T.C.
GEBZE TECHNICAL UNIVERSITY

Departmant Of Computer Engineering

ORTHOGONAL DESIGNS

Bengi YÖRÜKOĞLU

Project Advisor:
Dr. Ogr. Uye Zafeirakis Zafeirakopoulos

May, 2019
Gebze, KOCAELİ

T.C.
GEBZE TECHNICAL UNIVERSITY

Departmant Of Computer Engineering

ORTHOGONAL DESIGNS

Bengi YÖRÜKOĞLU

Project Advisor:
Dr. Ogr. Uye Zafeirakis Zafeirakopoulos

May, 2019
Gebze, KOCAELİ

This work has been accepted as .... / .... / 200 .. on the date of the graduation project in the Department of Computer Engineering by the following jury.

Graduation Project Jury

Consultant Name	Zafeirakis ZAFEIRAKOPOULOS	
University	Gebze Technical University	
Faculty	Faculty of Engineering	

Name of Jury	Gökhan KAYA	
University	Gebze Technical University	
Faculty	Faculty of Engineering	

PREAMBLE

I would like to express my sincere gratitude to Dr. Öğr. Üye Zafeirakis ZAFEIRAKOPOULOS, who has contributed to the preparation of the project, and to the Gebze Technical University supporting this study. In addition, during my education, I offer my full support to my family and to all of my teachers, who are exemplary with their lives.

May, 2019 Bengi Yörükoğlu

CONTENTS

LISTS OF FIGURES

LISTS OF ABBREVATION

ODs : Orthogonal Designs

SUMMARY

Orthogonal designs (ODs) are square matrices with entries in the field of quotients with certain orthogonality properties while complementary sequences are tuples of sequences with zero autocorrelation function. Orthogonal designs (ODs) have a great number of applications in Cryptography , Statistics and Telecommunications. [1]

The goal of the project is to create new sequences of zero autocorrelation using the Algebraic Model. These sets of sequences can be used to generate desired Orthogonal Designs. Orthogonal Designs have 2 lemmas, one of them is equating and the other is killing. In Equating we can change the entry associated with the indices and it is still orthogonal designs. Main goal of the project is to implement an algorithmic version for the reverse operations of Equating and to produce different sequence of tuples tuples from a given sequence of tuples as more as possible. Therefore, I refer to the reverse of Equating as Splitting.

The Algebraic System which I've used has 4 polynomial equations. These are zero autocorrelation , binary condition, and 2 type condition polynomial. The system provides solutions. I consider that I have a tuple of sequences and each position of the tuple may change with assigning to a new variable according to the solution of the algebraic system.

When the algebraic system is implemented and the sequence of tuples is given as a parameter, new tuples are observed. Afterwards in order to eliminate the tuples that are the same as each other I construct a struct of dictionary. I observed that most of the sequence of tuples which created with the system are the same. The reason I erase these identical tuples is for shorter duration and don't unnecessary calls when system make recursive calls to generate new sequences of tuples from each sequences.

I implemented the system in 2 stages. Firstly, I gave a parameter the system as a sequence of tuples. Then, as the process continued, I stopped moving through the characters and gave the system the indices of the each different characters should be found.

My success criterias are discovering new zero autocorrelation tuples and populating database at least two hundred and fifty tuples. I have discovered new zero autocorrelation tuples with the system solving.

1. INTRODUCTION

Orthogonal designs (ODs) are square matrices with entries in the field of quotients of the integral domain Z[a1,a2,...,a`] with certain orthogonality properties while complementary sequences are tuples of sequences with zero autocorrelation function and elements from the same domain as the orthogonal designs. Orthogonal designs have numerous applications in Statistics, Telecommunications, Coding Theory and Cryptography, see [2, 6, 7]. An OD of order n and type (t1,t2,...,t`) denoted OD(n;t1,t2,...,t`) in the commuting variables a1,a2,...,a`, is a square matrix D of order n with entries from the set {0,±a1,±a2,...,±a`} satisfying DDT =P` i=1(tia2 i)In, where In is the identity matrix of order n.[1]

Orthogonal Designs have 2 lemmas. One of them is killing and the other is equating. In Equating we can change the entry associated with the indices. In killing we can destroy the entry in the field that we want to clear off. As in the example below:

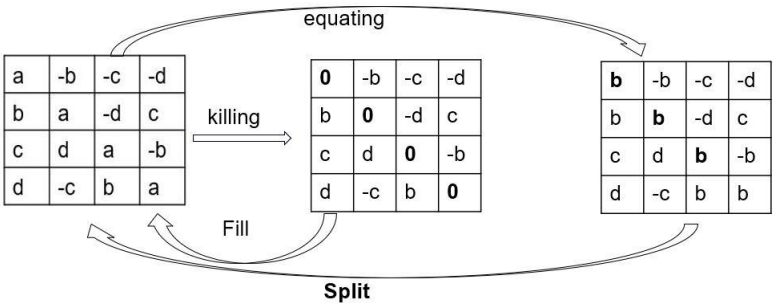

Figure 1: ODs killing and equating

The above illustration is given as example of killing and equating. Character ‘a’ replaced with 0 in the first matrix. In the second matrix, instead of character ‘a’, 0 is written. This process is defined as killing. In the first matrix we can change ‘a’ with ‘b’ as we see in the third matrix. In the third matrix, instead of ‘a’, ‘b’ is written. This process is defined as equating. Main goal is to provide an reverse operations of Equating , name is split.

In this project, I focused on complementary sequences. With using algebraic model, I tried to generate a new set of complementary sequences. These generated sequences can be used to produce orthogonal designs. I used Python programming language and SageMath in all process.

Firstly I implemented the algebraic system to solve all equations that containing 4 polynomial equations such as zero autocorrelation, binary condition and other 2 polnomial for type condition. System takes parameter as a sequence of tuples than according to each solution after system solved, system introduce two

new(signed) symbols to the tüple according to rule of the solution. The example of the start sequence of tuples and the one of the final sequence of tuples is as follows :

T=[[a,b,a],[a,b,-a],[b,-a,b],[b,d,-b]] ⟶ T'=[[a,b,-c],[-c,b,-a],[b,c,b],[b,d,-b]]

Numbers of the each character in sequences of tuples T are:
(b:6, a:5, d:1)

Numbers of the each character in sequences of tuples T' are:
(b:6, c:3, a:2, d:1)

When we examine the T and T' arrays, we realize that T' also contains character of c(signed). That mean when the system is solved solution can divide the character into two parts. Introduce new 2 symbols to the tuple. Also. We can reliaze that number of character 'a' of T' is 2 and the number of new symbol 'c' of T' is 3. When we add this numbers we can reach initial number of the symbol that we want to split.

2.METOT

The system that I've implemented in order to generate new sequences is as follow:

SPLIT

$$S_s = \begin{cases} \mathrm{AF}_{T'}(s) \text{ for } s \in [n] & \rightarrow 1 \text{ Autocorrelation Function} \\ B_i = x_i^4 - 1, i \in [m] & \rightarrow 2 \text{ Bounded Discrete Variables} \\ T_1 = \prod_{i=1}^{B} (x_t - i) & \rightarrow 3 \text{ Type conditions first polynomial} \\ T_2 = \left(\sum_{i=1}^{m} x_i^2\right) - m + 2x_t & \rightarrow 4 \text{ Type conditions second polynomial} \end{cases}$$

Figure 2 System for split [2]

Obtain new sequences, algebraic system is implemented takes 2 parameters.

createSystemForSplit([[a,b,a],[a,b,-a],[b,-a,b],[b,d,-b]] ,'a')

We have a sequence of tuples T=[[a,b,a],[a,b,-a],[b,-a,b],[b,d,-b]] to give a system to generate new sequences. After we find the number of each character we see that the number of symbol a is 5, number of symbol b 6 and the number of symbol d is 1.
Firstly , we choose the variable to split to create new sequence of tuple. This is the second parameter of createSystemForSplit method. If we choose symbol a, the system produces up to the number of symbols to replace with the chosen symbol. New sequences after replace is :

T’= [[x1,b,x2],[x3,b,x4],[b,x5,b],[b,d,-b]]

Then system use these 4 polynomial above the figure 2 to produce solutions.
Autocorrelation Function multiply each two symbol shifted by k. If k=1, the output polynomial of the autocorrelation function is x1*b+x2*b+x3*b+x4*b+2*x5*b
The system use bounded discrete variables because each variable can take infinite set of integers. After using this function on new generated variable replacing with a, system bound the variable with 1, -1, i, -i. Output polynomials of this function are $x1^4$-1, $x2^4$-1, $x3^4$-1, $x4^4$-1, $x5^4$-1. We use both type condition first polynomial and type conditions second polynomial to understand whether the selected symbol can be divided into 2 in a tuple. If x_t is found 2 after the system is solved that means system can split variable a into 2 part. If x_t is found 1 then system can’t split variable into 2 part. To explain this problem through the example in figure 3:

for k=1: b*x1+b*x2+b*x3+b*x4+2*b*x5
$x1^4$-1, $x2^4$-1, $x3^4$-1, $x4^4$-1, $x5^4$-1
xt^2-3*xt+2
$x1^2 + x2^2 + x3^2 + x4^2 + x5^2$ + 2*xt -5

Figure 3: All polynomial

After the system solved we can see some of the solution as below:

```
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: -1.00000000000000, x_1: -1.00000000000000*I, x_0: -1.00000000000000*I}
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: -1.00000000000000, x_1: 1.00000000000000*I, x_0: 1.00000000000000*I}
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: -1.00000000000000*I, x_1: -1.00000000000000, x_0: 1.00000000000000*I}
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: -1.00000000000000*I, x_1: -1.00000000000000*I, x_0: 1.00000000000000}
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: -1.00000000000000*I, x_1: 1.00000000000000*I, x_0: -1.00000000000000}
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: -1.00000000000000*I, x_1: 1.00000000000000, x_0: -1.00000000000000*I}
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: 1.00000000000000*I, x_1: -1.00000000000000, x_0: -1.00000000000000*I}
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: 1.00000000000000*I, x_1: -1.00000000000000*I, x_0: -1.00000000000000}
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: 1.00000000000000*I, x_1: 1.00000000000000*I, x_0: 1.00000000000000}
{x_t: 2.00000000000000, x_3: -1.00000000000000, x_2: 1.00000000000000*I, x_1: 1.00000000000000, x_0: 1.00000000000000*I}
```

Figure 4: Some of the results

For all result x_t and all new variable that replaced with a is found. If x_t of the one of the solution is 2 this means system can split chosen symbol of T into 2 part.

T= [[a, b, a],[a, b, -a],[b, -a, b],[b, d, -b
Numbers of the each character in sequences of tuples T are:
(b:6, a:5, d:1)

Figure 5: sequence of tuples T

T’=[[a,b,-c],[-c,b,-a],[b,c,b],[b,d,-b
Numbers of the each character in sequences of tuples T’ are:
(b:6, c:3, a:2, d:1)

Figure 6: Sequence of tuples T’

If we examine T in figure 5 return to T’ in figure 6 we become aware of splitting of symbol a into 2 part.

After splitting into two parts, total number of new symbol c and a of T' equal to number of symbol a of T.

The system determines what should replace the new variables that are replaced for the selected symbol, according to the following rule in figure 7.

$$\text{SPLIT}\left\{\begin{array}{r} \text{if } \alpha_i = 1 \text{ then } x_i = a \\ \text{if } \alpha_i = -1 \text{ then } x_i = -a \\ \text{if } \alpha_i = i \text{ then } x_i = \text{ new} \\ \text{if } \alpha_i = -i \text{ then } x_i = \text{-new} \end{array}\right\}$$

Figure 7 – Rule for split

That means If x1 (one of the new variable that replaced with a) is 1 then the symbol that system want to split remain the same. If x1 is -1 than the new symbol for x1 is negative of the same value. If x1 is i than we create new value to replace with x1. If x1 is- i than we create negative of the new value to replace with x1.

There is a **createNewTuples** method to achieve replacing symbols with the right ones according to rule for split. Here are the some of the new sequence of tuples produced from the parameters as figure 8:

1

```
[['a', b, 'a'], ['-c', -b, '-c']]
[['a', b, 'a'], ['-c', -b, '-c']]
[['a', b, '-c'], ['a', -b, '-c']]
[['c', b, 'a'], ['a', -b, '-c']]
[['-c', b, 'a'], ['a', -b, '-c']]
[['a', b, 'c'], ['a', -b, '-c']]
[['a', b, '-c'], ['a', -b, '-c']]
[['-c', b, 'a'], ['a', -b, '-c']]
[['c', b, 'a'], ['a', -b, '-c']]
[['a', b, 'c'], ['a', -b, '-c']]
```

2

```
[['a', b, 'a'], ['c', -b, '-c']]
[['a', b, 'a'], ['c', -b, '-c']]
[['a', b, '-c'], ['-c', -b, 'a']]
[['c', b, 'a'], ['-c', -b, 'a']]
[['-c', b, 'a'], ['-c', -b, 'a']]
[['a', b, 'c'], ['-c', -b, 'a']]
[['-c', b, '-c'], ['a', -b, 'a']]
[['c', b, 'c'], ['a', -b, 'a']]
[['c', b, '-c'], ['a', -b, 'a']]
[['-c', b, 'c'], ['a', -b, 'a']]
[['a', b, '-c'], ['c', -b, 'a']]
```

Figure 8: All tuples

When we look at the boxes 1 and 2, we see that the tuples are completely the same. With the **findDifferentTuples** method the program take all produced sequence of tuples and made a list of different ones by eliminating one of the different ones.

```
[['a', 'b', 'a'], ['-c', '-b', '-c']]
[['a', 'b', '-c'], ['a', '-b', '-c']]
[['c', 'b', 'a'], ['a', '-b', '-c']]
[['-c', 'b', 'a'], ['a', '-b', '-c']]
[['a', 'b', 'c'], ['a', '-b', '-c']]
[['a', 'b', 'a'], ['c', '-b', '-c']]
[['a', 'b', '-c'], ['-c', '-b', 'a']]
[['c', 'b', 'a'], ['-c', '-b', 'a']]
[['-c', 'b', 'a'], ['-c', '-b', 'a']]
[['a', 'b', 'c'], ['-c', '-b', 'a']]
[['-c', 'b', '-c'], ['a', '-b', 'a']]
[['c', 'b', 'c'], ['a', '-b', 'a']]
[['c', 'b', '-c'], ['a', '-b', 'a']]
[['-c', 'b', 'c'], ['a', '-b', 'a']]
[['a', 'b', '-c'], ['c', '-b', 'a']]
[['-c', 'b', 'a'], ['c', '-b', 'a']]
```

Figure 9: Tuples after eliminating

We can see above that the exact same ones have been deleted. After eliminating process I thought of a recursive algorithm that could produce production for the results of all results.

```
def recursiveOperation(listTuples,character):

    if (len(listTuples)==0):
        print("finished")
    else:
        print(listTuples[0])
        S,L,s,Vm,VA= createSystemForSplit(listTuples[0],character)
        listAllSeq=createNewTuples(S,L,character,Vm)
        listAllDif=findDifferentTuples(listAllSeq)
        if(len(listAllDif)!=0):
            resultMap={}
            resultMap=createMap(listAllSeq[0])
            createDictForTuples(resultMap,listAllDif)

        if(len(S)!=0):
            if(S[0].values()[0]!=0 and len(listTuples)!=0 and len(listAllDif)!=0):
                for i in listAllDif:
                    listTuples.append(i)
                    createList.append(i)

        recursiveOperation(listTuples[1:],character)
```

Figure 10: recursiveOperation method

In this recursive operation method, recursive operation proceed through listTuples parameter. New sequence of tuples are added to the end of the listTuples list. In this method previously described functions are called such as createSystemForSplit,

createNewTuples and findDifferentTuples. When the end of the list is reached in a recursive manner, all generated tuples are being collected in a dictionary created with the createDictForTuples method.

CreateDictForTuples

This function creates a dictionary which value is tuple and the key is number of the each symbol in a sequence of tuple. After the recursive operation is over the structure of the operation is as follow:

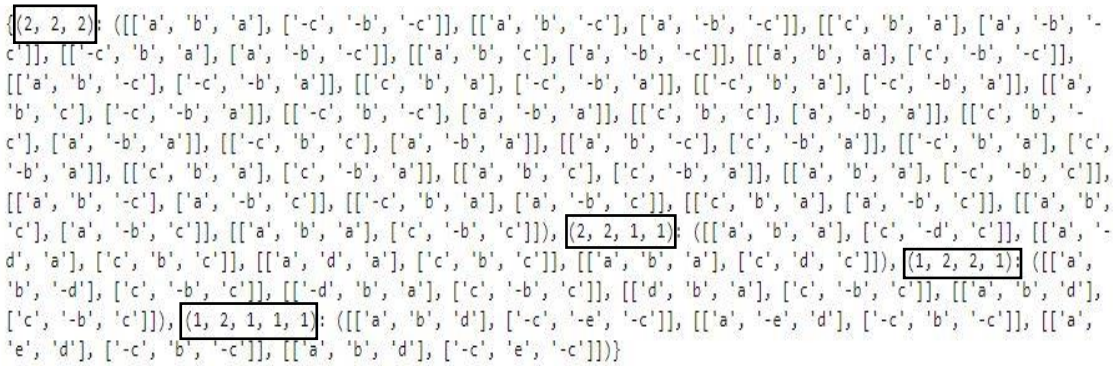

Figure 11: Structure of the dictionary

The keys of the dictionarys with marked, represent numbers associated with sequence of tuples in value part.

Then I reliazed that these 2 sequence of tuples in also equal according to lemmas of Orthogonal Designs.

[[b, a, b], [b, -a, a]] == [[a, b, a], [a, -b, b]]

Figure 12: Equivalence

So, I determine a structure in order to figure out the equivalence. The structure is as shown in the figure 13 and figure 14.

```
                [ [b, a, b], [b, -a, a] ]
{ 0: { '+' :   [ [0,1], [1,2] ]        ,   '-':  [ 1,1 ] }
  1: { '+':    [[0,0], [0,2], [1,0]]  ,   '-':  []  }
}
```

0 +	list1
0 -	list3
1+	list1
1-	list3

Figure 13: Structure of tuple T

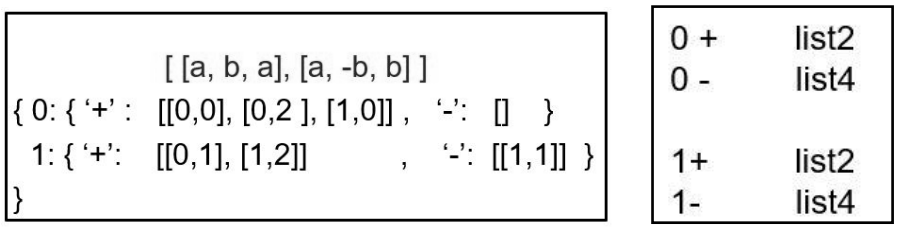

Figure 14: Structure of tuple T2

In structure of T and T2, There are 0 and 1 in the keys. This means tuples have 2 different symbol (such as 'a' and 'b'). Also there are '+' and '-' in the insider dictionary's key. This means every symbol in the tuple may have positive and negative symbol. The indices with 0 and + represent where positive symbols 'a' are. The indices with 0 and - represent where negative symbols 'a' are. I put all positive indices in list1. I put all positive indices of T in list 1, put all positive indices of T2 in list 2. I put all positive indices of T2 in list 2, put all positive indices of T2 in list 4.
In **findDifferentDict** method program determine the different sequence of tuples with the structure in figure 13 and figure 14.

```
def findDifferentDict(alltuples):
    temp=[]
    temp2=[]
    temp3=[]
    temp4=[]
    diffTuples=[]
    sayac1=0
    sayac2=0
    if(len(alltuples)!=0):
        diffTuples.append(alltuples[0])

        for i in alltuples:
            sameElement=0
            for j in diffTuples:
                if(len(i.keys())==len(j.keys())):
                    for insideDictionary in (i.values()):
                        #print(insideDictionary['+'])
                        temp.append(insideDictionary['+'])
                        temp3.append(insideDictionary['-'])
                    for insideTuples in (j.values()):
                        #print(insideTuples['+'])
                        temp2.append(insideDictionary['+'])
                        temp4.append(insideDictionary['-'])
                    if(temp.sort() == temp2.sort()):
                        sayac1=1
                    if(temp3.sort() == temp4.sort()):
                        sayac2=1
                    if(sayac1==1 and sayac2 ==1):
                            sameElement=sameElement+1

            if(sameElement==0):
                diffTuples.append(i)
            temp2=[]
            temp=[]
            temp3=[]
            temp4=[]
            sayac1=0
            sayac2=0
    return diffTuples
```

Figure 15: method for finding different sequence

In figure 15 we can see how method check the equality in findDifferentDict method, if all positive indices both T and T1 equal and if all negative indices both T and T1 then this means that two sequence of tuples are equal.

```
If ( sorted(list1) == sorted(list2) and
   sorted(list3) == sorted(list4) )
```

Figure 16: Checking Equality

I used sorted function in order to sort the indices in this findDifferentDict method. After using this method I decided to change the system instead of taking the tuples with symbols as a parameter, system take this dictionary structure with indices as a parameter. We can see the difference in figure 17.

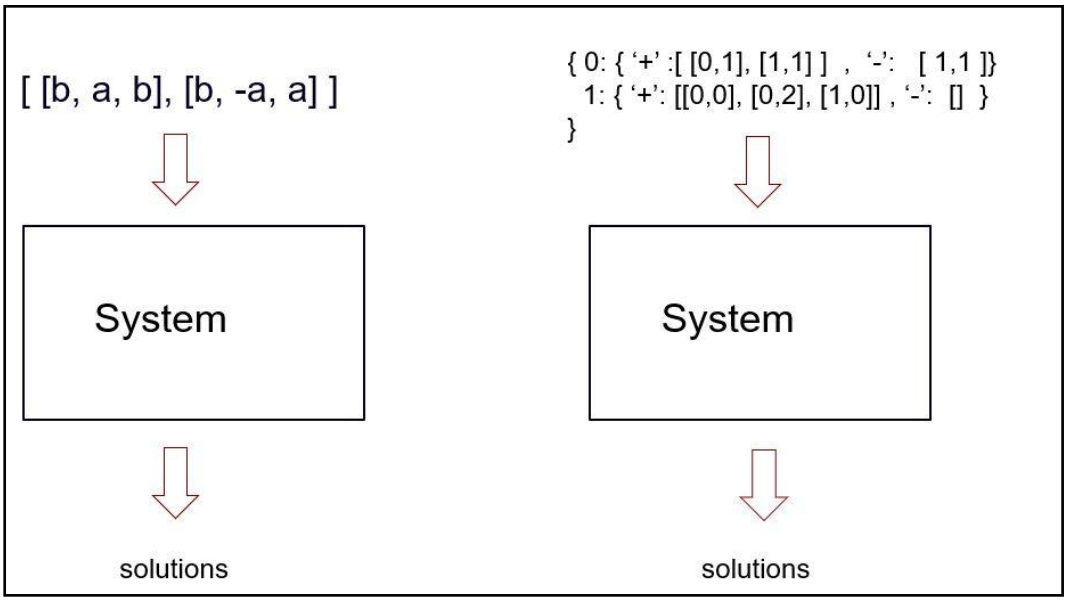

Figure 17: System changing

3.TEST RESULTS

The results obtained by giving different parameters to the system are as follows. If you enter the system with parameters larger than 3 × 2, the result's computation time is very long. Therefore, the parameters I can get results are as follows. All sequence of tuples in the screen images is only a certain part of all occurring sequences.

```
Parameter tuple=[['0', '2', '0'], ['1', '-2', '1']]

Different new sequence of tuples=
[['x0', 'x2', 'x0'], ['x1', '-x3', 'x1']]

[['x1', 'x2', '-x3'], ['x0', '-x4', 'x0']]

All sequence of tuples=

[['x1', 'x2', 'x1'], ['x0', '-x3', 'x0']]

[['x1', 'x2', 'x1'], ['x0', '-x3', 'x0']]

[['x1', '-x3', 'x1'], ['x0', 'x2', 'x0']]

[['x1', 'x3', 'x1'], ['x0', 'x2', 'x0']]

[['x1', '-x3', 'x1'], ['x0', 'x2', 'x0']]

[['x1', 'x3', 'x1'], ['x0', 'x2', 'x0']]

[['x1', 'x2', 'x1'], ['x0', 'x3', 'x0']]

[['x1', 'x2', 'x1'], ['x0', 'x3', 'x0']]

[['x2', 'x2', '-x4'], ['x1', '-x2', '-x3']]

[['x2', 'x2', '-x4'], ['x1', '-x2', '-x3']]

[['-x4', 'x2', 'x2'], ['x1', '-x2', '-x3']]

[['x4', 'x2', 'x2'], ['x1', '-x2', '-x3']]

[['-x4', 'x2', 'x2'], ['x1', '-x2', '-x3']]

[['x4', 'x2', 'x2'], ['x1', '-x2', '-x3']]

[['x2', 'x2', 'x4'], ['x1', '-x2', '-x3']]

[['x2', 'x2', 'x4'], ['x1', '-x2', '-x3']]

[['x2', 'x2', '-x4'], ['x1', '-x3', 'x1']]
```

Figure 18: Example 1

```
Parameter tuple=[['0', '2', '2'], ['2', '-2', '2']]

Different new sequence of tuples=
[['x0', 'x1', 'x1'], ['-x2', '-x2', '-x2']]

All sequence of tuples=

[['x0', 'x1', 'x1'], ['-x2', '-x2', '-x2']]

[['x0', 'x1', 'x1'], ['-x2', '-x2', '-x2']]

[['x0', 'x1', '-x2'], ['x1', '-x2', '-x2']]

[['x0', 'x1', 'x2'], ['x1', '-x2', '-x2']]

[['x0', 'x1', '-x2'], ['x1', '-x2', '-x2']]

[['x0', 'x1', 'x2'], ['x1', '-x2', '-x2']]

[['x0', '-x2', 'x1'], ['x2', '-x2', '-x2']]

[['x0', 'x1', 'x1'], ['x2', '-x2', '-x2']]

[['x0', 'x1', 'x1'], ['x2', '-x2', '-x2']]

[['x0', 'x2', 'x1'], ['x2', '-x2', '-x2']]

[['x0', '-x2', 'x1'], ['x2', '-x2', '-x2']]

[['x0', 'x1', 'x1'], ['x2', '-x2', '-x2']]

[['x0', 'x1', 'x1'], ['x2', '-x2', '-x2']]

[['x0', 'x2', 'x1'], ['x2', '-x2', '-x2']]

[['x0', 'x2', 'x1'], ['-x2', 'x1', '-x2']]

[['x0', '-x2', 'x1'], ['-x2', 'x1', '-x2']]

[['x0', '-x2', '-x2'], ['x1', 'x1', '-x2']]
```

Figure 19: Example 2

```
Parameter tuple=[['0', '2', '0'], ['1', '-2', '1']]

Different new sequence of tuples=
[['x1', 'x2', 'x1'], ['x0', '-x3', 'x0']]

[['x0', 'x2', 'x0'], ['x1', '-x4', '-x3']]

All sequence of tuples=

[['x1', 'x2', 'x1'], ['x0', '-x3', 'x0']]

[['x1', 'x2', 'x1'], ['x0', '-x3', 'x0']]

[['x1', '-x3', 'x1'], ['x0', 'x2', 'x0']]

[['x1', 'x3', 'x1'], ['x0', 'x2', 'x0']]

[['x1', '-x3', 'x1'], ['x0', 'x2', 'x0']]

[['x1', 'x3', 'x1'], ['x0', 'x2', 'x0']]

[['x1', 'x2', 'x1'], ['x0', 'x3', 'x0']]

[['x1', 'x2', 'x1'], ['x0', 'x3', 'x0']]

[['x2', 'x2', '-x4'], ['x1', '-x2', '-x3']]

[['x2', 'x2', '-x4'], ['x1', '-x2', '-x3']]

[['-x4', 'x2', 'x2'], ['x1', '-x2', '-x3']]

[['x4', 'x2', 'x2'], ['x1', '-x2', '-x3']]

[['-x4', 'x2', 'x2'], ['x1', '-x2', '-x3']]

[['x4', 'x2', 'x2'], ['x1', '-x2', '-x3']]

[['x2', 'x2', 'x4'], ['x1', '-x2', '-x3']]

[['x2', 'x2', 'x4'], ['x1', '-x2', '-x3']]
```

Figure 20: Example 3

```
Parameter tuple=[['1', '-2', '0'], ['0', '-2', '0']]

Different new sequence of tuples=
[['x0', 'x2', 'x1'], ['x1', '-x3', 'x1']]

[['x1', '-x3', 'x0'], ['x2', '-x3', '-x4']]

All sequence of tuples=

[['x0', 'x2', 'x1'], ['x1', '-x3', 'x1']]

[['x0', 'x2', 'x1'], ['x1', '-x3', 'x1']]

[['x0', '-x3', 'x1'], ['x1', 'x2', 'x1']]

[['x0', 'x3', 'x1'], ['x1', 'x2', 'x1']]

[['x0', '-x3', 'x1'], ['x1', 'x2', 'x1']]

[['x0', 'x3', 'x1'], ['x1', 'x2', 'x1']]

[['x0', 'x2', 'x1'], ['x1', 'x3', 'x1']]

[['x0', 'x2', 'x1'], ['x1', 'x3', 'x1']]

[['x1', 'x2', 'x2'], ['-x4', '-x3', '-x4']]

[['x1', 'x2', 'x2'], ['-x4', '-x3', '-x4']]

[['x1', 'x2', '-x4'], ['x2', '-x3', '-x4']]

[['x1', 'x2', 'x4'], ['x2', '-x3', '-x4']]

[['x1', 'x2', '-x4'], ['x2', '-x3', '-x4']]

[['x1', 'x2', 'x4'], ['x2', '-x3', '-x4']]

[['x1', 'x2', 'x2'], ['x4', '-x3', '-x4']]

[['x1', 'x2', 'x2'], ['x4', '-x3', '-x4']]
```

Figure 21: Example 4

4.CONCLUSION

In this study, it is aimed to produce a new sequence of tuples as more as possible for given a sequence of tuple according to the rules of orthogonal design and main goal of the project is to implement an algorithmic version for the reverse operations of Equating. Therefore, I refer to the reverse of Equating. To obtain new sequences I implemented Algebric system. When new sequences will be produced, the system create new variables according to solution that mentioned about Methods part. After system is solved there are a lot of solution. And method for splitting is done for each solution. After producing these sets of sequences can be used to generate desired Orthogonal Designs.

In this project, my success criterias is discovering new zero autocorrelation tuples and populating database at least two hundred and fifty tuples. The methods I used to implement the program, are explained in detail in Methods part. And all this duration I focused to find the different tuples so I've eliminated the same ones with using dictionary structure in order to determine the equivalence during process. The reason focusing these different sequence of tuples is for shorter duration and don't unnecessary calls when system make recursive calls to generate new sequences of tuples for each solution from each sequences. When the system is solved I have discovered new zero autocorrelation tuples.

RESOURCES

[1] Christos Koukouvinos , Dimitris E. Simos , Zafeirakis Zafeirakopoulos, *A Gro¨bner Bases Method for Complementary Sequences*, July 2nd-6th, 2013