ALGORITHMS FOR GENERATING NEAR-RINGS ON FINITE CYCLIC GROUPS

Angel Golev

Abstract. *In the present work are described the algorithms that generate all near-rings on finite cyclic groups of order 16 to 29.*

Keywords: near-ring, finite cyclic group
2010 Mathematics Subject Classification: 16Y30

1. Introduction

J. R. Clay started the study of near-rings whose additive groups are finite cyclic ones in 1964 [2]. In 1968 all the near-rings on cyclic groups of order up to 7 were computed [3]. Later all the near-rings on cyclic groups of order 8 [7], up to 12 [12], up to 13 [9] and up to 15 [1] were computed.

In works [10, 5] calculating the number of all near-rings on $\mathbb{Z}_n$, $16 \le n \le 29$ is announced. In the present work the algorithms that generate these near-rings are described.

The annotations, used in this paper, are described in [10].

It is known [2] 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 the near-rings on $\mathbb{Z}_n$ is equivalent to obtaining functions π such that equation (1) holds.

2. Data Structure

We use the following notation for the near-rings

$$k)\ (\ x_0\ x_1\ \ldots\ x_{n-1}\ )\ , \tag{2}$$

where k is the number of the generated near-ring and x_i are the values of the function π: $x_i = \pi(i)$, $i \in \mathbb{Z}_n$.

For example, "2) (0 0 0 1)" means the second near-ring on $\mathbb{Z}_4$ with values of the function π: $\pi(0) = \pi(1) = \pi(2) = 0,\ \pi(3) = 1$.

In the developed programs we represent a near-ring by using the function π. To store values of π we use one-dimensional array of integers *pi*.

3. Algorithms for generating near-rings on finite cyclic groups

I check the correctness of described algorithms and programs by using a known number of near-rings on $\mathbb{Z}_n$, $n \leq 15$. For verification the number of near-rings on $\mathbb{Z}_n$, $n > 16$ the exact values for $\mathbb{Z}_n$ where n is prime are used and the number of non-zero-symmetric near-rings described in [8].

Algorithm 1

The elements of the function π are constructed consequently, by adding elements in the array *pi* which meet the Equation (1). If the new element of π does not meet (1), we go to the previous level. In the calculation of (1), the right side of equality it can happen that $x \cdot \pi(y)$ is greater than the number of the elements found so far. In this case, it is assumed that the new element fulfills Equation (1).

Function: EQUATION1(x, y, q)
Input: x, y – indexes of the elements of π, q – index of the last found element;
Operation: checks the Equation (1) for x and y;
Output: 1 – Equation (1) is satisfied; 0 – not.

```
function EQUATION1(x, y, q)
    t ← (x ∗ pi[y]) mod n
    if t ≤ q and ((pi[x] ∗ pi[y]) mod n) ≠ pi[t] then
        return 0
    else
        return 1
    end if
end function
```

Function: CHECKCONDITIONS (q)
Input: q – the index of new element of π;
Operation: checks the Equation (1) for each previous element of π and q
Output: 1 – Equation (1) is satisfied; 0 – not.

```
function CHECKCONDITIONS(q)
    if EQUATION1(q, q, q) = 0 then return 0
    end if
    for p ← 0, q−1 do
        if EQUATION1(p, q, q) = 0 then
            return 0
        end if
        if EQUATION1(q, p, q) = 0 then
            return 0
        end if
    end for
    return 1
end function
```

Because not all elements satisfy the Equation (1), the obtained function π must be checked again that all pairs of elements meet (1).

Here are used some programming techniques to improve the performance of the program. For example, to calculate $a \cdot b = a * b \mod n$ a two-dimensional array mod_n with pre-calculated elements of all products of numbers from 0 to n is used: $a \cdot b \equiv mod\_n[a, b]$.

This algorithm is much better than generating all possible functions π and verifying Equation (1). It is used to generate and find the number of all near-rings on $\mathbb{Z}_n$, $n \leq 23$. We also use this algorithm to verify the output of the next algorithms.

The accumulated empiric data from generation of these near-rings is used to make some hypotheses about the lower bounds of near-rings. Some properties are found, and are used to obtain the number of all near-rings on finite cyclic groups for larger n.

Algorithm 2

By definition, for Equation (1) to be fulfilled, the values of the function π must be a multiplicative subgroup of $(\mathbb{Z}_n, \cdot)$.

At the end of the function CheckConditions, if the right side of (1) is greater than q, we check whether this new value forms a multiplicative subgroup with previous values of π.

```
...
Inc(quantity[pi[q]])
for qn ← 0, q−1 do
    if quantity[qn] > 0 and quantity[(pi[qn] * pi[q]) mod n] = 0 then
        return 0
    end if
end for
...
```

Here we use an array *quantity*, which contains the number of different values of the function π.

This algorithm does not improve significantly the performance of the program, but the idea can be further developed as follows: The functions π can be generated only from elements of a previously found multiplicative subgroup.

Algorithm 3

In this algorithm we do a complete verification of Equation (1) for the new elements of the function π. In some cases this may result in inconsistent addition of new elements to the array *pi*.

These "inconsistent" elements can not be saved directly into the array *pi*. Therefore two new array pi_2 and pi_n are used. In the first we save the value of the "inconsistent" element, equal to $x \cdot \pi(y)$, and in the second array we save the number of occurrences of that value at this position, because the value can be produced on adding different elements. An array of pointers *pi_ptr* to lists of "inconsistent" elements is used. This helps us to remove these elements more easily.

Procedure: InsertNode $(q, p, value)$
Input: q – the index of new element, p – the value of $x \cdot \pi(y)$, $value$ – the value of the left side of (1);
Operation: adds a new element to the list for position q.

```
procedure InsertNode(q, p, value)
    pi_2[p] ← value
    Inc(pi_n[p])
    node_ptr ← NewNode(p)
    pi_ptr[q].ListAdd(node_ptr)
end procedure
```

Procedure: RemovePList (q)
Input: q – index of element of π;
Operation: removes the list for position q.

```
procedure RemovePList(q)
    if pi_ptr[q] = null then
        return
    end if
    for all node ∈ pi_ptr[q].list do
        Dec(pi_n[node.value])
    end for
    pi_ptr[q].ListRemove
    pi_2[q] ← −1
end procedure
```

For example, for a new element q of the function π with a value $pi[q]$ it calculates $q * \pi(q)$, which is equal to t and all $q * \pi(i) = t_{1i}$, $0 \leq i < q$ and $\pi(i) * q = t_{2i}$, $0 \leq i < q$. For all $t,\ t_{1i},\ t_{2i}$ we check:

a) if they are less than or equal to q, they are compared directly with the values in the array *pi*;

b) if they are greater than q, check whether there is a value in $pi\_2[q]$:

b1) if there is no element – add a new element;

b2) if there exists a value at this place:

b21) if the value is not equal to the value of new element – Equation (1) is not satisfied;

b22) else – add the new element to the list $pi\_ptr[q]$ and increase the element $pi\_n[q]$ and Equation (1) holds.

In this way of working, the obtained function π does not need to be checked again if all pairs of elements meet (1).

Function: EQUATION1(x, y, q)
Input: x, y – indexes of the elements of π, q – index of the last found element;
Operation: checks the Equation (1) for x and y;
Output: 1 – Equation (1) is satisfied; 0 – not.

```
function EQUATION1(x, y, q)
    ls ← (pi[x] * pi[y]) mod n
    t ← (x * pi[y]) mod n
    if t ≤ q then
        if ls ≠ pi[t] then
            return 0
        end if
    else
        if pi_2[t] ≠ −1 and pi_2[t] ≠ ls then
            return 0
        end if
        INSERTNODE(q, t, ls)
    end if
    return 1
end function
```

The function CHECKCONDITIONS does not change.

Practically the execution time of the algorithm is linear to the number of near-rings on $\mathbb{Z}_n$. The number of near-rings grows at least twice compared to the previous n. The complexity is $O(2^n)$.

Using some proven properties to calculate the number of near-rings on finite cyclic groups of order greater than 24

We cannot use the algorithms described above to generate all near-rings and to obtain the number of near-rings on $\mathbb{Z}_n$, $n \geq 24$ when $(\mathbb{Z}_n, \cdot)$ has nonzero nilpotents of second degree, because the number of these near-rings is very large ([10, Theorem 9]) and they can not be generated in real time.

In this case, to calculate the number of near-rings, we do not generate near-rings described in [10, Theorem 9]. On generating near-rings we skip entire groups of possible near-rings corresponding to this theorem. After that the number of these skipped near-rings is calculated. This can be done with reference to [5, Corrolary 17].

```
...
if i = nilp[1] and pi[i] = 0 then
    nilp_all ← 0
    nilp_zero ← 0
```

```
        for k ← 1, nilp[1]−1 do
            if pi[k] = 0 then
                INC(nilp_zero)
            end if
            if NILPOTENT(pi[k]) then
                INC(nilp_all)
            end if
        end for
        if nilp_zero < nilp[1]−1 and nilp_all = nilp[1]−1 then
            SKIP THE REST OF THE ELEMENTS OF π
        end if
    end if
    ...
```

In this way we calculate the number of near-rings on $\mathbb{Z}_n$, for n equal to 25 and 27.

For example the number of all near-rings on $\mathbb{Z}_{25}$ corrsponding to [10, Theorem 9] is 5^{20} or 95 367 431 640 625. According to [5, Corrolary 17] we do not generate near-rings which begin with values for function π:

$0, x_{11}, x_{12}, x_{13}, x_{14}, 0;$

$0, 0, 0, 0, 0, 0, x_{21}, x_{22}, x_{23}, x_{24}, 0;$

$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, x_{31}, x_{32}, x_{33}, x_{34}, 0$ and

$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, x_{41}, x_{42}, x_{43}, x_{44}, 0,$

where $x_{ij} \in \{d_1, \ldots, d_m\}$ and one x_{ik}, k=1, 2, 3, 4 at least has nonzero-value; In this case we generate near-rings corresponding to [10, Theorem 9] which have values for π:

$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, x_{51}, x_{52}, x_{53}, x_{54}$

and their number is $5^4 = 625$. For $\mathbb{Z}_{25}$ the program generated 17887556 near-rings and number of all near-rings on $\mathbb{Z}_{25}$ is $17887556 + 5^{20} - 5^4 = 95367449527556$.

Using the algorithms described, we generated and obtained the exact number of all near-rings on $\mathbb{Z}_n$, $16 \leq n \leq 29$. The results are presented in Table 1.

The obtained results for $\mathbb{Z}_{17}, \mathbb{Z}_{19}, \mathbb{Z}_{23}, \mathbb{Z}_{29}$ (n is prime) are identical with the exact values from [6] and the obtained results for non-zero-symmetric near-rings on $\mathbb{Z}_n$, $n = 6, 10, 14, 15, 21, 22, 26$ ($n = p.q$, p and q are primes) are identical with the exact values from [8].

	Zero-symmetric	Non-zero-symmetric	Total number
$\mathbb{Z}_{16}$	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 $\mathbb{Z}_n$, $3 \leq n \leq 29$.

	Number of near-rings	Algorithm 1	Algorithm 3
$\mathbb{Z}_{15}$	27 998	0:00	0:00
$\mathbb{Z}_{16}$	16 834 654	1:40	0:28
$\mathbb{Z}_{17}$	72 817	0:01	0:01
$\mathbb{Z}_{18}$	15 642 899	2:50	0:37
$\mathbb{Z}_{19}$	286 381	0:04	0:02
$\mathbb{Z}_{20}$	986 766	0:22	0:06
$\mathbb{Z}_{21}$	1 468 857	0:25	0:11
$\mathbb{Z}_{22}$	3 336 633	0:51	0:22
$\mathbb{Z}_{23}$	4 371 616	1:07	0:30
$\mathbb{Z}_{24}$	18 441 731	34:20	2:05
$\mathbb{Z}_{25}$	95 367 449 527 556	»	»
$\mathbb{Z}_{26}$	52 149 753	»	6:03
$\mathbb{Z}_{27}$	286 174 087 735	»	»
$\mathbb{Z}_{28}$	227 490 140	»	25:17
$\mathbb{Z}_{29}$	273 300 896	»	35:00

Table 2. Execution time of programs with algorithms 1 and 3 in minutes and seconds. Programs are executed on CPU: Intel(R) Core(TM)2 Duo P8600 @ 2.40GHz

4. Conclusion

By using new algorithms we computed the numbers of all near-rings on $\mathbb{Z}_n$, $16 \leq n \leq 29$.

The empiric data accumulated from the generated near-rings allows constructing hypotheses and improve the lower bounds for the number of near-rings on finite cycling groups in [10, 5].

Acknowledgements

This research has been partially supported by the project of Bulgarian National Scientific Found from 2010.

References

[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. R., The near-rings on a finite cycle group, Amer. Math. Monthly, 71, 1964, 47–50.

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

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

[5] Golev A. A., A. K. Rahnev, Computing Near-rings on Finite Cyclic Groups of Order up to 29, Compt. rend. Acad. bulg. Sci. (to appear).

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

[7] Pilz G., Near-rings, North-Holland, Amst., 23, 1977.

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

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

[10] Rahnev A. K., A. A. Golev, Computing Near-rings on Finite Cyclic Groups, Compt. rend. Acad. bulg. Sci., 63, No. 5, 2010, 645–650.

[11] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill, 2001.

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

Angel Golev Faculty of Mathematics and Informatics
University of Plovdiv
236 Bulgaria Blvd.
4003 Plovdiv, Bulgaria
e-mail: angelg@uni-plovdiv.bg