

A SHORT DESCRIPTION OF THE ALGORITHMS USED FOR CIRCULANT HADAMARD MATRICES

GAURUSH HIRANANDANI AND JEAN-MARC SCHLENKER

ABSTRACT. We present a short description of the program used to search for circulant Hadamard matrices of Butson type for small values of n, l in the paper “Small circulant complex Hadamard matrices of Butson type”.

1. CIRCULANT.PY

This is the program that produces the list of possible first lines of circulant Hadamard matrices of Butson type for given (small) values of n and l .

We give here a short description of each function called by the main program, and then of the main program itself.

1.1. **factors()**. This function takes an integer and produces its distinct factors other than 1. The algorithm it follows is trivial.

1.2. **Pcheck()**. This takes input as the line to be checked and an empty line, the possible cycles for a number, an array of 1 and -1 to track any possibility, the length of the line to be checked, and the value of n . It returns a boolean value depending whether there exists a regular line or not from that partial line.

1.3. **Cycles()**. It produces the possible types of cycles for a given l value. The algorithm is trivial.

1.4. **Bcheck()**. It takes a partial line, l value, the set of cycles possible from a number, and n value. And it checks the corresponding *difference lines*, whether they are admissible or not by using *Pcheck()*. If all of them are admissible then it returns *True* else it returns *False*. The *difference lines* looks like, for example,

Suppose we are given a partial line P

$$[a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4]$$

The corresponding *difference lines* are;

$$[a_1 - a_0 \quad a_2 - a_1 \quad a_3 - a_2 \quad a_4 - a_3]$$

$$[a_2 - a_0 \quad a_3 - a_1 \quad a_4 - a_2]$$

$$[a_3 - a_0 \quad a_4 - a_1]$$

$$[a_4 - a_0]$$

1.5. **Least()**. It takes a partial line and l value. Then it checks that after bringing each element to the first index, whether it is least in Lexicographic order or not. The algorithm it follows is trivial.

If for all the elements present is least in the lexicographic sense, then it returns *True* else *False*.

Date: March 2015.

2000 Mathematics Subject Classification. 05B20.

Key words and phrases. Circulant Hadamard matrix.

1.6. **Full()**. It takes partial line and n value. It extends the line by appending *None* to it till its length becomes n .

1.7. **Orthocheck()**. It takes a line, set of possible cycles for all the numbers between 0 to $l-1$, and n value. Then it checks the orthogonality between lines by considering cyclic permutations of the fully extended lines. If they are, it returns *True*, else returns *False*.

1.8. **rotate()**. It rotates the list by given jumps. Here, l is the list and n is the jump.

1.9. **LeastLex()**. It takes a line and returns the least ordered (Lexicographically) line by considering only cyclic permutations. The Algorithm it follows is trivial and used *rotate()* function.

1.10. **CompleteC()**. It takes partial line, l value, set of possible cycles for all numbers between 0 and $l-1$, and n value. Its a recursive function that follows these steps.

- (1) If the length of the line is n then it returns it.
- (2) Else, it appends one value from 0 to $l-1$, into the line.
- (3) Then it checks, if *Bcheck()* is *True* or not.
- (4) If it is, then it checks *Least()* is *True* or not.
- (5) If it is, then it checks whether *Orthocheck()* is *True* or not.
- (6) If it is, then it sends this partial line for further extension, else not.
- (7) In the last it produces all the first lines for the *Circulant Butson Matrcies* without repetition.

In the *main* program, all the code should be self-evident, except for the fact that the program starts the computation with the partial line that has 0 in the beginning. The reason behind it is simple.

GH: SENIOR UNDERGRADUATE, DEPARTMENT OF MATHEMATICS AND STATISTICS, IIT KANPUR, KANPUR, UTTAR PRADESH, INDIA - 208016. gaurushh@iitk.ac.in

JMS: UNIVERSITY OF LUXEMBOURG, CAMPUS KIRCHBERG, MATHEMATICS RESEARCH UNIT, BLG, 6 RUE RICHARD COUDENHOVE-KALERGI, L-1359 LUXEMBOURG. jean-marc.schlenker@math.uni.lu