обработка в C#

Преработката на кода е фокусирана върху това генерирането на независимите подмножества да става под формата на масив от задачи, които могат да бъдат обработени чрез метода Parallel.For [17].

Променяме главната функция за генериране на почти-пръстени над крайни циклични групи, така че да приема като входни параметри началните и крайните стойности на π функцията. Тези стойности дефинират уникално подмножествата, описани в предишната секция на настоящия труд.

```
private static void MakeNearRings(int[] pi, int[]
    pi_2, int[] pi_n, List<int>[] pi_ptr)
```

След като задаваме началните стойности на π функцията и някои други параметри, започваме паралелното извикване на главната функция за генериране:

```
Parallel.For(0, subsetsnum, k =>
    {MakeNearRings(pi[k],pi_2[k],pi_n[k],pi_ptr[k])});
```

Калкулирането на всички подмножества може да се случи едновременно, ако са налични достатъчно процесорни ядра; тогава времето за изпълнение на програмата зависи единствено от времето, нужно да се изчисли най-голямото подмножество.

2.3 Изводи от процеса на оптимизация

Алгоритъмът, описан в настоящия труд, използва масиви за съхранение на междинни данни – това е най-добрият подход за нашия случай, защото не включва нищо повече от директен достъп до паметта при четене и запис на междинните данни. Ако няколко нишки

достъпват едновременно един и същи масив от масиви, те винаги вършат работата си на различни редове – няма сблъсък за достъпване на едни и същи данни.

Тествахме преработка на алгоритъма чрез използването на класове и обекти, както и експериментирахме с преместване на структурите от данни на стека. Нито един от тези подходи не доведе до ускорение на изпълнението.

2.4 Допълнителни техники за ускорението на генерирането на почти-пръстени

Направени са допълнителни оптимизации на проекта за генериране на почти-пръстени над крайни циклични групи. Тези подобрения ускориха процеса на генериране 4 пъти.

2.4.1 Оптимизации на платформата и компилатора

Платформата .NET Core на Майкрософт, която споделя голяма част от API-то си с .NET Framework, се използва за приложения, при които целта е скорост и скалируемост. Мигрирането на кода към .NET Core намали на половина времето за изпълнение за всяко n. Допълнително, повечето функции в нашия код са анотирани с AggresiveInlining флагове. Всичко това, в комбинация с още настройки на компилатора за оптимизация на изпълнението, допълнително намали времето за изпълнение.

2.4.2 Анализ на кода с инструментите на Visual Studio

Инструментите на Visual Studio за анализ на изпълняващ се код ни дадоха подробна информация за точки на забавяне при изпълнение на алгоритъма за генериране на почти-пръстени над крайни циклични групи. Чрез преработване на въпросните блокове код, успяхме да постигнем 50% увеличение в скоростта отгоре на вече постигнатото такова при преминаването към .NET Core. Главните проблемни точки се оказаха ненужни цикли и някои скъпи операции, които бяха заменени с по-оптимизирани такива.

2.5 Измерване на ускорението чрез Benchmark.NET

2.5.1 Benchmark.NET

Benchmark.NET е библиотека за измерване скоростта на даден .NET код върху конкретен хардуер. Кодът се изпълнява многократно и накрая се взимат усреднени стойности за скоростта му, за да се елиминират влиянията на други системни процеси върху изпълнението на кода т.е. резултатите от Benchmark.NET са много по-точни и надеждни от ръчно измерване на скоростта само с едно изпълнение [18]. Финалният резултат от измерването на скоростта на даден код с дадени входни параметри (ако има такива) се нарича *замерване* (от англ. *benchmark*).

2.5.2 Конфигурация, подготовка и хардуер

По време на описаните по-горе оптимизации, може да се разграничат пет различни основни версии на кода за генериране на почти-пръстени. Първата от тях е началното състояние на кода – тя се използва за отправна точка и за сравнение на скоростите на изпълнение на следващите четири версии. Общо петте версии са както следва:

1. Първоначално състояние на алгоритъма, без промени.
2. Паралелизираният вариант на кода, в който независими групи от почти-пръстени над крайни циклични групи се генерират едновременно (според възможностите на хардуера).
3. Вариант 2, но с ръчно поставени компилационни флагове Aggressive Inlining.
4. Вариант 3, но обновен да работи върху .NET Core 3.1.
5. Вариант 4, но изменен след анализ на кода с инструментите на Visual Studio и отстраняване на точките на забавяне (т.нар. bottlenecks).

Петте различни версии на кода се подготвят под формата на пет различни класа в един и същи нов .NET проект във Visual Studio. Прави впечатление, че в един и същи проект се замерва скоростта на код, който се изпълнява на базата на различни версии на .NET

Framework. Още повече, във финалните резултати са добавени и такива на базата на трета, най-актуална към момента версия на .NET Framework.

Първата група замервания се изпълнява на базата на .NET Framework версия 4.6.1 и включва версии на кода от 1 до 3. Правят се по три замервания на всяка версия, със различни стойности на входния параметър N – редът на почти-пръстените над крайни циклични групи, за които ще се извършва замерване на скоростта на генериране. Втората група замервания се базира на версия Core 3.1 на .NET Framework и версии 4 и 5 на кода за генериране. Третата група замервания се базира на версия 6 на .NET Framework и версия 5 на кода за генериране. Останалите анотации и настройки да идентични с първата група замервания.

Резултатите, представени в следващата секция, са получени след замерване на скоростта върху машина с параметрите, показани в Таблица 1 на страница 14.

Характеристика	**Стойност**
OS Name	Microsoft Windows 10 Pro
Version	10.0.19044 Build 19044
System Type	x64-based PC
Processor	Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz, 3301 Mhz, 4 Core(s), 4 Logical Processor(s)
Installed Physical Memory (RAM)	32,0 GB

Таблица 1: Системни параметри на машината, използвана за замервания

2.5.3 Измервания и резултати

На Фигура 2 на страница 15 са показани резултатите от замервания на първите три версии на алгоритъма за генериране на почти-пръстени над крайни циклични групи с ред 18, 20 и 23. Важно е да се отбележи рязкото ускорение – между десет и сто пъти – в скоростта на алгоритъма от версия 1 до версия 2, в която за пръв път е реализирано паралелното изчисление на отделни независими групи от почти-пръстени над крайни циклични групи. При наличието на

```
BenchmarkDotNet=v0.13.1, OS=Windows 10.0.19044.1469 (21H2)
Intel Core i5-6600 CPU 3.30GHz (Skylake), 1 CPU, 4 logical and 4 physical cores
.NET SDK=6.0.101
  [Host]               : .NET Core 3.1.22 (CoreCLR 4.700.21.56803, CoreFX 4.700.21.57101), X64 RyuJIT
  .NET Framework 4.6.1 : .NET Framework 4.8 (4.8.4420.0), X64 RyuJIT

Job=.NET Framework 4.6.1  Runtime=.NET Framework 4.6.1
```

Method	N	Mean	Error	StdDev	Ratio
v1Initial	18	32,650.2 ms	316.95 ms	296.47 ms	1.00
v2Parallel	18	351.3 ms	4.55 ms	4.26 ms	0.01
v3AggressiveInlining	18	336.2 ms	3.58 ms	3.17 ms	0.01
v1Initial	20	3,085.4 ms	29.61 ms	27.70 ms	1.00
v2Parallel	20	252.5 ms	4.98 ms	7.75 ms	0.08
v3AggressiveInlining	20	239.3 ms	2.74 ms	2.14 ms	0.08
v1Initial	23	15,015.1 ms	118.44 ms	105.00 ms	1.00
v2Parallel	23	1,487.3 ms	28.95 ms	33.34 ms	0.10
v3AggressiveInlining	23	1,409.9 ms	27.69 ms	27.19 ms	0.09

Фигура 2: Първа група резултати от замервания, Версии 1-3 върху .NET 4.6.1

още повече процесорни ядра, ускорението би било потенциално дори повече. Ускорението в скоростта между версия 2 и 3 е по малко, но въпреки това забележимо.

На Фигура 3 на страница 16 са показани резултатите от втората група замервания – версии 4 и 5 на алгоритъма за генериране на почти-пръстени над крайни циклични групи, изпълнявани върху версия 3.1 на .NET Core рамката. Забелязва се ускорение от два пъти във версия 4 спрямо версия 3 на алгоритъма за генериране на почти-пръстени над крайни циклични групи – единствената промяна в случая е преминаване към по-нова версия на .NET рамката – Core 3.1. Допълнително, забелязва се и ускорение във версия 5 спрямо версия 4, след подобренията на база на изследвания за проблеми точки на забавяне в кода чрез инструментите на Visual Studio.

На Фигура 4 на страница 17 са показани резултатите от третата група замервания – версия 5 на алгоритъма за генериране на почти-пръстени над крайни циклични групи, изпълнявани върху версия 6 на .NET рамката от 2022 година. Не се забелязва ускорение спрямо

```
BenchmarkDotNet=v0.13.1, OS=Windows 10.0.19044.1469 (21H2)
Intel Core i5-6600 CPU 3.30GHz (Skylake), 1 CPU, 4 logical and 4 physical cores
.NET SDK=6.0.101
  [Host]        : .NET Core 3.1.22 (CoreCLR 4.700.21.56803, CoreFX 4.700.21.57101), X64 RyuJIT
  .NET Core 3.1 : .NET Core 3.1.22 (CoreCLR 4.700.21.56803, CoreFX 4.700.21.57101), X64 RyuJIT

Job=.NET Core 3.1  Runtime=.NET Core 3.1
```

Method	N	Mean	Error	StdDev
v4NetCore	18	174.26 ms	2.299 ms	2.150 ms
v5NetCoreCpuOptimized	18	139.89 ms	1.329 ms	1.178 ms
v4NetCore	20	122.26 ms	2.281 ms	2.240 ms
v5NetCoreCpuOptimized	20	92.14 ms	1.022 ms	0.956 ms
v4NetCore	23	663.16 ms	12.746 ms	15.654 ms
v5NetCoreCpuOptimized	23	490.70 ms	9.690 ms	17.719 ms

Фигура 3: Втора група от резултати от замервания, Версии 3-4 върху .NET Core 3.1

изпълнението на същия код над версия 3.1 Core на .NET рамката.

Глава 3. Генериране на SQL заявки за филтриране на почти-пръстени над крайни циклични групи

3.1 Изследване на структурата на вече генерирани почти-пръстени

За да подобрим ефективността на изследователския процес, всички почти-пръстени се преместват в релационна база данни и се създава инструмент за тяхното филтриране, което да подобри изследователския процес. За всяко n, $3 \leq n \leq 25$ се прави отделна таблица, съдържаща почти-пръстените над $\mathbb{Z}_n$.

Колоните на всяка таблица в базата данни са, както следва:

- ‘N’ – пореден лексикографски номер на почти-пръстена;
- n числените полета (‘E1’, ‘E2’, ..., ‘En’) за всяка стойност на функцията π;
- ‘NOTE’ – поле за текстови коментари;

```
BenchmarkDotNet=v0.13.1, OS=Windows 10.0.19044.1469 (21H2)
Intel Core i5-6600 CPU 3.30GHz (Skylake), 1 CPU, 4 logical and 4 physical cores
.NET SDK=6.0.101
  [Host]   : .NET Core 3.1.22 (CoreCLR 4.700.21.56803, CoreFX 4.700.21.57101), X64 RyuJIT
  .NET 6.0 : .NET 6.0.1 (6.0.121.56705), X64 RyuJIT

Job=.NET 6.0  Runtime=.NET 6.0
```

Method	N	Mean	Error	StdDev
v5NetCoreCpuOptimized	18	131.14 ms	1.487 ms	1.391 ms
v5NetCoreCpuOptimized	20	89.18 ms	1.584 ms	1.482 ms
v5NetCoreCpuOptimized	23	488.40 ms	9.370 ms	12.508 ms

Фигура 4: Трета група резултати от замервания

- 'THEOREM' – индекс, който сочи към таблица с вече доказани или нови теореми;
- 'STATUS' – булево поле за маркиране на почти-пръстена.

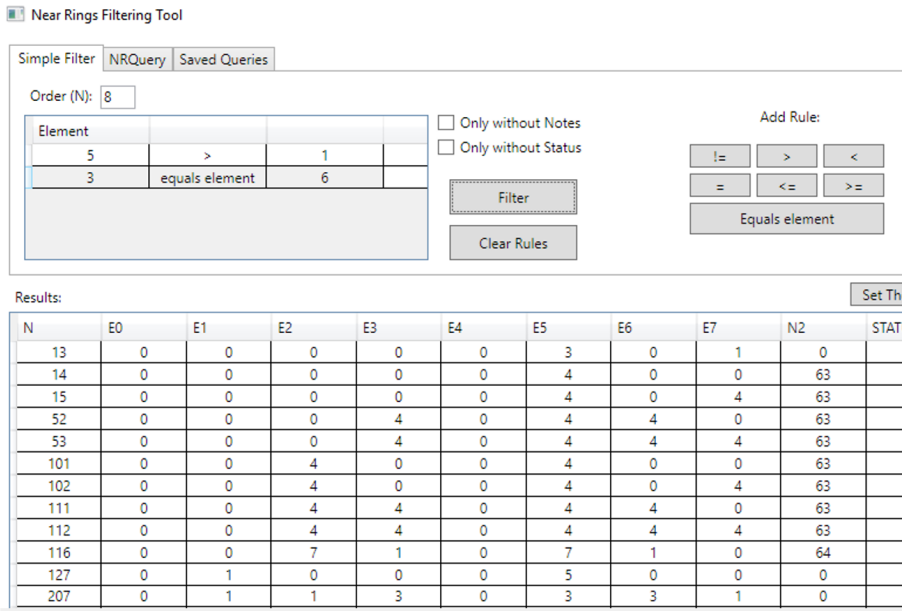
Колоните $NOTE, THEOREM, STATUS$ служат за маркиране на почти-пръстени с крайна и циклична група, които са описани от вече известни теореми.

3.2 Модул за визуализация и филтриране на почти-пръстени

В рамките на системата за изследване на почти-пръстени над крайни циклични групи [6] е реализиран модул, който изпълнява заявки за филтриране и визуализация на почти-пръстени. Модулът предлага два начина за описване на желаното подмножество от почти-пръстени.

Първият начин за описание на филтъра е чрез задаване на стойностите на функцията π за конкретен аргумент, използвайки бутони и падащи менюта за задаване на отношението или стойността на функцията π за този аргумент. Правилото се добавя към филтъра. На базата на избрания филтър се генерира и изпълнява SQL заявка към съответната таблица.

Например, за $n = 6$ и добавени филтри за $\pi(0) = 0$ и $\pi(3) = 3$ се генерира следната SQL заявка: **Select** $*$ **from** NR6 **where** E0 $= 0$ **AND** E3 $= 3$;



Фигура 5: Модулът за филтриране на почти-пръстени

Има две отметки – 'Only without Notes' (Само без бележки) и 'Only without Status'(Само без статус) на екрана. Когато са избрани, съответното условие ще бъде добавено към WHERE клаузата на SQL заявката.

Стойностите на аргументите варират от 0 до $n-1$. Отношението между е едно от следните: $<$, $>$, $=$, $!=$ и $\in$. Стойностите на функцията π са числа в интервала 0 и $n-1$, множества или интервали от цели положителни числа.

Тази функционалност на модула не е подходяща за сложни заявки за филтриране или за големи n, особено когато трябва да се зададат стойности на функцията π за почти всички аргументи. За тези цели е създаден допълнителен инструмент, описан в следващата подточка на дисертационния труд.

3.3 Формални заявки за филтриране на почти-пръстени

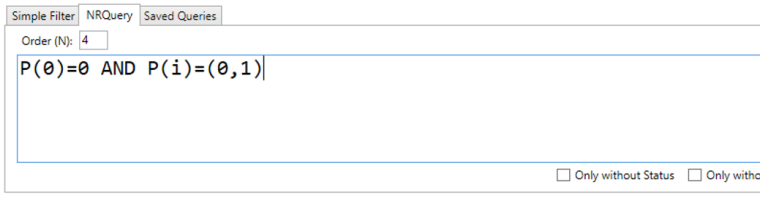
Създаден е нов инструмент за формално описание на подмножества от почти-пръстени над крайни циклични групи, разработен като част от програмния модул за филтриране. С помощта на такова формално описание, дадено специфично подмножество от почти-пръстени над крайни циклични групи може да бъде описано кратко и бързо. Например, за $n = 4$ с формалната заявка "`P(0)=0 and P(i)=(0,1)`", ще бъдат селектирани нула-симетричните почти-пръстени над крайни циклични групи със стойност на функцията за първия аргумент = 0 и стойности 0 или 1 за всички останали аргументи. Ще бъде генерирана SQL заявката, показана на Фигура 6). Графичният интерфейс е показан на Фигура 7.

```
select * from NR4
where
E0 = 0
and E1 in (0, 1)
and E2 in (0, 1)
and E3 in (0, 1);
```

Фигура 6: Генерираната SQL заявка

3.3.1 Описание на формалния език

За да дефинираме аргумент на π използваме `P (<argument>) = <value>` или `P (<списък от аргументи или интервали>) = <value>`,



Фигура 7: Текстовият редактор за формалната заявка

където аргументът може да е числена константа или специален идентификатор. Например: `P(0)=0, P(1,3,5)=1`.

За да дефинираме аргумента, може да използваме идентификаторите `odd` и `even`, които се асоциират с всички нечетни и четни аргументи. Използваме константи и прости аритметични изрази, свързани с n, например `P (n/2) = n/2`. Тези константи се калкулират предварително в зависимост от n.

Ако искаме да дефинираме стойностите на всички останали аргументи, използваме `i: P(0)=0 and P(n/2)=1 and P(i)=(0,3)`. За да зададем стойностите на функцията за даден аргумент, използваме константи, списъци или интервали в малки скоби: `P(0)!=0 and P(i)=(0,1,3)`. Когато имаме само `and` оператори в израза, те могат да бъдат заменени с ",": `P(0)=0, P(N/2)=0, P(odd)=(0,1)`.

Има друг специален вариант за изрази от следния тип: `P(2,5,8) = [1,4,7]`, които използваме за да означим, че стойностите на $\pi(2)$, $\pi(5)$, $\pi(8)$ могат да са равни на пермутацията от 1, 4 и 7.

За $n = 9$ от заявката `P(0)=0, P(2,5,8)=[1,4,7], P(i)=0` ще се генерира показаната по-долу SQL заявка.

```
select * from NR9
where
E0 = 0
and E2 in (1,4,7) and E5 in (1,4,7) and E8 in (1,4,7)
and (E2 + E5 + E8 = (1 + 4 + 7))
and E1 = 0 and E3 = 0 and E6 = 0 and E7 = 0;
```

Резултатът от заявката ще бъде:

(172) 0, 0, 1, 0, 0, 7, 0, 0, 4
(318) 0, 0, 4, 0, 0, 1, 0, 0, 7
(402) 0, 0, 7, 0, 0, 4, 0, 0, 1

Модулът дава възможността подмножеството да се запише във файл или в нова таблица в базата. Генерираните SQL заявки могат да бъдат запаметени в базата в специална таблица на заявките. Програмата дава възможност те да бъдат заредени и изпълнени отново.

3.3.2 Генериране на SQL заявка от формалното описание

За да се превърнат въведените формални заявки в SQL такива, е разработен специален парсер, който разпознава граматиката на формалния език. Работата му се състои в следните стъпки:

- Въведената формална заявка се разделя на базовите си елементи – смислени комбинации от символи, които по-късно ще бъдат разпознати като променливи, числа и константи;
- Следващата стъпка е да се намерят идентификаторите за константи и интервали от числа;
- Не-числените константи се заменят със своите предварително изчислени стойности и изразите се пресмятат;

Програмният модул съдържа предварително зададени правила, чрез които дадено позволено условие от формалния израз да се превърне в SQL код. Резултатът обработката на даден валиден формален израз е поредица от такива условия, които засягат една или повече позиции от структурата на почти-пръстените над крайни циклични групи от някакъв ред n. Условията се преобразуват във валиден SQL израз, резултатът от който се извежда в програмния модул за филтрация на почти-пръстени в табличен вид.

Глава 4. Намиране на зависимости на стойността на пореден елемент от стойностите на елементи на други позиции в почти пръстени над крайни циклични групи

Създаден е програмен модул, който при подадени за входни данни вече генерирани почти-пръстени над крайни циклични групи, автоматично ги анализира, анотира, филтрира и групира. Анализът се състои в това да се намират зависимости на стойността на даден елемент на почти-пръстена над крайна циклична група от стойностите на елементи на други позиции. Резултатите спомагат и улесняват по-нататъшните изследвания на структурата на почти-пръстени над крайни циклични групи.

Модулът зарежда почти-пръстени над крайни циклични групи от база от данни или от файл, който се придържа към стандартизирания запис за изследване на такива почти-пръстени. Изведените от модула данни също могат или да бъдат записани в база, или във файл, който отново се придържа към стандарта. Това дава възможност за обмен на данни с останалите програмни модули, разработени с цел изследване на почти-пръстени над крайни циклични групи.

Модулът е написан на Python 3.9, като за манипулация на данни се използват популярните библиотеки с отворен код numpy и pandas, които дават възможност за зареждане на и работа с огромни обеми от данни.

4.1 Разпознаване на почти-пръстени над крайни циклични групи, описани от вече съществуващи теореми

При зареждане на всички пръстени от ред n в паметта, те се преобразуват в Data Frame обект с три базови колони:

- ring_id – лексикографският номер на почти-пръстена с крайна и циклична група;
- ring – самият почти-пръстен над крайна циклична група;
- unique_nums – множеството от уникални елементи, които се съдържат в почти-пръстена над крайна циклична група.

На Фигура 8 е са показани в табличен вид заредените в паметта почти-пръстени над $\mathbb{Z}_5$.

Разработихме програмен метод за разпознаване на почти-пръстени над крайни циклични групи, които са описани от вече съществуващи теореми.

Изпълнението на кода на Фигура 9 ни дава информация дали даден почти-пръстен над крайна циклична група е вече описан от съществуваща теорема или не, т.е. се дава възможност по-нататъшни анализи и извадки да се извършват само върху все още неописани почти-пръстени над крайна циклична група.

Всяка теорема е записана под формата на кратък опростен израз, който обаче е и изпълним Python код. За всеки почти-пръстен се изпълнява последователно изразът, съответстващ на всяка теорема, с помощта на вградената Python функционалност eval(). Даден израз може да има резултат **True** или **False**. При резултат **True** проверката на теореми се прекратява и почти-пръстенът се маркира като вече обработен. Подробно описание на структурата и работата на тези Python изрази е дадено в следващата секция.

	ring_id	ring	unique_nums
0	1	[0, 0, 0, 0, 0]	(0,)
1	2	[0, 0, 0, 0, 1]	(0, 1)
2	3	[0, 0, 0, 1, 0]	(0, 1)
3	4	[0, 0, 0, 1, 1]	(0, 1)
4	5	[0, 0, 1, 0, 0]	(0, 1)
5	6	[0, 0, 1, 0, 1]	(0, 1)
6	7	[0, 0, 1, 1, 0]	(0, 1)
7	8	[0, 0, 1, 1, 1]	(0, 1)
8	9	[0, 0, 1, 4, 0]	(0, 1, 4)
9	10	[0, 0, 4, 1, 0]	(0, 1, 4)
10	11	[0, 1, 0, 0, 0]	(0, 1)

Фигура 8: Състояние на Data Frame обекта след първоначалното зареждане и обработка

```
def isMarked(ring):
    global r, n
    r=ring
    n=len(ring)
    flag=False

    for t in theorems:
        flag = eval(t)
        if(flag):
        break

    return flag
```

Фигура 9: Функцията, което проверява дали даден почти-пръстен над крайна циклична група е описан от някоя съществуваща теорема

4.2 Описание на теореми под формата на Python изрази

Разработихме метод за представяне на теореми за описание на множества от почти-пръстени над крайни циклични групи под формата на кратки и четими Python изрази.

За да могат тези изрази да са максимално кратки и четими, създадохме помощни функции, които оперират върху текущия разглеждан от алгоритъма почти-пръстен. Комбинирането на тези помощни функции с булевия оператор *and* дава възможност теоремите, описващи множества от почти-пръстени над крайни циклични групи, да бъдат описани под формата на Python изрази, които да приличат на лесно четим псевдокод и да могат да бъдат съхранявани като символни низове. В кода на Фигура 9 глобалният масив theorems съдържа именно тези символни низове.

4.2.1 Помощни функции за извършване на проверки върху почти-пръстен над крайна циклична група

Всяка помощна функция очаква стойността на текущо разглеждания почти-пръстен над крайна циклична група да е в глобално достъпна променлива, което означава, че той няма нужда да бъде подаван като параметър. Това означава, че цялостният запис на помощните функции – и в следствие, на изразите, представляващи теореми, е по-кратък.

Помощната функция **ALL**(allowed) проверява дали всички елементи на почти-пръстена с крайна циклична група принадлежат на елементите на масива allowed.

Помощната функция PI(x) връща стойността на елемент от почти-пръстена с крайна циклична група за позиция x, където x е елемент от крайната циклична група $\mathbb{Z}_n$.

Помощната функция ODD(x) проверява дали всички елементи на нечетни позиции в почти-пръстена с крайна циклична група принадлежат на масива от числа x. Това, заедно с аналогичната функция EVEN(x), е една от най-честите проверки в някоя теорема.

Функцията POS(x) връща всички елементи на крайната циклична група $\mathbb{Z}_n$, за които стойността на функцията π е x.

При наличието на по-сложни условия в дадена теорема се налага да се добави код, който намалява четимостта, но за сметка на това позволява да бъде описана произволна теорема. За целта се използва

помощната функция ALL_POS(l1, l2), която приема за параметри две анонимни функции (*lambdas*):

- l1 е условие, което трябва да е вярно за всеки елемент на почти-пръстена с крайна циклична група;
- l2 не е задължителна и се използва за филтриране на това върху кои позиции да се провери първото условие.

Вътрешно в ALL_POS се използва функционалността на Python наречена *list comprehension* – или условно преобразуване на един масив в друг на един ред. В следния код е показана проверка за това дали всички елементи на четни позиции имат стойност различна от 4:

```
ALL_POS(lambda pos, el: el!=4, lambda pos, el:
    pos%2==0)
```

4.2.2 Представяне на някои теореми под формата на Python изрази

В тази подсточка от дисертационния труд са дадени примери за записване на съществуващи теореми под формата на Python изрази. Първата теорема в списъка за проверки е известното вече множество от нуласиметрични почти-пръстени с крайна циклична група, които имат за елементи 0 и 1 – теорема 2.1.1 от [7]:

За всяко $n \geq 2$ *съществуват поне* 2^{n-1} *нуласиметрични почти-пръстена с крайна циклична група над* Z_n.

Записът на тази теорема в програмния модул, използвайки описаните вече помощни функции, е показан в следния код:

```
n>2 and PI(0)==0 and ALL([0,1])
```

4.3 Намиране на зависимости между елементи на почти-пръстен над крайна циклична група

След маркирането на вече описаните от съществуващи теореми почти-пръстени с крайна циклична група, те могат да бъдат премахнати от последващи анализи и извадки чрез филтриране. Такива филтрирани почти-пръстени с крайна циклична група остават заредени в паметта под формата на табличния тип данни DataFrame.

Към DataFrame-а се добавят N на брой празни колони, в които да се запишат намерените за всеки елемент връзки с други елементи. Разработен е нов алгоритъм, който проверява и записва списък от елементи, с които е свързан даден елемент в рамките на един почти-пръстен с крайна циклична група. Алгоритъмът се изпълнява върху всеки ред от DataFrame обекта. Така за дадени позиции X се образуват списъци от други позиции. Тези списъци се попълват в новите колони на DataFrame обекта.

Част от получените резултати за такава обработка при почти-пръстени над Z_6 са показани на Фигура 10. Забелязват се групи от почти-пръстени, които имат еднакви списъци със зависимости между елементите. Тези групи с еднакви резултати са изключително важни при по-нататъшния анализ и изследователка дейност с цел намиране на нови теореми, описващи множества от почти-пръстени над крайни циклични групи.

	ring_id	ring	unique_nums	isMarked	E0	E1	E2	E3	E4	E5
32	33	[0, 1, 0, 3, 0, 5]	(0, 1, 3, 5)	(False, None)	None	(1, 3, 5)	None	(3,)	None	(1, 3, 5)
33	34	[0, 1, 0, 3, 4, 3]	(0, 1, 3, 4)	(False, None)	None	(1, 3, 4)	None	(0, 3)	(0, 4)	(2, 3, 5)
45	46	[0, 1, 2, 3, 4, 5]	(0, 1, 2, 3, 4, 5)	(False, None)	None	(1, 2, 3, 4, 5)	(0, 2, 4)	(0, 3)	(0, 2, 4)	(1, 2, 3, 4, 5)
48	49	[0, 1, 4, 3, 4, 1]	(0, 1, 3, 4)	(False, None)	None	(1, 3, 4)	(0, 2)	(0, 3)	(0, 4)	(2, 3, 5)
54	55	[0, 3, 4, 3, 0, 1]	(0, 1, 3, 4)	(False, None)	None	(1, 3, 4)	(0, 2)	(0, 3)	None	(2, 3, 5)
60	61	[0, 5, 0, 3, 0, 1]	(0, 1, 3, 5)	(False, None)	None	(1, 3, 5)	None	(3,)	None	(1, 3, 5)
63	64	[0, 5, 4, 3, 2, 1]	(0, 1, 2, 3, 4, 5)	(False, None)	None	(1, 2, 3, 4, 5)	(0, 2, 4)	(0, 3)	(0, 2, 4)	(1, 2, 3, 4, 5)
65	66	[1, 1, 1, 1, 1, 1]	(1,)	(False, None)	(0,)	(1,)	(2,)	(3,)	(4,)	(5,)
66	67	[3, 1, 1, 3, 1, 1]	(1, 3)	(False, None)	(0,)	(1, 3)	(0, 2)	(3,)	(0, 4)	(3, 5)

Фигура 10: Почти-пръстени над Z_6 с информация за връзки между елементите.

4.4 Групиране и извеждане на данни

Преди да се изведат финалните данни се извършва групиране на почти-пръстените с крайна циклична група по колоната unique_nums и по комбинацията от стойности на всички колони с връзки между елементите. Това представяне спомага за изучаването им и намирането на нови теореми. На Фигура 11 са показани почти-пръстени над Z_5, групирани и сортирани по този начин.

	ring	unique_nums	E0	E1	E2	E3	E4
0	[0, 0, 0, 0, 1]	(0, 1)	()	()	()	()	(4,)
1	[0, 0, 0, 1, 0]	(0, 1)	()	()	()	(3,)	()
2	[0, 0, 0, 1, 1]	(0, 1)	()	()	()	(3,)	(4,)
3	[0, 0, 1, 0, 0]	(0, 1)	()	()	(2,)	()	()
4	[0, 0, 1, 0, 1]	(0, 1)	()	()	(2,)	()	(4,)
5	[0, 0, 1, 1, 0]	(0, 1)	()	()	(2,)	(3,)	()
6	[0, 0, 1, 1, 1]	(0, 1)	()	()	(2,)	(3,)	(4,)
7	[0, 1, 0, 0, 0]	(0, 1)	()	(1,)	()	()	()
8	[0, 1, 0, 0, 1]	(0, 1)	()	(1,)	()	()	(4,)
9	[0, 1, 0, 1, 0]	(0, 1)	()	(1,)	()	(3,)	()
10	[0, 1, 0, 1, 1]	(0, 1)	()	(1,)	()	(3,)	(4,)
11	[0, 1, 1, 0, 0]	(0, 1)	()	(1,)	(2,)	()	()

Фигура 11: Почти-пръстени с крайна циклична група и връзките в тях, групирани и сортирани

4.5 Нови хипотези за почти-пръстени над крайни циклични групи

С помощта на разработените програмни средства са направени хипотези за структурата на почти-пръстени над Z_n, където $n = 2^k, k > 2$.

Предположенията са проверени с разпределената система за тестване на хипотези за почти-пръстени [6] и са формулирани следващите две теореми. Проверките за теорема 4.5.1 са направени за $n = 8, 16, 32$.

Теорема 4.5.1. Нека $n = 2^k$, $k > 2$. Ако $\pi(0) = \pi(2^{k-1}) = 0$, $\pi(2^i * m) = 2^{i+1}, m = 1, 3, ..., 2^{k-i} - 1, i = 1, ..., k - 2$ и $\pi(2q + 1) \in \{2, 2^{k-1} + 2\}, q = 0, ..., 2^{k-1} - 1$, тогава условие (1) е изпълнено и има точно 2^{k-1} на брой почти-пръстена над $\mathbb{Z}_n$ отговарящи на условията.

Неформално описание на почти-пръстените описани в Теорема 4.5.1:

- за $n = 8$ функцията π има следния вид:

$$(0, x, 4, x, 0, x, 4, x),$$

  където $x \in \{2, 6\}$;

- за $n = 16$ функцията π има следния вид:

$$(0, x, 4, x, 8, x, 4, x, 0, x, 4, x, 8, x, 4, x),$$

където $x \in \{2, 10\}$;

- за $n = 32$ функцията π има следния вид:

$$(0, x, 4, x, 8, x, 4, x, 16, x, 4, x, 8, x, 4, x, 0, x, 4, x, 8, x, 4, x, 16, x, 4, x, 8, x, 4, x),$$

където $x \in \{2, 18\}$.

Теорема 4.5.2. Нека $n = 16$ или $n = 32$ и $\pi(0) = 0$. Нека ν_1 е най-малкият ненулев нилпотент от втора степен, а ν_3 е най-големият нилпотент от втора степен над Z_n. Ако $\pi(\nu_1 i) = \nu_1, i = 1, ..., 3, \pi(\nu_1 j) = 0; j = 1, 2, 3, j \neq i, \pi(\nu_1 m + i) = 1, m = 0, ..., 3, \pi(q) \in \{0, \nu_1\}$, за всички останали позиции q или $\pi(\nu_1 i) = \nu_3, i = 1, ..., 3, \pi(\nu_1 j) = 0; j = 1, 2, 3, j \neq i, \pi(\nu_1 m - i) = 1, m = 1, ..., 4, \pi(q) \in \{0, \nu_3\}$, за всички останали позиции q. Тогава условие (1) е изпълнено и има точно $6.2^{n/2}$ на брой почти-пръстена над Z_n отговарящи на условията.

Неформално описание на почти-пръстените описани в Теорема 4.5.2:

За $n = 16$ функцията π има следния вид:

$$(0, 1, x, x, 4, 1, x, x, 0, 1, x, x, 0, 1, x, x),$$
$$(0, x, 1, x, 0, x, 1, x, 4, x, 1, x, 0, x, 1, x),$$
$$(0, x, x, 1, 0, x, x, 1, 0, x, x, 1, 4, x, x, 1),$$
$$(0, y, y, 1, 12, y, y, 1, 0, y, y, 1, 0, y, y, 1),$$
$$(0, y, 1, y, 0, y, 1, y, 12, y, 1, y, 0, y, 1, y),$$
$$(0, 1, y, y, 0, 1, y, y, 0, 1, y, y, 12, 1, y, y),$$

където $x \in \{0, 4\}, y \in \{0, 12\}$.

Заключение и приноси

Изпълнена е основната цел на дисертационния труд – да се подобри и ускори изследването и проверката на нови хипотези за почти-пръстени над крайни циклични групи. Използвани са иновативни

технологии за разработката на софтуерни модули и системи, чрез които да се подобри цялостният работен процес, свързан с теорията на почти-пръстените с крайна циклична група.

Основните приноси на дисертационния труд са:

1. Направено е подробно изследване на работата на алгоритъма за генериране на почти-пръстени с крайна циклична група и са идентифицирани и предложени възможности за оптимизация. Генерирането на почти пръстени с крайна циклична група е ускорено значително, чрез направени: обновяване на софтуерната рамка, оптимизации на компилатора, доразвиване на алгоритъма и подхода на генериране и прилагане на техники за паралелна обработка.

2. Предложен и реализиран е нов метод на съхранение на почти-пръстени и теоремите, свързани с тях, в релационна база от данни. Създаден е софтуерен модул, който автоматично генерира SQL заявки за филтриране на множества от почти-пръстени с крайна циклична група, които се намират в релационна база от данни. Това улеснява изследователския процес, като позволява филтрирането на много големи обеми от почти-пръстени с крайна циклична група и извеждането на целеви извадки от тях.

3. Разработен е метод за автоматично обозначаване на почти-пръстени като описани от дадена съществуваща теорема. Реализиран е метод за описание на теореми за крайни почти-пръстени с крайна циклична група под формата на кратък Python израз. Създадени са помощни функции, които да се използват във формулировката на изразите и които правят записа четим за потребителите на Python модула. Това дава възможност да се изолират само неописаните до сега почти-пръстени с крайна циклична група за дадено n.

4. Разработен е алгоритъм, който анализира връзките между елементите в почти-пръстени с крайна циклична група. Алгоритъмът се прилага върху произволно големи групи от неописани почти-пръстени с крайна циклична група, които после се групират в клъстери с еднакви вътрешни зависимости. Това значително улеснява процеса по формулиране и доказателство на нови хипотези за почти-пръстени с крайна циклична група.

5. Направени са някои нови хипотези за структурата на почти-пръстени над Z_n, $n = 2^k, k > 2$, които са проверени с помощта на вече разработени софтуерни приложения и съответстват на генерираните почти-пръстени. Формулирани са две нови теореми за структурата и броя на почти-пръстени.

Връзка между приноси, задачи и публикации:

Принос	Задача	Секция в дисертационния труд	Публикация
1.	1	1.8, 1.9, 2.1, 2.2, 2.3, 2.4, 2.5	2, 3
2.	2	3.1, 3.2, 3.3	1, 3
3.	2, 3	4.1, 4.2	4
4.	2, 3	4.3, 4.4	4
5.	3, 4	4.5	

Перспективи за развитие

С помощта на новите разработени софтуерни модули могат да се формулират и докажат нови теореми за долни граници на почти-пръстени над крайни циклични групи. Има възможност да се опише напълно структурата на всички почти-пръстени над Z_n за $n \leq 16$.

С постигнатото ускорение на алгоритъма за генериране на почти-пръстени с крайна и циклична група чрез паралелни изчисления, може да се намери точния брой на почти-пръстените над Z_n за $37 \leq n \leq 40$.

След преработка на софтуерните модули могат да се изследват различни свойства на вече намерените почти-пръстени над Z_n, да се изследват планарни почти-пръстени и други крайни алгебрични обекти в областта на почти-пръстените и почти-полетата.

Апробация

Части от дисертационния труд са изготвени в изпълнение на задачи последните научни проекти:

- Проект ФП17-ФМИ-008 (2017-2018) „Иновационни софтуерни инструменти и технологии с приложения в научни изследвания по математика, информатика и педагогика на обучението“ към фонд „Научни изследвания“ на ПУ. (участник)

- Проект ФП19-ФМИ-002 (2019-2020) „Иновационни ИКТ за дигитално научноизследователско пространство по математика, информатика и педагогика на обучението“ към фонд „Научни изследвания“ на ПУ. (участник)

- Националната научна програма „Информационни и комуникационни технологии за единен цифров пазар в науката, образованието и сигурността (ИКТвНОС), Д01-205/23.11.2018 г., 2018-2021. (участник)

- Проект ФП21-ФМИ-002 (2021-2022) „Иновационни ИКТ за дигитално научноизследователско пространство по математика, информатика и педагогика на обучението“ към фонд „Научни изследвания“ на ПУ. (участник)

Публикации по дисертациония труд

1. Malinova, M., Golev, A, Rahnev, A., *Generating SQL Queries for Filtering Near-Rings on Finite Cyclic Groups*, International Journal of Pure and Applied Mathematics, Vol. 119, No. 1, 2018, pp. 225–234, ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (online version).

2. Malinova, M., Golev, A., Rahnev, A., *Speeding up the generation of near-rings on finite cyclic groups using parallel processing in C♯*, Neural, Parallel, and Scientific Computations, Vol. 26, No. 3, 2018, pp. 345-354, ISSN:1061-5369.

3. Malinova, M., Golev, A., Rahnev, A., *Research and accelerated generation of near-rings on finite cyclic groups*, Scientific Conference Innovative ICT in Research and Education: Mathematics, Informatics and Information Technologies, 29-30 November 2018, Pamporovo, pp. 27-35, ISBN: 978-619-202-439-0.

4. Malinova, M., *Investigating new dependencies in the structure of near-rings over finite cyclic groups*, November 2021, International Journal of Differential Equations and Applications, Vol. 20, No. 2, 2021, pp. 197-206, ISSN (Print): 1311-2872; ISSN (Online): 1314-6084; Scopus, **SJR 2021: 0.214**.

Използвана литература

[1] Dickson L., On finite algebras, Nachr. Acad. Wiss. Gottingen, (1905), 358–393.

[2] Clay J.R., The near-rings on a finite cycle group, Amer. Math. Monthly, 71, 1964

[3] G. Pilz, (1982), "Near-Rings: What They Are and What They Are Good For"in Contemp. Math., 9, pp. 97–119. Amer. Math. Soc., Providence, R.I., 1981.

[4] Lockhart, R., The Theory of Near-Rings, *Lecture Notes in Mathematics*, Vol. 2295, Springer Cham, 2021.

[5] Pilz, G., Weber, F., Mueller, W., Schaefer, J., Statistical methods to support difficult diagnoses, *Diagnostics*, Vol. 11 , No. 7, art. no. 1300, 2021, eISSN: 2075-4418, IF 2020: 3.706.

[6] Pavlov N., A. Golev, A. Rahnev, Distributed Software system for Testing Near-Rings Hypotheses and New Constructions for Near-Rings on Finite Cyclic Groups, *Int. Journal of Pure and Applied Mathematics*, **90**, No.3 (2014), 345–356.

[7] Golev A., Algorithms for Generating Near-rings on Finite Cyclic Groups, *Proceedings of the Anniversary International Conference REMIA 2010, Plovdiv*, 255–262, 10-12 December 2010.

[8] Aichinger Erhard, J. S. Wilson, Reinhard Poeschel, Dragan Masulovic, Completeness for concrete near-rings, in Journal of Algebra, Vol. 279, Nummer 1, Seite(n) 61-78, 9-2004, ISSN: 0021-8693

[9] Aichinger E., F. Binder, J. Ecker, R. Eggetsberger, P. Mayr and C.N obauer. SONATA: Systems Of Nearrings And Their Applications, Package for the group theory system GAP4. Johanes Kepler University Linz, Austria, 2008. http://www.algebra.uni-linz.ac.at/sonata/

[10] Brassard Gilles, Paul Bratley, Fundamentals of Algorithmics, Prentice-Hall, 1995.

[11] Golev A., A. Rahnev, Computing Classes of Isomorphic Near-rings on Cyclic Groups of Order up to 23, Scientific Works, Plovdiv University, vol. 37, book 3, Mathematics, 2010, 53–66, ISSN 0204–5249.

[12] Golev A.A., A.K. Rahnev, Computing Near-rings on Finite Cyclic Groups of Order up to 29, Compt. rend.Acad.bulg.Sci., 64, No.4, 2011, ISSN 1310–1331, IF 2009:0.204.

[13] Golev A., Algorithms for Generating Near-rings on Finite Cyclic Groups, Proceedings of the Anniversary International Conference REMIA 2010, Plovdiv, 10-12 December 2010, 255–262, ISBN 978–954–423–648–9.

[14] Sedgewick Robert, Algorithms, Addison-Wesley, 1984.

[15] Szeto G., On regular near-rings with no nilpotent elements, Math. Japon., 19 (1974), 65–70.

[16] http://PlanetMath.Org/encyclopedia/NearRing.html

[17] Parallel.For Method, https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel.for?view=net-6.0

[18] Overview of Benchmark.NET, https://benchmarkdotnet.org/articles/overview.html