Научна конференция „Иновационни ИКТ в научните изследвания и обучението: математика, информатика и информационни технологии“, 29-30 ноември 2018 г., Пампорово, България
Scientific Conference “Innovative ICT in Research and Education: Mathematics, Informatics and Information Technologies”, 29-30 November 2018, Pamporovo, Bulgaria

RESEARCH AND ACCELERATED GENERATION OF NEAR-RINGS ON FINITE CYCLIC GROUPS

Maria Malinova, Angel Golev, Asen Rahnev

Abstract. We reduce the generation time of near-rings on finite cyclic groups using the parallel processing provided by C#. A formal language to describe subsets from near-rings on finite cyclic groups has been created. Software modules for parallel generating and for filtering and viewing near-rings have been implemented. Both modules are a part of the system for generating and researching near-rings.

Key Words and Phrases: *near-rings, generating near-rings, generating sql query, parallel programming*

2010 Mathematics Subject Classification: *16Y30*

1. Introduction

An algebraic system $(G,+,*)$ is a *(left) near-ring* on $(G,+)$, if $(G,+)$ is a group, $(G,*)$ is a semigroup and $a*(b+c)=a*b+a*c$ for $a,b,c\in G$. The left distributive law yields $x*0=0$ for $x\in G$. A near-ring $(G,+,*)$ is called *zero-symmetric*, if $0*x=0$ holds for $x\in G$.

J. R. Clay initiated the study of near-rings, whose additive groups are finite cyclic in 1964 [2]. Some sufficient conditions for the construction of near-rings on any finite cyclic groups were obtained. In 1968 all near-rings on cyclic groups of order up to 7 were computed [3]. Later all near-rings on cyclic groups of order 8 [5], of order up to 12 [7], of order up to 13 [8], of order up to 15 [1] as well as of order up to 24 [12] and up to 29 [14], 32 [16], 35 [17] were computed.

We will assume G coincides with the set $\mathbb{Z}_n=\{0,1,\ldots,n-1\}$, $2\le n<\infty$ since every cyclic group of order n is isomorphic to the group of the remainders of modulo n. We will denote the functions mapping $\mathbb{Z}_n$ into itself by π, and the

Дата на получаване: 03.12.2018 г.
Дата на рецензиране: 29.03.2019 г.
Дата на публикуване: 17.05.2019 г.

addition and the multiplication modulo n we will denote by $+$ and $\cdot$ respectively. The equality $c = a \cdot b$ will be equivalent to the congruence $ab \equiv c \pmod{n}$.

It is known that there exists a bijective correspondence between the left distributive binary operations * defined on $\mathbb{Z}_n$ and the n^n functions π mapping $\mathbb{Z}_n$ into itself. If $r*1 = b$ defines the function $\pi(r) = b$, then according to [2, Theorem II], the binary operation * is left distributive exactly when, for any $x, y \in \mathbb{Z}_n$, the equality

$$\pi(x) \cdot \pi(y) = \pi(x \cdot \pi(y)) \tag{1}$$

holds.

According to the above result, obtaining of the near-rings on $\mathbb{Z}_n$ is equivalent to obtaining the functions π such that equation (1) holds. We represent a near-ring with a list of the values of function $\pi(r)$, $r = 0, \ldots, n-1$.

2. Speeding up the generation of near-rings

We want to reduce the generation time of near-rings on finite cyclic groups by using parallel processing in `C#` and especially `Parallel.For`. We want to speed up the process for generating near-rings [15] with minimal changes to the core functionality.

We try to divide the set of near-rings into subsets, which can be generated independently of one another. We use `Parallel.For` to start calculating the subsets simultaneously.

We divide the near-rings into at least 6 different subsets, for which the π function has the following preset values:

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: $\pi(0) \geq 1$

For near-rings over a $\mathbb{Z}_n$ with only the trivial idempotents 0 and 1, subset 6 contains only one near-ring.

For a $\mathbb{Z}_n$ with non-trivial idempotents, there exist nonzero symmetric near-rings, for which the value of $\pi(0)$ is equal to the non-trivial idempotent. In that

case we divide subset 6 into additional subsets – one for each non-trivial idempotent in $\mathbb{Z}_n$.

The number of near-rings in the subsets is proportional. There are exceptions for $\mathbb{Z}_n$ with non-trivial idempotents or nilpotents of order 2.

Our goal is to be able to compute different subsets simultaneously. To avoid synchronization, every near-rings subset has its own copy of the main data structures we use. If enough processing cores are available, the calculation of all of the subsets can happen simultaneously and then the execution time for the program is reduced to the time needed to calculate only the largest subset of near-rings.

Every subset needs 4 one-dimensional arrays. This means that in our software we have 4 jagged arrays (C# arrays of arrays) – `pi`, `pi_2`, `pi_n`, `pi_ptr` – that we use to find the values of π. The computation of each near-rings subset uses one row from each jagged array. The initialization is shown below.

```
for (int k = 0; k < subsetsnum; k++)
{
   pi[k] = new int[n];
   pi_2[k] = new int[n];
   pi_n[k] = new int[n];
   pi_ptr[k] = new List<int>[n];
   for (int i = 0; i < n; i++) pi_ptr[k][i] = new
   List<int>(n+2);
}
```

We change the main function for generating near-rings so that it accepts as input parameters the starting and ending values of the π function. These values uniquely define the subsets described in the previous section.

```
private static void MakeNearRings(int[] pi, int[] pi_2, int[]
    pi_n, Stack pi_ptr[])
```

After setting the initial values of the π function and some other parameters, we start the parallel calls to the main generator function.

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

Our technique utilizes 2D arrays for storing intermediate data – this is the best approach for our use case, because there is nothing more than direct memory access involved in the reading and writing of intermediate results. If there are multiple threads accessing the same 2D array at the same time, they are always

doing their work onto different rows – there is no fighting for resources or locking involved.

We made additional optimizations to out project, which sped up the generation 4 times:

- Moving the code base onto .NET Core halved the execution time for any given n. Most functions were also annotated with AggresiveInlining flags which, in combination with compiler flags for code optimization, additionally reduced the execution time.
- The Visual Studio tools for analyzing running code gave us insight into bottlenecks in the code base. With some refactoring of "heavy" code snippets, we were able to achieve a 50 % increase in speed over the above mentioned results.

3. Using advanced filtering for finding new correlations

So far, a text editor has been used for our research on generated near-rings and make various assumptions before being strictly proven. But the number of near-rings for big n increases extremely fast, and this kind of work has been quite slow.

For more efficient research, all the generated near-rings were stored in a database. For each n, $3 \le n \le 25$, a table containing near-rings above $\mathbb{Z}_n$ is created. Especially for near-rings over $\mathbb{Z}_{25}$, only 17 886 931 near-rings are saved in the table. A huge group of near-rings above $\mathbb{Z}_{25}$ that are described with a theorem [12, Theorem 9], are not included in the table. The number of these near-rings is $5^{20} = 95\ 367\ 431\ 640\ 625$ and practically they are not generated.

The fields of each table are as follows:

- 'N' – a consecutive number of a near-ring;
- n numerical fields ('E1', 'E2', ..., 'En') for each value of the function π;
- 'NOTE' – a field for text description and comments;
- 'THEOREM' – an index which points to a table containing the names of already proven and new theorems;
- 'STATUS' – boolean field to mark a near-ring.

We mark these near-rings, which are described with theorems or for which a hypothesis has been made. Several major groups of described near-rings are marked in the database in advance, such as those with all values of π: 0 or 1; those with values 0, 1 or $n-1$; and a few more fully described groups of near-rings. We created a tool for filtering the near-rings according to the values of the function π, and thus facilitates their investigation.

	Zero-symmetric	**Nonzero-symmetric**	**Total number**
$\mathbb{Z}_3$	6	1	7
$\mathbb{Z}_4$	16	1	17
$\mathbb{Z}_5$	28	1	29
$\mathbb{Z}_6$	65	33	98
$\mathbb{Z}_7$	111	1	112
$\mathbb{Z}_8$	349	1	350
$\mathbb{Z}_9$	1 169	1	1 170
$\mathbb{Z}_{10}$	807	393	1 200
$\mathbb{Z}_{11}$	1 311	1	1 312
$\mathbb{Z}_{12}$	4 467	1 055	5 522
$\mathbb{Z}_{13}$	5 263	1	5 264
$\mathbb{Z}_{14}$	10 505	5 256	15 761
$\mathbb{Z}_{15}$	21 783	6 215	27 998
$\mathbb{Z}_{16}$	16 834 653	1	16 834 654
$\mathbb{Z}_{17}$	72 816	1	72 817
$\mathbb{Z}_{18}$	15 032 215	610 684	15 642 899
$\mathbb{Z}_{19}$	286 380	1	286 381
$\mathbb{Z}_{20}$	876 919	109 847	986 766
$\mathbb{Z}_{21}$	1 164 023	304 834	1 468 857
$\mathbb{Z}_{22}$	2 225 545	1 111 088	3 336 633
$\mathbb{Z}_{23}$	4 371 615	1	4 371 616
$\mathbb{Z}_{24}$	15 821 973	2 619 758	18 441 731
$\mathbb{Z}_{25}$	95 367 449 527 555	1	95 367 449 527 556
$\mathbb{Z}_{26}$	34 749 177	17 400 576	52 149 753
$\mathbb{Z}_{27}$	286 174 087 734	1	286 174 087 735
$\mathbb{Z}_{28}$	207 919 830	19 570 310	227 490 140
$\mathbb{Z}_{29}$	273 300 895	1	273 300 896

Table 1. Number of near-rings on $\mathbb{Z}_n$, $3 \le n \le 29$ ***[14]***

Within the software system for research of near-rings [17], we implemented a module which performs requests for filtering and visualizing of near-rings. The module features two different options for describing the desired subset of near-rings.

The first way to describe the filter is by setting the values of the function π for a specific argument using buttons and drop-down menus to set the value of

the argument, relation and the value of the function π for this argument. The rule is added to the filter. Based on the selected filter, an SQL query is generated and executed for the corresponding table. For example, for $n=6$ and added filters for $\pi(0)=0$ and $\pi(3)=3$ it will generate the following SQL query:

```
Select * from NR6 where E0 = 0 AND E3 = 3;
```

This feature of the module is not convenient for large n, especially when we must set values of the π function for almost all arguments. It is also inconvenient in cases to select a subset of near-rings as follows: we want the values of π for k of chosen arguments to be exactly k values but in any order, for example: the values of $\pi(a), \pi(b), \pi(c)$ can be equal to a permutation of three X, Y and Z distinct values.

In order to be able to perform complex filtering queries on our database of near-rings, we created a formal language to describe a subset of near-rings in a short and quick way. This description is then converted to an SQL query by our software.

For example, for $n=4$ by the formal query `P(0)=0` and `P(i)=(0,1)}`, we will select zero-symmetric near-rings with value of the function for its first argument $=0$ and values 0 or 1 for all other arguments.

The following SQL query will be generated:

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

If we want to describe the near-ring $\pi(1,1,\ldots,1)$ it is enough to write "`P(i)=1`".

The system also supports special options like the following kind `P(2,5,8) = [1,4,7]`, which we use to denote that the values of $\pi(2)$, $\pi(5)$, $\pi(8)$ can be equal to a permutation of 1, 4 and 7.

For $n=9$ from the query `P(0)=0`, `P(2,5,8)=[1,4,7]`, `P(i)=0` the following SQL query will be generated:

```
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;
```

The result from the query will be:

```
(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
```

We generate the SQL for the query by parsing the formal description into its basic elements, finding the constant and range identifiers and replacing the non-numeric constants with their pre-defined values. We generate a rule for each element of the near-ring and output an SQL query. The program is written on C#, and uses the Microsoft SQL Server.

We are currently working on adding interactive features like inline edits and multi-select to the data grid, which shows the filtered near-rings. This will simplify the research process by enabling us to edit the data faster.

4. Conclusion

We have created a module for generating near-rings on finite cyclic groups, using the parallel processing provided by C#. The near-rings on $\mathbb{Z}_n$ are divided into at least 6 independent subsets and then their generation is executed simultaneously. Further optimizations were applied during refactoring, code analysis and porting the code to a newer framework. If enough processor cores are available the generation may be speed up around 5 times. We store the near-rings in a database and use our custom software for viewing and filtering subsets of near-rings to help us discover new theorems and correlations.

Acknowledgements

This work was partially supported by the project FP17-FMI-008 of the Scientific Fund of the University of Plovdiv “Paisii Hilendarski”, Bulgaria.

Referances

[1] Aichinger E., F. Binder, J. Ecker, R. Eggetsberger, P. Mayr and C. Nöbauer.

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/

[2] Clay, J., The near-rings on a finite cycle group, *Amer. Math. Monthly*, **71** (1964), 47–50.

[3] Clay, J., The near-rings on groups of low order, *Math. Zeitschr.*, **104** (1968), 364–371.

[4] Jacobson, R., The structure of near-rings on a group of prime order, *Amer. Math. Monthly*, **73** (1966), 59–61.

[5] Pilz, G., *Near-rings*, North-Holland, Amst., **23** (1977).

[6] Pilz, G., *Near-rings*, North-Holland, Amst., Revised edition, **23** (1983).

[7] Yerby, R., H. Heatherly, Near-Ring Newsletter, **7** (1984), 14–22.

[8] Rakhnev, A., G. Daskalov, Construction of near-rings on finite cyclic groups, *Math. and Math. Education*, Sunny Beach, Bulgaria, (1985), 280–288.

[9] Rakhnev, A., On near-rings, whose additive groups are finite cyclics, *Compt. rend. Acad. bulg. Sci.*, **39** No. 5 (1986), 13–14.

[10] Rahneva, O., A. Golev, N. Pavlov, Dynamic Generation of Testing Question in SQL in DeTC, *BAS, Cybernetics and Information Technologies*, **8**, No1 Sofia 2008, 73–81.

[11] Rahnev, A., A. Golev, Some New Lower Bounds for the Number of Near-rings on Finite Cyclic Groups, *Int. Journal of Pure and Applied Mathematics*, **59**, No.1 (2010), 59–75.

[12] Rahnev, A., A. Golev, Computing Near-rings on Finite Cyclic Groups, *Compt. rend. Acad. bulg. Sci.*, **63**, book 5 (2010), 645–650.

[13] Golev, A., A. Rahnev, Computing Classes of Isomorphic Near-rings on Cyclic Groups of Order up to 23, *Scientific Works, Plovdiv University*, **37**, book 3, Mathematics (2010), 53–66.

[14] Golev, A., A. Rahnev, Computing Near-rings on Finite Cyclic Groups of Order up to 29, *Compt. rend. Acad. bulg. Sci.*, **64**, No. 4, (2011), 461–468.

[15] 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.

[16] Golev, A., A. Rahnev, New results for near-rings on finite cyclic groups, *Proceedings of Annual Workshop Coding "Theory and Applications"*, 51–54, Gabrovo, December 2011.

[17] 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.

[18] Malinova, M., A. Golev, A. Rahnev, Generating SQL Queries for Filtering Near-Rings on Finite Cyclic Groups, *Int. Journal of Pure and Applied Mathematics*, **119**, No.1 (2018), 225–234.

Faculty of Mathematics and Informatics

University of Plovdiv "Paisii Hilendarski"

236 Bulgaria Blvd, Plovdiv 4003, Bulgaria

e-mails: mvmalinova@gmail.com, angelg@uni-plovdiv.bg,
assen@uni-plovdiv.bg

ИЗСЛЕДВАНЕ И УСКОРЕНО ГЕНЕРИРАНЕ НА ПОЧТИ-ПРЪСТЕНИ НАД КРАЙНИ ЦИКЛИЧНИ ГРУПИ

Мария Малинова, Ангел Голев, Асен Рахнев

Резюме. С използване на средствата за паралелно програмиране на C# ускоряваме намирането на почти-пръстени над крайни циклични групи. Създаден е формален език за описание на подмножества от почти-пръстени. Написани са програмни модули за паралелно генериране и филтриране на вече намерени почти-пръстени. Тези модули са част от програмна система за генериране и изследване на почти-пръстени над крайни циклични групи.