Obtaining Non-isomorphic Two-level Regular Fractional Factorial Designs

by

Chunfang Lin

B.Sc., 2002.

University of Science and Technology of China.

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

Statistics and Actuarial Science

SIMON FRASER UNIVERSITY

August 2004

APPROVAL

Name: Chunfang Lin

Degree: Master of Science

Title of project: Obtaining Non-isomorphic Two-level Regular Fractional Factorial Designs

Examining Committee: Dr. Michael A. Stephens
Chair

Dr. Randy R. Sitter
Senior Supervisor
Simon Fraser University

Dr. Derek Bingham
Simon Fraser University

Dr. Boxin Tang
External Examiner
Simon Fraser University

Date Approved: _____

Abstract

Two fractional factorial designs are isomorphic if one can be obtained from the other by reordering the treatment combinations, relabeling the factor levels and relabeling the factors. By defining a word-pattern matrix, we are able to create a new isomorphism check which is much faster than existing checks for certain situations. We combine this with a new, extremely fast, sufficient condition for non-isomorphism to avoid checking certain cases. We then create a faster search algorithm by combining the Bingham and Sitter (1999) search algorithm, the isomorphism check algorithm of Clark and Dean (2001) with our proposed isomorphism check. The algorithm is used to extend the known set of existing non-isomorphic 128-run designs to situations with 12, 13, 14, and 15 factors.

Acknowledgments

Firstly, thanks to my supervisor, Dr. Randy Sitter, for his continual guidance and support. Not only has he given me helpful suggestions and motivation on interesting research constantly but also helped me struggle with improving my English. I would also thank Dr. Derek Bingham and Dr. Boxin Tang for their many helpful suggestions and nice talks. To all my friends in the department of Statistics and Actuarial Science, thanks for the fun times and working hard together. Finally, my love and thanks go to my family, who have always made me their pride.

Contents

Abstract iii

Acknowledgements iv

Contents v

List of Tables vii

1 Introduction 1

2 Existing Methods for Checking Isomorphism 8

2.1 A Brief Review 8

2.2 Isomorphism Checking Algorithms 11

2.2.1 The Chen, Sun and Wu Algorithm 11

2.2.2 The Clark and Dean Algorithm 15

2.2.3 The Block and Mee Algorithm 16

2.3 Introduction of Existing Search Algorithms 17

2.3.1 The Franklin-Bailey Algorithm 17

2.3.2 The Chen, Sun and Wu Sequential Algorithm 19

2.3.3 The Bingham and Sitter Combined Algorithm 19

2.4 Summary and Discussion 20

3 Proposed Isomorphism Check and Algorithm 22

3.1 The Basic Idea 22

3.2 Hamming Distance Method When Using W Matrix 24

3.3 Eigenvalue and Eigenvector Criterion 27

3.4 Sequential Construction of Non-isomorphic 2^{k-p} Designs 29
3.5 128-run Resolution IV Designs 31
3.6 Discussion 32
4 Future Work 34
4.1 General 34
4.2 Our Proposal 35
Appendices
A Matrix for 128-run Design 37
B Minimum Aberration 128-run Designs With $k \leq 16$ Factors 39
C Fortran Program Isocheck.f90 42
Bibliography 62

List of Tables

1.1 Design Matrix and Data for Example 1. 2
1.2 Estimate of Effects and Alias Structure of Example 1. 4

2.3.1 Search table for 2^{6-2} designs 18
2.4.1 Number of Designs Entertained in Creating Catalogs of FF designs with 5 Basic Factors 21

3.5.1 Number of Non-isomorphic 2^{k-p} FF Resolution IV Designs with $n = 128$ 32
3.6.1 Time Spent when Using Design Matrix and W Matrix with $p \leq k - p$ for 128-run Designs 33

4.1.1 Number of Designs Entertained in Creating Catalogs of FF designs with 5 Basic Factors and Number of Corresponding Non-isomorphic Designs . 35
4.2.1 Time Using Proposed Algorithm and Eigenvalue Criterion Algorithm for 128-run Designs 36

Chapter 1

Introduction

Fractional factorial designs are commonly used in many areas of science and engineering. Suppose we wish to perform an experiment which considers k factors (variables), each at q levels. A full factorial would run the experiment at every possible combination of factor level settings (treatment or a run), i.e., q^k. For example, if $q = 2$ and $k = 3$, there are 2^3 possible factor level combinations as depicted in the first 3 columns of Table 1.1, where the '+' and '−' denote the high and low level of a factor, respectively, and each row of the first 3 columns represents a run at a possible factor level combination.

To carry out the experiment, a **design matrix** is used to describe the experimental plan by using standard notation for levels such as '+' and '−' (or '1', and '−1'). For a design with k factors and n runs, its $n \times k$ design matrix has n rows for the experimental runs and k columns for the factors.

Two important properties of the 2^k full factorial designs are **balance** and **orthogonality** (see Wu and Hamada (2000, p. 102), references therein). A design is **balanced** if each factor level appears in the same number of runs. Two factors are termed to be **orthogonal** if all their possible level combinations appear in the same number of runs. A design is orthogonal if all pairs of its factors are orthogonal. Consider the first three columns in Table 1.1. The 2^3 design is balanced since for each factor, the levels '+'

and '−' each appear in four runs. It is also orthogonal because each of the four level combinations (−, −), (−, +), (+, −) and (+, +) appears in two runs for each pair of factors.

To illustrate the concepts in the full factorial design, we consider an example given by Montgomery (2001 p. 308).

Example 1: A factorial experiment is carried out in a pilot plant to study the factors thought to influence the filtration rate of a chemical product which is produced in a pressure vessel, each at two levels:

A. Temperature

B. Pressure

C. Concentration of formaldehyde

D. Stirring rate.

Table 1.1: Design Matrix and Data for Example 1.

Run	A	B	C	D=ABC	Response(y_i)
1	−	−	−	−	45
2	+	−	−	+	100
3	−	+	−	+	45
4	+	+	−	−	65
5	−	−	+	+	75
6	+	−	+	−	60
7	−	+	+	−	80
8	+	+	+	+	96

Let y_i denote the response for the i-th run (last column of Table 1.1). To measure the average effect of a factor, for example A, compute the difference between the average of the y_i's at $A+$ (the high, +, level of factor A) and the average of the y_i's at $A-$ (the low, −, level of factor A). This difference is termed the **Main Effect (ME)** of A, i.e.,

$$ME(A) = \bar{y}(A+) - \bar{y}(A-).$$

The interaction effect of two factors A and B, AB, is defined as

$$INT(AB) = \frac{1}{2}(ME(B|A+) - ME(B|A-)) = \frac{1}{2}(ME(A|B+) - ME(A|B-)),$$

where ME$(B|A+)$ is the conditional ME of B at $A+$, ME$(B|A-)$ is the conditional ME of B at $A-$, ME$(A|B+)$ is the conditional ME of A at $B+$, ME$(A|B-)$ is the conditional ME of A at $B-$.

For Example 1, based on the given definition above, we can obtain

$$\begin{aligned}
ME(A) &= \frac{96+60+65+100}{4} - \frac{80+75+45+45}{4} = 19 \\
ME(B) &= \frac{96+80+65+45}{4} - \frac{60+75+100+45}{4} = 1.5 \\
ME(C) &= \frac{96+80+60+75}{4} - \frac{65+45+100+45}{4} = 14 \\
INT(AB) &= \frac{1}{2}(\frac{96+65}{2} - \frac{60+100}{2} - (\frac{80+45}{2} - \frac{75+45}{2})) = -1 \\
INT(AC) &= \frac{1}{2}(\frac{60+96}{2} - \frac{45+100}{2} - (\frac{80+75}{2} - \frac{45+45}{2})) = -13.5 \\
INT(BC) &= \frac{1}{2}(\frac{80+96}{2} - \frac{45+65}{2} - (\frac{60+75}{2} - \frac{45+100}{2})) = 19.
\end{aligned}$$

Running a full factorial design may be undesirable and/or too costly. Instead, one could run a fraction of the full factorial design, which is known as a fractional factorial (FF) design. One common way to do this is to assign the levels of p of the factors to the columns of the interactions of remaining columns from a full factorial with $k-p$ factors, denoted as a q^{k-p} design. For example, suppose that it is too expensive to conduct all 16 runs of the 2^4 full factorial design in Example 1. We could run a 2^{4-1} design with factor D assigned to the ABC interaction column of a 2^3 full factorial (see Table 1.1). If so, the column of D is used for estimating the ME of D and also for the interaction effect among A, B, and C. i.e., the data from such a design is not capable of distinguishing the estimate of D from the estimate of ABC. Therefore, the ME of D is said to be **aliased** with the ABC interaction. Notationally, this aliased relation or **defining relation** is denoted by

$$D = ABC \quad \text{or} \quad I = ABCD,$$

where I here refers to the overall mean, a column of +'s. Similarly, A is aliased with the BCD interaction, B is aliased with ACD interaction, C is aliased with ABD, AB is aliased with CD, etc. If two effects are aliased with each other, we cannot distinguish between their effects. However, according to the hierarchical ordering principle, that ME's are more likely to be important than 2-factor interactions (2fi), which are more likely to be important than 3-factor interactions (3fi), etc., the effect of 3fi's is often assumed negligible and the estimate is attributed to the ME or 2fi. For Example 1, the estimates of effects and alias structure are shown in Table 1.2.

Table 1.2: Estimate of Effects and Alias Structure of Example 1.

Estimate	Alias Structure
$l_A = 19$	$l_A \rightarrow A+BCD$
$l_B = 1.5$	$l_B \rightarrow B+ACD$
$l_C = 14$	$l_C \rightarrow C+ABD$
$l_D = 16.5$	$l_D \rightarrow D+ABC$
$l_{AB} = -1$	$l_{AB} \rightarrow AB+CD$
$l_{AC} = 13.5$	$l_{AC} \rightarrow AC+BD$
$l_{AD} = 19$	$l_{AD} \rightarrow AD+BC$

Those fractions defined through such defining relations are called **group-generated fractions**, and the corresponding designs are called **regular designs** since any two factorial effects either can be estimated independently of each other or are fully aliased. Otherwise, a design is termed a **non-regular design**, which includes many so-called **Plackett-Burman designs** and mixed-level orthogonal arrays. In this thesis, we only consider the orthogonal designs. Those group-generated fractions are determined by the p **generators** or defining **words**, which are all the columns that are equal to the identity column I. A word consists of **letters** which are labels of factors denoted by 1, 2, ..., k or A, B, ... The number of letters in a word is termed the **word length**. The **defining contrast subgroup** for the design consists of all the columns that are equal to the identity column I. These include the generators and interactions of the

generators. As an example, consider a 2^{7-2} design. Suppose we have I=$ABCF$ and I=$BCDEG$ as the generators. In this case, the defining contrast subgroup of the design is

$$I = ABCF = BCDEG = AEFG,$$

where $AEFG$ is obtained by multiplying $ABCF$ and $BCDEG$ such that any letters in common disappear. The vector

$$(A_1, \ldots, A_k) \tag{1.1}$$

is the **word-length pattern**, where A_i denotes the number of words of length i in the defining contrast subgroup.

Box and Hunter (1961) define the **resolution** of a fractional factorial (FF) design to be the smallest r such that $A_r \geq 1$ and argue that a design with highest resolution is better as it aliases ME's with highest-order interactions. Thus maximum resolution is a useful and convenient criterion for selecting practical designs in terms of the hierarchical ordering principle.

If two designs have the same resolution, they are not necessarily equivalent and we need a further criterion to characterize or discriminate FF designs. Fries and Hunter (1980) propose the following criterion. It has been a popular and most commonly used criterion to select a good design.

Minimum Aberration Criterion: For any two 2^{k-p} designs d_1 and d_2, let r be the smallest integer such that $A_r(d_1) \neq A_r(d_2)$. Then d_1 is said to have *less aberration* than d_2 if $A_r(d_1) < A_r(d_2)$. If there is no design with less aberration than d_1, then d_1 has *minimum aberration*(MA).

Example 2: The following two 2^{7-2} designs,

$$d_1 : I = ABCF = BCDEG = ADEFG$$

$$d_2 : I = ABCF = ADEG = BCDEFG,$$

both have resolution IV, but they have different word-length patterns,

$$d_1 : (0, 0, 0, 1, 2, 0, 0), \quad \text{and} \quad d_2 : (0, 0, 0, 2, 0, 1, 0).$$

Based on the MA criterion, the first design is a better design.

For a given k and p, a MA design always exists but is not always unique. Chen et al. (1993) suggest a competing criterion for design selection, the **number of clear effects**. A ME is **eligible** if it is not aliased with other ME's and **clear** if it is also not aliased with any 2fi's. This concept can also be extended to interactions of any order as follows. A q-factor interaction is eligible if it is not aliased with any effects of order less than q. An eligible q-factor interaction is clear if it is also not aliased with any other q-factor interaction. A ME or 2fi is **strongly clear** if none of its aliases are ME's, 2fi's, or 3fi's.

Clearly, we can derive the following important and useful rules from the definition of clear effects:

1. In any resolution IV design, the ME's are all clear.

2. In any resolution V design, the ME's are strongly clear and the 2fi's are clear.

3. Among the resolution IV designs with given k and p, those with the largest number of clear 2fi's are the best.

To further illustrate the criterion, let us consider two 2^{6-3} designs. The first design, d_1, has defining contrast subgroup

$$I = 1245 = 1236 = 3456$$

and the second design, d_2,

$$I = 136 = 1245 = 23456.$$

From the definition introduced above, we know that d_1 has resolution IV and six clear ME's but no clear 2fi's while d_2 has resolution III and only three clear ME's 2, 4, and

5 but six clear 2fi's 23, 26, 34, 35, 46, and 56. Since d_2 has only three clear ME's and ME's are usually more important than 2fi's based on the hierarchial ordering principle, one may say d_2 is inferior to d_1. On the other hand, d_2 has more clear effects than d_1. If in an investigation, only three factors and some of their two-factor interactions are believed to be important a priori, d_2 will be a preferred choice.

Chapter 2

Existing Methods for Checking Isomorphism

2.1 A Brief Review

A regular FF design is uniquely determined by its defining words and design matrix. Two designs are said to be isomorphic or equivalent if one can be obtained from the other by relabeling the factors having the same number of levels, reordering the treatment combinations and/or relabeling the levels of one or more factors. Otherwise, these two designs are non-isomorphic (non-equivalent). In other words, isomorphic designs can be transferred into each other by the usual randomization of factor labels and level labels. Since isomorphic designs share the same statistical properties in classical ANOVA models and essentially are the same, it is necessary to include only one of them in any catalogue of designs, or if possible to avoid considering more than one of them in any search for optimal designs and thus avoid unnecessary computations. To obtain a catalogue of designs, a straightforward approach does not work. For example, in a 2^{20-15} design, there are 5 independent factors and 15 additional factors, and yet they can be defined in $\binom{31-5}{20-5} = 7,726,160$ combinations. It is impractical to identify isomorphic designs among

all the 7,726,160 designs because of the computational difficulties involved in determining whether any two designs are isomorphic. The identification of the isomorphism of two designs is a vital combinatorial problem. For two k-factor (each having q levels) n-run designs, a complete search compares $n!k!(q!)^k$ designs based on the definition of isomorphism. It is known as an NP problem, when n and k increase.

The isomorphism of two regular FF designs has been discussed extensively in the literature. Draper and Mitchell (1968) develop a "sequential conjecture" method for testing the isomorphism of two designs. The method tests isomorphism by comparing the word-length patterns of designs. Unfortunately, it has since been determined that two designs could be non-isomorphic even though they have the same word-length pattern. For example, there exist two 2^{12-3} FF designs that have identical word-length patterns but are not isomorphic. Also, in Draper and Mitchell (1968)'s stage-by-stage procedure, a design which has the same word-length pattern as the one previously found would automatically be discarded, even if the two designs are not isomorphic. Therefore, the set of designs constructed using the word-length pattern comparison to test isomorphism is not necessarily a complete set of non-isomorphic designs of the specified type. Of course, the word-length pattern completely determines aberration and resolution, but for specific design situations, there are various other ways to rank designs. For example, the MA design may be far from the best design in terms of clear effects as we mention in Chapter 1.

Draper and Mitchell (1970) propose a more sensitive test for isomorphism using a "letter pattern comparison." Let a_{ij} be the number of words of length j in which letter i appears, then $A=(a_{ij})_{k\times k}$ is the **letter pattern** matrix of the design with k factors. They then declare two designs, d_1 and d_2, to be isomorphic if and only if $A_{d_1} = P(A_{d_2})$, where $P(A_{d_2})$ is some permutation of the rows of A_{d_2} and where A_{d_1} and A_{d_2} are the letter pattern matrices corresponding to d_1 and d_2, respectively. However, Chen and Lin (1990) show that this is not an isomorphism test by giving two non-isomorphic

2^{31-15} designs with identical letter pattern matrices. Thus the letter pattern does not uniquely determine a FF design either. Note that the use of letter pattern is a finer representation of a design than using the word-length pattern since the word-length pattern of a design can be written as $(\sum_{i=1}^{k} a_{i1},\ldots,\sum_{i=1}^{k} a_{ij}/j,\ldots,\sum_{i=1}^{k} a_{ik}/k\ )$, which implies that two designs having identical letter pattern matrices necessarily have the same word-length pattern.

Chen (1992) discusses the isomorphism of 2^{k-p} FF designs in terms of the existence of a relabeling map between two frequency vectors together with an appropriately defined matrix X. With the help of this frequency representation, Chen (1992) proves that any 2^{k-p} FF design with p=1 or 2 is uniquely determined by its word-length pattern and further proves that the word-length pattern uniquely determines any MA 2^{k-p} FF design when p=3 or 4. Chen et al. (1993) propose a sequential algorithm for constructing complete sets of FF designs by exploring the algebraic structure of the FF designs. A collection of FF designs with 16, 32, and 64 runs is given.

Clark and Dean (2001) present a method of determining isomorphism of any two factorial designs (non-regular as well as regular). Two designs are isomorphic if the factors can be relabeled so that the Hamming distance between a pair of corresponding points runs is the same for the two designs in all possible dimensions. The method gives a necessary and sufficient link between isomorphism and the Hamming distance matrices of two designs. They also provide an algorithm for checking the isomorphism of FF designs when all the factors have two levels which saves considerable time for detecting non-isomorphic designs.

Ma et al. (2001) propose a new algorithm based on the centered L_2-discrepancy(CD_2), a measure of uniformity, for detecting the isomorphism of FF designs and show it can significantly reduce the complexity of the computation. For two higher-level designs, they create a uniformity criterion for isomorphism (UCI). However, UCI is only a necessary condition for design isomorphism. They conjecture that UCI is also a sufficient

condition, but are unable to prove it thus far.

Sun et al. (2002) present an algorithm for sequentially constructing non-isomorphic regular and non-regular orthogonal designs. The algorithm is based on a minimal column base. They introduce an extended word-length pattern criterion, the definition of minimal column base, column base and its properties. Finally, they successfully obtain the complete catalog of orthogonal designs of 12, 16 and 20 runs.

Block and Mee (2004) present the results of an enumeration of n=128 run resolution IV designs. Rather than determining whether a new candidate design is isomorphic to the existing designs based on a complete permutation check, they retain all the designs that differ in their projections. Resolution IV designs are tabulated for k=12,. . . ,40 factors in 128-run designs. Since their criterion is not a sufficient and necessary condition, they still cannot claim that the designs that they provide are a complete non-isomorphic set.

2.2 Isomorphism Checking Algorithms

In this section, we examine three of the isomorphic checking algorithms in more detail. These are the Chen, Sun and Wu Algorithm (Chen et al., 1993), the Clark and Dean Algorithm (Clark and Dean, 2001), and the Block and Mee Algorithm (Block and Mee, 2004).

2.2.1 The Chen, Sun and Wu Algorithm

To define a 2^{k-p} FF design, Chen (1992) divides the k letters into 2^{p-1} subsets. Let f_i be the number of letters in the i-th subset such that $\sum_{i=1}^{2^p-1} f_i = k$, and let $\mathbf{f}$=(f_1, f_2,. . . , f_{2^p-1}) be called the frequency vector of the design. Chen (1992) constructs a matrix,

$$\mathbf{X} = \begin{pmatrix} I_p & B \\ B^t & B^t B \end{pmatrix}, \tag{2.1}$$

where I_p is a $p \times p$ identity matrix and B is a $p \times (2^p - p - 1)$ matrix which contains all the distinct and nonzero linear combinations (modulo 2) of column vectors of I_p. If V_i is a column vector of $\mathbf{X}$, $\langle V_i, \mathbf{f} \rangle$ equals the length of the i-th word. Chen (1992) suggests the following testing method with the help of this frequency representation (Chen's Theorems 5 and 6):

THEOREM 2.2.1. *Let* $\boldsymbol{f}=(f_1, \ldots, f_{2^p-1})^t$ *and* $\boldsymbol{g}=(g_1, \ldots, g_{2^p-1})^t$ *be two frequency vectors,* $\boldsymbol{X}$ *be given by* (2.1) *such that* ($\boldsymbol{X}$, $\boldsymbol{f}$) *and* ($\boldsymbol{X}$, $\boldsymbol{g}$) *are two* 2^{k-p} *FF designs. If there exists a relabeling map* ψ *for* $(1, 2, \ldots, 2^p - 1)$*, such that for any* i *and* j*,*

1. $\boldsymbol{f}_i = \boldsymbol{g}_{\psi(i)}$
2. $V_\psi(i) * V_\psi(j) = V_\psi(l), where \quad V_i * V_j = V_l$

where V_i*,* V_j *are row vectors of* $\boldsymbol{X}$ *and* $*$ *denotes the sum modulo 2, then* $\boldsymbol{f}$ *and* $\boldsymbol{g}$ *are equivalent, otherwise, they are not.*

THEOREM 2.2.2. *Any* 2^{k-p} *FF design with p=1 or 2 is uniquely determined by its word-length pattern.*

Consider two 2^{5-2} FF designs with defining contrast subgroups,

$$I = 124 = 135 = 2345$$

$$I = 124 = 1235 = 345.$$

Both designs have word-length pattern (0, 0, 2, 1). Furthermore, $A_{d_1} = P(A_{d_2})$, where A_{d_1} and A_{d_2} are the letter patterns of design d_1 and d_2, respectively. From the definition of frequency vector, we can easily obtain $\mathbf{f}$=(2, 2, 1), $\mathbf{g}$=(1, 2, 2), and

$$\mathbf{X} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix},$$

and thus V_1=(1, 0, 1), V_2=(0, 1, 1) and V_3=(1, 1, 0). Obviously, there exists a map ψ for (1, 2, 3). That is,

$1 = \psi(3), 2 = \psi(2), 3 = \psi(1)$
$f_1 = g_{\psi(1)}, f_2 = g_{\psi(2)}, f_3 = g_{\psi(3)}$
$V_{\psi(i)} * V_{\psi(j)} = V_{\psi(l)}, when \quad V_i * V_j = V_l$, for any i and j.

As a result, we conclude that design d_1 and design d_2 are isomorphic.

Subsequently, Chen et al. (1993) present a more detailed and executable algorithm to detect the isomorphism of two FF designs.

Each 2^{k-p} design has a design matrix. Thus it can be viewed as submatrices of regular Hadamard matrices. A *regular* Hadamard matrix of order 2^q (q=k-p) is a $2^q \times 2^q$ orthogonal matrix of ± 1's with the additional property that the entrywise product of any two columns is among the 2^q columns. By replacing -1 by 1 and 1 by 0 and using addition over GF(2), these 2^q columns form an elementary Abelian group over GF(2), where GF(2) is the Galois Field with two elements. Except for the column corresponding to the identity element in the group, we may write the remaining columns as

$$\mathbf{C} = \{C_1, \ldots, C_{2^q-1}\}. \tag{2.2}$$

Within $\mathbf{C}$, we can find q independent columns that generate all the columns in $\mathbf{C}$. A 2^{k-p} design can be viewed as a subset of $\mathbf{C}$ with k columns.

To identify isomorphic designs, Chen et al. (1993) divide all designs into different categories according to their word-length patterns and letter patterns. Designs with different word-length patterns or letter patterns are obviously non-isomorphic. Therefore,

they only need to check the isomorphism of designs with the same letter patterns. By applying the algorithm mentioned above, this can be done.

Their isomorphism check can be illustrated by a simple example. Suppose we have two 2^{7-3} designs with defining relation

$$d_1: \quad I = abe = abdf = bdcg$$

$$d_2: \quad I = ace = acdf = abcdg.$$

These two designs have the following three properties in common:

1. The set of C:
   { a, b, ab, c, ac, bc, abc, d, ad, bd, abd, cd, acd, bcd, abcd }
2. Word-length pattern: (0, 0, 2, 3, 2)
3. Letter pattern:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 0 & 0 \end{pmatrix}. \tag{2.3}$$

They apply the following algorithm to do the isomorphism check.

1. Select four independent columns from d_2, for example, $\{a, b, ac, abcd\}$, There are a total of $\binom{7}{4}$ choices.
2. Select a relabeling map from $\{a, b, ac, abcd\}$ to $\{A, B, C, D\}$, i.e., A=a, B=b, C=ac, D=$abcd$. There are $4! = 24$ choices.

3. Write the remaining columns $\{c, d, acd\}$ in d_2 as interaction of $\{A, B, C, D\}$, i.e., $c = AC$, $d = BCD$, $acd = BD$. Therefore, d_2 can be written as $\{A, B, C, D, AC, BCD, BD\}$.

4. Compare the new representation of d_2 with that of d_1. If they match, d_1 and d_2 are isomorphic and the process stops. Otherwise, go to step 2 and try another map of $\{A, B, C, D\}$. When all the relabeling maps are exhausted, go to step 1 and choose another 4 independent columns.

2.2.2 The Clark and Dean Algorithm

Let T_d be an $n \times k$ design matrix of a 2^{k-p} FF design, and define the Hamming distance matrix H_d to have (i, j)-th element

$$[H_d]_{i,j} = \begin{cases} \sum_{l=1}^{k} \delta[T_d]_{i,j}^{l} & i \neq j \\ 0 & i = j, \end{cases} \tag{2.4}$$

where $\delta[T_d]_{i,j}^{l}$ is equal to 1 if in the l-th column of T_d, the symbols in the i-th and j-th rows are different, and equal to zero if they are the same. The (i, j)-th element of H_d counts the number of dimensions in which the i-th and j-th points fail to coincide. The distance matrix H_d is invariant to the permutation of columns and relabeling of levels within columns of T_d.

THEOREM 2.2.3. *Designs d_1 and d_2 are isomorphic iff there exists an $n \times n$ permutation matrix R and a permutation $\{c_1, c_2, \ldots, c_k\}$ of $\{1, 2, \ldots, k\}$ such that, for every q=1,2,..., k, $H_{d_1}^{\{1,2,\ldots,q\}} = R(H_{d_2}^{\{c_1,c_2,\ldots,c_q\}})R'$.*

PROOF: Clark and Dean (2001).

When all the factors in the design have two levels, coded as -1 and 1,

$$H_d = (kJ_n - T_d T_d')/2,$$

where J_n is an $n \times n$ matrix of 1's. Let $T_d^{\{c_1,c_2,\ldots,c_q\}}$ be a matrix consisting of q columns $\{c_1, c_2, \ldots, c_q\}$ of T_d. Based on the theorem mentioned above, the following corollary can be obtained.

COROLLARY 2.2.1. *Designs d_1 and d_2 are isomorphic iff there exists an $n \times n$ permutation matrix R and a permutation $\{c_1, c_2, \ldots, c_k\}$ of $\{1, 2, \ldots, k\}$ such that, for every q=1,2,. . . , k,*

$$T_{d_1}^{\{1,2,\ldots,q\}} T_{d_1}^{\{1,2,\ldots,q\}\,T} = R(T_{d_2}^{\{c_1,c_2,\ldots,c_q\}} T_{d_2}^{\{c_1,c_2,\ldots,c_q\}\,T}) R^T.$$

Based on this corollary, Clark and Dean (2001) provide two FORTRAN programs (deseq1.f and deseq2.f) for identifying the isomorphism of any two 2-level designs. For two isomorphic 2-level designs, $T_{d_1} = RT_{d_2}CL$, where R and C are permutation matrices and L is a diagonal matrix with $L^2 = I$. Therefore, the first program (deseq1.f) does an initial check for non-isomorphism by checking whether for each q=1, 2, . . . , k, there is some subset $\{c_1, c_2, \ldots, c_q\}$ of $\{c_1, c_2, \ldots, c_k\}$ such that the rows of $H_{d_1}^{\{1,\ldots,q\}}$ and $H_{d_2}^{\{c_1,\ldots,c_q\}}$ contain the same set of distances with the same multiplicity. If there exists q=1, 2, . . ., k such that there is no subset of $\{c_1, c_2, \ldots, c_q\}$ of $\{c_1, c_2, \ldots, c_k\}$ making the rows of $H_{d_1}^{\{1,\ldots,q\}}$ and $H_{d_2}^{\{c_1,\ldots,c_q\}}$ contain the same set of distances with the same multiplicity, then designs d_1 and d_2 are non-isomorphic. However, two non-isomorphic designs could pass this initial test, thus requiring the second program to do a complete comparison. The second program (deseq2.f) looks through all row and column permutations that transform one design to the other. If we cannot find any row and column permutation, then the two designs are non-isomorphic.

2.2.3 The Block and Mee Algorithm

Block and Mee (2004) list all resolution IV designs for n=64 and show their projections. They observe that every 2_{IV}^{k-p} design of size 64 has a unique set of delete-one-factor

projections. As an example (given in their paper), a design with the generators G=ABC, H=ADE and J=$ABDF$, has nine factors and nine delete-one-factor projections:

- Design with word-length pattern=(0,2,1) if one deletes factor A
- Design with word-length pattern=(1,1,0,1) if one deletes factor B or D
- Design with word-length pattern=(1,2) if one deletes factor C, E, G, or H
- Design with word-length pattern=(2,0,1) if one deletes factor F or J.

Block and Mee (2004) conjecture that for resolution IV designs with n=128, if two designs have isomorphic delete-one-factor projection sets, then there exists a permutation of the columns and rows of one design to make it identical to the other, i.e., they are isomorphic. This test is more discriminating than word-length pattern and letter pattern. It is still a conjecture, however, and if the conjecture is false, they have not enumerated the complete catalogue of non-isomorphic designs with n=128.

2.3 Introduction of Existing Search Algorithms

In the following section, we introduce some existing search algorithms for obtaining collections of non-isomorphic two-level FF designs. These are the Franklin-Bailey Algorithm (Franklin and Bailey, 1977), the Chen, Sun and Wu Algorithm (Chen et al., 1993), the Bingham and Sitter Combined Algorithm (Bingham and Sitter, 1999).

2.3.1 The Franklin-Bailey Algorithm

Das (1964) defines $k-p$ of the factors as *basic factors* and the remaining p factors as *added factors* in a 2^{k-p} FF design. The group of size 2^{k-p} containing all the main effects and interactions among the $k-p$ basic factors is called *the basic effects group*. Originally a search algorithm uses the search-table data structure to identify designs that

Table 2.3.1: Search table for 2^{6-2} designs

	5	6
12	125	126
13	135	136
14	145	146
23	235	236
24	245	246
34	345	346
123	1235	1236
124	1245	1246
134	1345	1346
234	2345	2346
1234	12345	12346

allow estimation of a requirements set of factors and interactions given that all other interactions are negligible. The algorithm can be adapted to our problem as follows. *Step* 1. Construct a two-way search table. The table has $2^{k-p}-(k-p)-1$ rows, headed by the generalized interactions of the basic factors, and p columns, headed by the added factors. The elements of the table are the generalized interactions between the row and column headers. The rows are sorted by word-length. For example, for 2^{6-2}, the search table is given in Table 2.3.1. *Step* 2. Select a generator from the i-th column, where i=1. *Step* 3. For i=2,..., p , select a generator which is not in the same row of the search table as the previous columns. That is, because selecting generators from the same row of the search table results in designs with resolution less than III, which is usually not of interest, avoid such selections.

Franklin (1985) notes that the search table can be used to construct the set of non-isomorphic FF designs. All possible combinations of generators obtained by the above algorithm contain the set of all non-isomorphic FF designs and many isomorphic designs. By applying the isomorphism test to compare designs, this set of designs can be reduced to the set of all non-isomorphic FF designs.

2.3.2 The Chen, Sun and Wu Sequential Algorithm

To create a catalogue of MA FF's, Chen et al. (1993) present a sequential construction algorithm to get all non-isomorphic designs.

Let $D^r_{k,p}$ be the set of non-isomorphic 2^{k-p} designs with resolution $\geq r$ (for the remainder of the discussion, it is assumed that r=III and the r superscript is suppressed for convenience). Suppose we begin with D_{k_1,p_1}, the set of of all non-isomorphic $2^{k_1-p_1}$ FF designs, then D_{k_1+1,p_1+1} is constructed as follows. Assign the additional factor to one of the unused columns in each of the designs in D_{k_1,p_1}. Since there are $2^{k_1-p_1}-k_1-1$ unused columns available, there are at most $2^{k_1-p_1}-k_1-1$ ways to assign this factor. After removing designs with resolution less than III, we obtain a class of designs, denoted by $\widetilde{D}_{k_1+1,p_1+1}$. Obviously, $D_{k_1+1,p_1+1} \subset \widetilde{D}_{k_1+1,p_1+1}$ because $\widetilde{D}_{k_1+1,p_1+1}$ contains many isomorphic designs. Perform isomorphism checking to remove all isomorphic designs to get D_{k_1+1,p_1+1}. Similarly, D_{k_1+2,p_1+2} can be constructed. This procedure continues until all the non-isomorphic designs with k factors and p generators are obtained.

2.3.3 The Bingham and Sitter Combined Algorithm

Bingham and Sitter (1999) propose combining the ideas in the Franklin-Bailey algorithm and the Chen, Sun and Wu algorithm to find all non-isomorphic $2^{k_1+k_2-p_1-p_2}$ fractional factorial split-plot (FFSP) designs, with $2^{k_1-p_1}$ the *whole-plot* design and $2^{k_2-p_2}$ the *sub-plot* design. By setting k_2 and p_2 equal to 0, their algorithm can be used to search for all the non-isomorphic 2^{k-p} FF designs.

Let D_{k_1,p_1} be the set of non-isomorphic $2^{k_1-p_1}$ designs with resolution $\geq r$, create D_{k_1+1,p_1+1} by selecting all non-isomorphic generators in the first column of the search table. There are as many non-isomorphic generators as there are different lengths of generators in the first column of the search table by Theorem 2.2.2.

Next, create D_{k_1+2,p_2+2} by selecting a design in D_{k_1+1,p_1+1} and adding generators from the second column and compare each design chosen to those already in D_{k_1+2,p_1+2}

to see if it is isomorphic to any in D_{k_1+2,p_1+2}. If it is isomorphic to the designs already in D_{k_1+2,p_1+2}, the design is discarded; otherwise it is included into D_{k_1+2,p_1+2}. Bingham and Sitter (1999) prove that one need only consider the generators in the second column of the search table that are below the generators from the first column. The notion of adding factors to the design by considering generators below the fixed design generators also is applied for all added columns. This helps significantly reduce the number of designs considered with respect to other methods. They continue adding factors until they obtain D_{k_1,p_1}.

2.4 Summary and Discussion

1. **The Search Algorithms.** There are three existing search algorithms. These are the Franklin-Bailey search table algorithm (Franklin and Bailey, 1977), the Chen, Sun and Wu sequential algorithm (Chen et al., 1993) and the Bingham and Sitter combined algorithm (Bingham and Sitter, 1999). For a 2^{k-p} FF design, the Franklin-Bailey search table algorithm requires consideration of $N_1 = \prod_{i=1}^{p}(2^{k-p} - (k-p) - i)$ designs. The Chen, Sun and Wu sequential algorithm reduces the number of designs considered since it adds another factor based on the set of non-isomorphic designs. The combined algorithm significantly reduces the number of designs considered with respect to the other algorithms. For example, Table 2.4.1 (recreated from Bingham and Sitter, 1999), gives the number of designs considered by each algorithm for various n=32 FF designs.

2. **Isomorphism Checking.** So far, there are five proposals for checking isomorphism. These are Chen et al. (1993), Clark and Dean (2001), Ma et al (2001), Sun et al. (2002), and Block and Mee (2004). For n-run and k-factor designs, a complete search needs $n!k!2^k$ reordering and remodelings. Chen et al. (1993) reduce this to $\binom{k}{k-p}(k-p)!$ comparisons. Clark and Dean (2001) require $k(k!)^2$

Table 2.4.1: Number of Designs Entertained in Creating Catalogs of FF designs with 5 Basic Factors

algorithm	2^{7-2}	2^{8-3}	2^{9-4}	2^{10-5}
search	650	15,600	358,800	7,893,600
sequential	96	184	330	609
combined	45	89	273	282

comparisons in the worst case. Each comparison requires $O(n!)$ operations. Ma et al. (2001) require $O(n^2k2^k)$ to compare 2^{k+1} CD_2 values in the worst case. No similar information is available for Sun et al. (2002) and Block and Mee (2004).

3. **Discussion.** Chen et al. (1993) obtain the complete collection of 16-, 32- and 64-run FF designs by their proposed search algorithm and isomorphism checking algorithm. This is a complete catalog since their isomorphism check is iff. Block and Mee (2004) combine the Chen, Sun and Wu search algorithm and their proposed sufficient condition for non-isomorphism test. Thus, no one has combined the efficient iff isomorphism test of Clark and Dean (2001) with the most efficient search algorithm of Bingham and Sitter (1999). In the next chapter, we first introduce a new isomorphism check which is more efficient than Clark and Dean (2001) for some situations and then combine this with the Bingham and Sitter search algorithm and Clark and Dean isomorphism check.

Chapter 3

Proposed Isomorphism Check and Algorithm

3.1 The Basic Idea

A 2^{k-p} design is uniquely determined by its p defining relations. In other words, the defining contrast subgroup determines a 2^{k-p} design. Therefore, we focus on design isomorphism through its defining contrast subgroup. It turns out that this will yield some computational advantages in some cases.

Recall that for a 2^{k-p} design, the defining relation is

$$I = w_1 = w_2 = \ldots = w_{2^p-1}$$

and the word-length pattern is $(A_1, A_2, \ldots, A_k)$, where A_q is the number of words of length q. Let W_q be an A_q by k matrix with elements

$$w_{ij} = \begin{cases} 1, & \text{if the } j\text{-th letter shows up in the } i\text{-th word with length } q; \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

Thus, for a 2^{k-p} design with resolution r, it has the set of matrices W_r, W_{r+1}, $\ldots, W_m$, where m is the maximum non-zero word length. For instance, consider the

2^{7-3} design with resolution III and defining contrast sub-group

$$I = 125 = 136 = 2356 = 1237 = 357 = 267 = 1567.$$

The design has word-length pattern (0, 0, 4, 3, 0, 0, 0) and the set

$$W_3 = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

and

$$W_4 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Let W be formed by stacking all of the W_q's and called word-pattern matrix. In this example,

$$W = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Obviously, for a 2^{k-p} design, the W_q matrix always has k columns for every q and the sum of each row is equal to q.

3.2 Hamming Distance Method When Using W Matrix

Let $\delta[W]^l_{i,j}$=1 if in the l-th column of W the symbols in the i-th and j-th rows are different, and =0 if they are the same. Similar to Clark and Dean (2001), we define the distance matrix H but in our case for W instead of the design matrix. Let the (i, j)-th element of H be equal to

$$[H]_{i,j} = \sum_{l=1}^{k} \delta[W]^l_{i,j} \quad \text{for} \quad i \neq j \tag{3.2}$$

and equal to 0 if i=j. The distance matrix H is invariant to permutations of columns and to switching the role of 0 and 1 so that 1 represents the presence of a factor and 0 the absence, but not to the re-ordering of the rows (words). We can get a similar theorem to Clark and Dean (2001)'s Hamming distance matrix theorem. Before we state and prove Theorem 3.2.1 below, we introduce a useful lemma.

LEMMA 3.2.1. *Two 2^{k-p} FF designs d_1 and d_2 are isomorphic if $W_{d_1} = RW_{d_2}C$ for some permutation matrices R and C.*

PROOF: Follows obviously from the fact that W is uniquely determined by defining contrast subgroup.

Note, it is possible to have two isomorphic but not identical design matrices which yield the same W. Thus, by viewing W, we have implicitly eliminated some isomorphic designs.

As to how to find C technically, refer to Chen et al. (1993) (p.10) or section 2.2.1. Lemma 3.2.1 tells us that if a design d_1 can be obtained by another design d_2 by relabeling the factor labels in the defining contrast subgroup, then d_1 is isomorphic to d_2. With the help of Lemma 3.2.1 above, we are now able to state and prove Theorem 3.2.1 below.

THEOREM 3.2.1. *2^{k-p} designs d_1 and d_2 are isomorphic iff there exists an $s \times s$*

permutation matrix R and a permutation $\{c_1, c_2, \ldots, c_k\}$ of $\{1, 2, \ldots, k\}$ such that, for $l = 1, 2, \ldots, k$, $H_1^{\{1,2,\ldots,l\}} = R\{H_2^{\{c_1,c_2,\ldots,c_l\}}\}R'$, where $s = 2^p - 1$.

PROOF:

1. Suppose that design d_1 and d_2 are isomorphic. Then we can write $W_{d_1} = RW_{d_2}C$ or $W_{d_1}^{\{1,2,\ldots,k\}} = RW_{d_2}^{\{c_1,c_2,\ldots,c_k\}}$ by Lemma 3.2.1, where C is the permutation matrix corresponding to the permutation $\{c_1, c_2, \ldots, c_k\}$ that maps the factor labels of d_2 to those of d_1, and R is the permutation matrix that reorders the words in d_2 into the same order as those in d_1. Thus for $1 \leq q \leq k$,

$$[H_1^{\{q\}}]_{i,j} = \delta[W_{d_1}]_{i,j}^{\{q\}} = \delta[RW_{d_2}C]_{i,j}^{\{q\}} = \delta[RW_{d_2}]_{i,j}^{\{c_q\}} = [H_2^{\{c_q\}}]_{r_i,r_j}.$$

Therefore, for each $l = 1, 2, \ldots, k$,

$$\begin{aligned}
[H_1^{\{1,2,\ldots,l\}}]_{i,j} &= \sum_{q=1}^{l} [H_1^{\{q\}}]_{i,j} \\
&= \sum_{q=1}^{l} [H_2^{\{c_q\}}]_{r_i,r_j} \\
&= [\sum_{q=1}^{l} R[H_2^{\{c_q\}}]R']_{i,j} \\
&= [R\{H_2^{\{c_1,c_2,\ldots,c_l\}}\}R']_{i,j}. \qquad (3.3)
\end{aligned}$$

2. Let $\{c_1, c_2, \ldots, c_k\}$ be a fixed permutation of $\{1, 2, \ldots, k\}$ and let $H_2^{\{c_1,c_2,\ldots,c_l\}}$ be the distance matrix corresponding to the columns $\{c_1, c_2, \ldots, c_l\}$ of W_{d_2}, then $RH_2^{\{c_1,c_2,\ldots,c_l\}}R'$ is the distance matrix corresponding to the columns $\{c_1, c_2, \ldots, c_l\}$ of RW_{d_2} for some given permutation R since $[RH_2^{\{c_1,c_2,\ldots,c_l\}}R']_{i,j} = [H_2^{\{c_1,c_2,\ldots,c_l\}}]_{r_i,r_j} = \sum_{q=1}^{l} \delta[W_{d_2}]_{r_i,r_j}^{\{c_q\}} = \sum_{q=1}^{l} \delta[RW_{d_2}]_{i,j}^{\{c_q\}}$. For a given permutation matrix R, for any $l < k$,

$$
\begin{aligned}
[RH_2^{\{c_1,c_2,\ldots,c_l\}}R']_{i,j} &= \sum_{q=1}^{l-1} \delta[RW_{d_2}]_{i,j}^{\{c_q\}} + \delta[RW_{d_2}]_{i,j}^{\{c_l\}} \\
&= [RH_2^{\{c_1,c_2,\ldots,c_{l-1}\}}R']_{i,j} + [RH_2^{\{c_l\}}R']_{i,j}. \qquad (3.4)
\end{aligned}
$$

thus, for a fixed sequence of distance matrices $RH_2^{\{c_1,c_2,\ldots,c_l\}}R'$, $l = 1, 2, \ldots, k$, (3.4) implies a fixed sequence $RH_2^{\{c_l\}}R'$, $l = 1, 2, \ldots, k$, and we may consider each column of the word-pattern matrix separately. Let W_{d_1} be an $s \times k$ word-pattern matrix with first row the same as the first row of $W_{d_2}C$ ($\{c_1, c_2, \ldots, c_k\}$ of the first row of W_{d_2}). For each $l \in \{1, 2, \ldots, k\}$, we construct the l-th column of W_{d_1} as follows. For $i = 2, \ldots, s$, if $[RH_2^{\{c_l\}}R']_{i,j}$=0 for some $j = 1, 2, \ldots, i-1$, then set the symbol in the i-th row of column l of W_{d_1} to be the same as the symbol in the j-th row, i.e., $[W_{d_1}]_{\{i,l\}} = [W_{d_1}]_{\{j,l\}}$. Otherwise set $[W_{d_1}]_{\{i,l\}}$ equal to a different symbol. Remember the symbol here can only be 0 or 1. Thus, the l-th column of W_{d_1} is the same as the c_l-th column of RW_{d_2}. Therefore W_{d_1} is identical to W_{d_2} up to a permutation of rows and columns, which means that W_{d_1}=$RW_{d_2}C$, i.e., designs d_1 and d_2 are isomorphic.

COROLLARY 3.2.1. *2^{k-p} designs d_1 and d_2 are isomorphic iff there exists an $s \times s$ permutation matrix R and a permutation $\{c_1, c_2, \ldots, c_k\}$ of $\{1, 2, \ldots, k\}$ such that, for $l = 1, 2, \ldots, k$, $H_1^{\{l\}} = R(H_2^{\{c_l\}})R'$, where $s = 2^p - 1$.*

Recall that applying Clark and Dean (2001)'s method requires $k(k!)^2$ comparisons and each comparison requires $O(n!)$ operations in theory for the worst case. In our check, we only need $O(s!)$ operations. Therefore, if $s = 2^p - 1 < 2^{k-p} = n$, i.e., $p < k - p$, as n increases, a larger p can be used to satisfy this inequality so that we can take more advantage by using Theorem 3.2.1.

3.3 Eigenvalue and Eigenvector Criterion

To boost the speed to identify the non-isomorphism of two designs, we also introduce the eigenvalue and eigenvector of a matrix $Z_i = \widetilde{W}_i'\widetilde{W}_i$, where $\widetilde{W}_i$ is the subset of $\{W_r, W_{r+1}, \ldots, W_{k-p}\}$. For example, let $k-p$=5 and r=3, then $\widetilde{W}_i$ could be $W_3, W_4, W_5, \binom{W_3}{W_4}, \binom{W_3}{W_5}, \binom{W_4}{W_5}$, and

$$\begin{pmatrix} W_3 \\ W_4 \\ W_5 \end{pmatrix}.$$

and we let $\widetilde{W}_1 = W_3$, $\widetilde{W}_2 = W_4$, $\widetilde{W}_3 = W_5$, $\widetilde{W}_4 = \binom{W_3}{W_4}$ and so on.

Before proceeding to the main result given in Theorem 3.3.1 below, we introduce concepts in linear algebra and a useful lemma. Let V be a vector space over the field F and let T be an $n \times n$ matrix on V. An **eigenvalue** of T is a scalar c in F such that there is a non-zero vector α in V with $T\alpha = c\alpha$ and the matrix $T - cI$ is singular. If c is an eigenvalue of T, then any α such that $T\alpha = c\alpha$ is called an **eigenvector** of T associated with the eigenvalue c. Since c is an eigenvalue of T iff $det(cI - T) = 0$, we form the matrix $(xI - T)$ with polynomial entries, and consider the polynomial $f = det(xI - T)$. Clearly the eigenvalues of T are just the scalars c such that $f(c) = 0$. This f is called the **characteristic polynomial** of T. Two matrices A and B are **similar** if there exists an invertible matrix P such that $B = P^{-1}AP$.

LEMMA 3.3.1. *Similar matrices have the same characteristic polynomial.*

PROOF: If $B = P^{-1}AP$, then

$$\begin{aligned} det(xI - B) &= det(xI - P^{-1}AP) \\ &= det(P^{-1}(xI - A)P) \\ &= det(P^{-1}) \cdot det(xI - A) \cdot det(P) \\ &= det(xI - A). \end{aligned}$$

In other words, similar matrices have identical eigenvalues.

Now we are ready to state the main result of this section. Let $\Lambda(q,i) = (\lambda_{q,i,1}, \lambda_{q,i,2}, \ldots, \lambda_{q,i,k})$ be the vector of eigenvalues of Z_i of design q, where $i = 1, \ldots, 2^{k-p-r+1} - 1$, $q = 1, 2$, and $\lambda_{q,i,1} \geq \lambda_{q,i,2} \geq \ldots \geq \lambda_{q,i,k}$. Let $\Gamma(q,i) = (\nu_{q,i,1}, \nu_{q,i,2}, \ldots, \nu_{q,i,k})$ denote the corresponding matrix of eigenvectors, where $\nu_{q,i,j}$ is the corresponding eigenvector of $\lambda_{q,i,j}$ for $j = 1, 2, \ldots, k$.

THEOREM 3.3.1. *If two 2^{k-p} resolution r designs d_1 and d_2 are isomorphic, then there exists a $k \times k$ permutation matrix C such that, for all i= 1, 2, ..., ($2^{k-p-r+1} - 1$),*

(a). $\Lambda_{1,i} = \Lambda_{2,i}$

(b). $\Gamma_{1,i} = C'\Gamma_{2,i}$.

PROOF:
Let $\widetilde{W}_{1i}$ and $\widetilde{W}_{2i}$ represent the $\widetilde{W}_i$ matrices for designs d_1 and d_2, respectively. If d_1 and d_2 are isomorphic, then they have the same word-length patten and satisfy

$$\widetilde{W}_{1i} = R_i \widetilde{W}_{2i} C,$$

where R_i and C are row and column permutation matrices. Then clearly,

$$\begin{aligned} Z_{1i} &= \widetilde{W}'_{1i}\widetilde{W}_{1i} \\ &= (R_i\widetilde{W}_{2i}C)'R_i\widetilde{W}_{2i}C \\ &= C'\widetilde{W}'_{2i}\widetilde{W}_{2i}C \\ &= C^{-1}Z_{2i}C. \end{aligned} \tag{3.5}$$

(a) Follows from Lemma 3.3.1 since Z_{1i} and Z_{2i} are similar matrices. (b) Based on the definition of eigenvalues and eigenvectors and equation (3.5), we obtain $Z_{1i}\nu_{1,i,j} = C'Z_{2i}C\nu_{1,i,j} = \lambda_{1,i,j}\nu_{1,i,j}$ so that $Z_{2i}C\nu_{1,i,j} = \lambda_{1,i,j}C\nu_{1,i,j}$ since $Z_{2i}\nu_{2,i,j} = \lambda_{2,i,j}\nu_{2,i,j}$ and $\lambda_{1,i,j} = \lambda_{2,i,j}$. This means $C\nu_{1,i,j} = \nu_{2,i,j}$, i.e., $\Gamma_{1,i} = C'\Gamma_{2,i}$.

Theorem 3.3.1 only gives a condition for checking the isomorphism of two designs. However, it can be used to efficiently and straightforwardly detect the non-isomorphism of two designs. In other words, if two designs do not satisfy either (a) or (b) in Theorem 3.3.1, then two designs are non-isomorphic. If they satisfy (a) and (b), we must resort to further checking. Thus, it is similar in spirit to the Block and Mee check.

3.4 Sequential Construction of Non-isomorphic 2^{k-p} Designs

To construct a complete catalogue of non-isomorphic 2^{k-p} designs, we propose the following sequential approach which combines the search method proposed by Bingham and Sitter (1999) which extends the algorithms of Franklin and Bailey (1977) and Franklin (1985), the isomorphism check of Clark and Dean (2001), and the results of sections 3.1−3.3.

Proposed Algorithm:

1. Construct a search table, which has added factors as columns, and all possible interactions of basic factors as rows, and the interaction of corresponding row and column as elements. The elements are thus words. For example, for 2^{6-2}, the search table is given in Table 2.3.1.

2. For the first column, we pick up generators with different length as non-isomorphic designs. Thus the number of non-isomorphic designs for a 2^{k-1} design is identical to the number of designs with different word-length patterns by Theorem 2.2.2. Let $D_{k,p}$ be the set of all the non-isomorphic 2^{k-p} designs (with at least resolution III).

3. Assume that we have all the 2^{k-p} non-isomorphic designs, $D_{k-p+1,1}$. To construct $D_{k-p+2,2}$, we select one design from $D_{k-p+1,1}$ and add the generators from the next

column of the search table. Note we only pick up the generators in the next column of the search table that are below the generator from the current column. Compare each successive design chosen to those already in $D_{k-p+2,2}$ to determine whether it is isomorphic to an already obtained design. We do the following isomorphic check for each two selected designs.

(a) Compute the word-length patterns. Compare them, if they are different, these two designs are non-isomorphic, otherwise go to (b).

(b) Compute the vector of eigenvalues of $\widetilde{W}_i'\widetilde{W}_i$. If any two vectors are different, these two designs are non-isomorphic, otherwise go to (c).

(c) If $s = 2^p - 1 < 2^k - p = n$, i.e., $p \leq k - p$, use the subroutine '*wcheck*' in the FORTRAN program isocheck.f90 in Appendix C which implements Theorem 3.2.1, otherwise use the subroutine '*deseq*2' which is the second program (deseq2.f) of Clark and Dean (2001), if the row permutation and column permutation are found, these two designs are isomorphic, otherwise, they are non-isomorphic.

If the design is isomorphic to some design in the $D_{k-p+2,2}$, discard it; otherwise, this design is added to $D_{k-p+2,2}$.

4. Repeat Step 3 to construct the complete set of all non-isomorphic designs $D_{k-p+q,q}$, $q = 3, 4, \ldots$.

A FORTRAN program is given in Appendix C and the following is an example to illustrate the algorithm.

Example 3: 16-run FF Designs

Firstly, based on the algorithm mentioned above, we can easily get $D_{5,1}$ ={(125), (1235), (12345)}. Secondly, to get $D_{6,2}$, we select a generator from $D_{5,1}$ and consider generators from the second column that are below it in Table 2.3.1. For example, we select the

generator w_1=125, and consider the first generator in the second column, w_2=136. Since $D_{6,2}$ is empty, the design should be added to $D_{6,2}$. We then consider the second generator in the second column 146, however $\{(125, 146)\}$ and $\{(125, 136)\}$ are isomorphic. Similarly, we find that $\{(125, 236)\}$, $\{(125, 246)\}$ are isomorphic to $\{(125, 136)\}$. When we consider $(125, 346)$, we find that this design has different word-length pattern from $\{(125, 136)\}$. Thus $D_{6,2}$ is expanded to $\{(125, 136), (125, 346)\}$. Next, we pick the design $\{(125, 1346)\}$, and compare it to (125, 136). Since they are non-isomorphic, we continue and compare it to (125, 346) and find they are also non-isomorphic. Therefore, $D_{6,2}$ becomes $\{(125, 136), (125, 346), (125, 1346)\}$. The same steps continue until we reach the last design (1235, 12346). After completing this search and comparison in the second column, the algorithm stops and $D_{6,2}$ =$\{(125, 136), (125, 346), (125, 1346), (1235, 1246)\}$. The FORTRAN program continues through the columns until we get $D_{15,11}$.

3.5 128-run Resolution IV Designs

So far, the appended FORTRAN program has helped us obtain the complete catalogue of 128-run designs with up to 15 factors, Table 3.5.1 lists the number of possible non-isomorphic FF 128-run resolution IV designs with $k \leq 15$ factors. A collection of these designs are available. The numbers in Table 3.5.1 do correspond to those given in Block and Mee (2004), where they conjecture that they are the complete set of non-isomorphic designs, thus supporting their claim up to designs with 15 factors. For $k \geq 12$ factors, FF designs of resolution V and more do not exist. In Appendix B, we list the best ten designs based on MA criterion for $k \leq 15$, where the columns of each design are columns of the matrix in Appendix A used for the generators. For example, $k = 9$, the best design has columns 31 and 103. In Appendix A, we can find the 31-th column corresponds to generator 12345, since rows 1, 2, 3, 4 and 5 of column 31 contain a 1. Similarly, the 103-th column is 12367.

Table 3.5.1: Number of Non-isomorphic 2^{k-p} FF Resolution IV Designs with $n = 128$

k	No. of Designs
8	5
9	13
10	33
11	92
12	249
13	623
14	1535
15	3522

3.6 Discussion

Comparing two isomorphic designs is computer intensive but the really bad situation is two non-isomorphic designs which pass the non-isomorphic checks. The reason is that, if two designs are isomorphic we merely need to consider row and column permutations until we get one from the other, but if two designs are non-isomorphic and pass all the non-isomorphism tests, we need to try all possible row and column permutations to be sure that they are not isomorphic. Our proposed isomorphism check algorithm can identify all non-isomorphic designs up to 15 factors since the non-isomorphic FF designs with $k \leq 15$ have different vectors of eigenvalues for some $\widetilde{W_i}$ matrix, thus helping a great deal in the real search and comparison.

We only prove that the eigenvalue and eigenvector criterion is a necessary condition to detect the isomorphism between two designs, which is the reason why we still use subroutine 'deseq2' or 'wcheck' (try all possible relabelings of rows and columns in both subroutines) to do a complete identification. On the other hand, our proposed algorithm is beneficial for $p \leq k - p$ thanks to using the W matrix instead of the design matrix. This can be supported by Table 3.6.1, where s, m, h and d represent seconds, minutes, hours and days, respectively. The W matrix is $(2^p - 1) \times k$, thus its dimension changes

with the number of factors instead of run size. In other words, it may be efficient to get non-isomorphic 256 or bigger run FF designs by our proposed algorithm.

Table 3.6.1: Time Spent when Using Design Matrix and W Matrix with $p \leq k - p$ for 128-run Designs

matrix	2^{9-2}	2^{10-3}	2^{11-4}	2^{12-5}	2^{13-6}
Design matrix	2m46s	29m	12h	16h	1d10h
W matrix	0.57s	4.7s	36s	5m3s	40m

Chapter 4

Future Work

The isomorphism of regular FF designs has been extensively discussed in the literature (section 2.1). The most efficient search algorithm is given by Bingham and Sitter (1999). However, there are a number of other considerations that could be made.

4.1 General

Firstly, the ratio of the number of the non-isomorphic designs to the number of designs considered in the most efficient search algorithm is still small. For example, Table 4.1.1 lists the number of designs entertained in creating catalogs of FF designs with 5 basic factors and the number of corresponding non-isomorphic designs. The percent is the ratio of number of the non-isomorphic designs and number of designs entertained in the combined Bingham and Sitter search algorithm. We can see all the percents are less than 20%. Therefore, an interesting work is to reduce the number of designs considered in search algorithm.

Secondly, the speed of detecting the isomorphism of two designs may still have potential room to improve. Therefore, if either the search algorithm or the isomorphism check can be made more efficient, one could combine the most efficient search algorithm

Table 4.1.1: Number of Designs Entertained in Creating Catalogs of FF designs with 5 Basic Factors and Number of Corresponding Non-isomorphic Designs

algorithm	2^{7-2}	2^{8-3}	2^{9-4}	2^{10-5}
combined	45	89	273	282
non-isomorphic	8	15	29	46
percent	17.78	18.85	10.62	16.31

and isomorphism check to obtain the catalogue of FF designs.

4.2 Our Proposal

We may apply this eigenvalue and eigenvector criterion to q-level ($q \geq 3$) and mixed-level FF, FFSP or non-regular designs. The criterion may have greater benefits in higher-level or mixed-level designs.

From the catalogue of all non-isomorphic designs, one can show that the word-length pattern $(A_1, A_2, \ldots, A_k)$ of a 2^{k-p} design with resolution r is uniquely determined by $A_r, A_{r+1}, \ldots, A_{k-p}$, when k-p=3, 4, 5, 6. For k-p=7, it is also true for the obtainable non-isomorphic designs. This fact tells us that to check the word-length pattern of two designs we only need to check the first $k-p$ word-lengths for 8, 16, 32, 64-run regular designs. For $p > k-p$, we actually do not need to write down the complete defining contrast subgroup. It is enough to write down the generators and the at most up to the $(k-p)$-th interactions of the generators in order to compare the word-length pattern of two designs. We conjecture that this is also true for k-$p \geq 7$.

Furthermore, we conjecture that the algorithm proposed in section 3.4 excluding part (c) of step 3 is sufficient to obtain the catalogue of non-isomorphic two-level FF designs. Thus it would only implement the word-length pattern comparison and eigenvalue criterion and would not look for any row, column and level permutations. Therefore, we call it the eigenvalue criterion algorithm (E.C.A in Table 4.2.1). We obtain the same

Table 4.2.1: Time Using Proposed Algorithm and Eigenvalue Criterion Algorithm for 128-run Designs

algorithm	2^{9-2}	2^{10-3}	2^{11-4}	2^{12-5}	2^{13-6}	2^{14-7}	2^{15-8}	2^{16-9}
P.A.	0.57s	4.7s	36s	5m3s	40m	10d	40d	–
E.C.A.	0.44s	2.5s	27s	5ms	22m42s	2h18m	14h18m	118h

number of non-isomorphic 128-run designs with $k \leq 16$ as that given by Block and Mee (2004). The reason we stop at $k = 16$ is not because of the speed of isomorphism check but rather because there are a huge number of non-isomorphic 128-run designs. Table 4.2.1 lists the time using the proposed algorithm (P.A. in Table 4.2.1) in section 3.4 and the eigenvalue criterion algorithm. S, m, h, and d in Table 4.2.1 represent seconds, minutes, hours and days, respectively. One can see the speed of E.C.A is about 1000 times faster than that of P.A. Almost certainly, the time can be further reduced by at least 10 times via astute programming since we have not taken full advantage of the Fortran programming language and faster computer facilities. Since the non-isomorphic designs are a small part of the designs considered in creating the catalogue of FF designs and the percentage of the number of non-isomorphic designs will be smaller as the number of factors increases, E.C.A will have greater benefit in creating the catalogue of FF designs with more factors.

In the proposed algorithm, we compute the vector of eigenvalues of all $\widetilde{W}_i'\widetilde{W}_i$, where $\widetilde{W}_i$'s are the subset of $\{W_r, W_{r+1}, \ldots, W_{k-p}\}$. It will be interesting to think about whether the vector of eigenvalues of $\widetilde{W}_s'\widetilde{W}_s$ may determine the vector of eigenvalues of $\widetilde{W}_c'\widetilde{W}_c$, where $\widetilde{W}_s$'s are some $\widetilde{W}_i$ and $\widetilde{W}_c$'s are the remaining subsets. In other words, is it possible that if the vectors of eigenvalues of $\widetilde{W}_s'\widetilde{W}_s$ for two designs are the same, the corresponding vectors of eigenvalues of $\widetilde{W}_c'\widetilde{W}_c$ will be the same? Exploring the existence and components of $\widetilde{W}s$ seems worthwhile.

Appendix A

Matrix for 128-run Design

1	**2**	3	**4**	5	6	7	**8**	9	10	11	12	13	14	15	**16**	17	18	19	20	21	22	23	24	25
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

26	27	28	29	30	31	**32**	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

51	52	53	54	55	56	57	58	59	60	61	62	63	**64**	65	66	67	68	69	70	71	72	73	74	75
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

NOTE: The independent columns are in boldface and numbered 1, 2, 4, 8, 16, 32, 64.

76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

116	117	118	119	120	121	122	123	124	125	126	127
0	1	0	1	0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1	0	0	1	1
1	1	1	1	0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

Appendix B

Minimum Aberration 128-run Designs With $k \leq 16$ Factors

NOTE: The MA 2^{16-9} designs listed below are obtained from Eigenvalue Criterion Algorithm (see section 4.2).

k	*Columns*	$(A_4, A_5, \ldots)$
8	127	0 0 0 0 1
8	63	0 0 0 1 0
8	31	0 0 1 0 0
8	15	0 1 0 0 0
8	7	1 0 0 0 0
9	31 103	0 0 3 0 0 0
9	15 115	0 1 1 1 0 0
9	15 113	0 2 0 0 1 0
9	15 51	0 2 1 0 0 0
9	7 123	1 0 0 2 0 0
9	7 121	1 0 1 0 1 0
9	7 59	1 0 2 0 0 0
9	7 120	1 1 0 0 0 1
9	7 57	1 1 0 1 0 0
9	7 27	1 2 0 0 0 0

k	*Columns*	$(A_4, A_5, \ldots)$
10	15 51 85	0 3 3 1 0 0 0
10	15 51 92	0 4 2 0 1 0 0
10	7 59 93	1 0 6 0 0 0 0
10	7 57 90	1 2 2 2 0 0 0
10	7 27 109	1 2 2 2 0 0 0
10	7 27 105	1 3 1 1 1 0 0
10	7 27 120	1 3 2 0 0 1 0
10	7 27 101	1 4 0 0 2 0 0
10	7 27 99	1 4 1 0 0 0 1
11	15 51 85 106	0 6 6 2 1 0 0 0
11	7 57 90 108	1 4 6 4 0 0 0 0
11	7 27 45 120	1 5 6 2 0 1 0 0
11	7 27 45 113	1 6 4 2 2 0 0 0
11	7 27 99 45	1 6 5 2 0 0 1 0
11	7 27 45 78	1 6 6 1 0 0 0 1
11	7 27 45 86	1 7 4 0 2 1 0 0
11	7 56 91 109	2 0 12 0 1 0 0 0
11	7 56 75 85	2 4 4 4 1 0 0 0
11	7 56 27 109	2 4 4 4 1 0 0 0
11	7 25 43 116	2 4 4 4 1 0 0 0
12	7 57 90 108 119	1 8 12 8 1 0 0 0 1
12	7 27 45 113 78	1 10 10 5 4 0 0 1 0
12	7 27 99 45 86	1 10 11 4 2 2 1 0 0
12	7 56 75 85 102	2 7 12 7 1 1 0 1 0
12	7 56 75 85 110	2 8 10 8 1 0 2 0 0
12	7 56 27 107 93	2 8 12 4 1 4 0 0 0
12	7 25 43 83 101	2 10 8 4 5 2 0 0 0
12	7 25 43 85 102	2 10 8 4 5 2 0 0 0
12	7 56 27 77 102	2 10 10 4 1 2 2 0 0
12	7 56 27 45 115	2 11 8 3 5 1 0 1 0

k	*Columns*	$(A_4, A_5, \ldots)$
13	7 56 27 77 102 115	2 16 18 10 9 4 2 2 0
13	7 56 27 45 107 117	2 16 20 8 5 8 4 0 0 0
13	7 56 27 107 93 118	3 12 24 8 3 12 0 0 1
13	7 25 98 45 85 126	3 14 17 14 8 2 3 2 0
13	7 25 42 83 101 127	3 14 18 12 7 6 2 0 1
13	7 25 43 75 101 118	3 15 17 11 8 5 3 1 0
13	7 25 42 83 53 124	3 15 17 11 8 5 3 1 0
13	7 25 98 43 83 125	3 16 15 10 12 4 1 2 0
13	7 56 27 45 107 93	3 16 16 8 11 8 0 0 1
13	7 25 43 75 101 115	3 17 15 7 12 7 1 1 0
14	7 56 27 45 107 117 94	3 24 36 11 24 12 0 1 0
14	7 25 42 83 53 124 111	4 24 30 20 19 16 10 4 0
14	7 25 97 42 84 78 114	5 22 30 22 17 18 10 2 1
14	7 25 97 42 83 77 63	5 23 27 23 22 13 9 5 0
14	7 25 42 84 75 54 108	5 23 27 23 22 13 9 5 0
14	7 25 42 83 53 101 127	5 24 26 20 25 16 6 4 1
14	7 25 42 84 99 78 60	5 24 26 20 25 16 6 4 1
14	7 11 49 83 45 78 124	5 24 26 20 25 16 6 4 1
14	7 11 49 29 101 94 123	5 24 26 20 25 16 6 4 1
14	7 11 29 45 113 78 127	5 24 28 16 23 24 4 0 3
15	7 25 97 42 84 78 114 127	7 32 52 40 35 48 28 8 5 0
15	7 25 97 42 84 78 114 63	7 34 46 42 45 38 26 14 3 0
15	7 11 49 83 45 78 60 93	7 38 44 28 51 56 20 4 5 2
15	7 25 97 42 82 77 54 61	8 31 50 42 35 48 30 6 4 1
15	7 25 97 42 84 78 63 125	8 32 44 48 45 32 28 16 2 0
15	7 25 97 42 82 77 94 123	8 32 49 40 37 48 30 8 2 0 1
15	7 11 49 81 99 29 109 127	8 32 49 40 37 48 30 8 2 0 1
15	7 25 97 42 82 77 60 123	8 33 44 44 45 38 28 12 2 1 0
15	7 11 49 82 45 77 116 110	8 33 44 44 45 38 28 12 2 1 0
15	7 11 49 82 29 101 46 127	8 33 44 44 45 38 28 12 2 1 0
15	7 11 49 82 29 101 46 123	8 33 44 44 45 38 28 12 2 1 0
15	7 11 21 41 51 101 122 95	8 33 44 44 45 38 28 12 2 1 0
16	7 25 97 42 84 78 114 63 125	10 48 72 80 90 80 72 48 10 0 0 0 1
16	7 25 97 42 82 77 54 61 117	11 44 82 72 71 112 76 24 13 4 2 0
16	7 11 49 82 100 29 46 92 127	11 47 72 77 87 94 72 34 13 3 0 1 0
16	7 11 49 82 100 29 105 46 119	11 48 70 76 91 92 72 36 9 4 2 0
16	7 11 49 81 99 29 46 116 109	11 50 66 72 103 92 60 40 13 2 2 0 0
16	7 11 49 84 104 29 45 102 90	11 50 68 68 99 104 60 28 17 6 0
16	7 11 49 84 104 29 102 90 63	11 52 66 64 103 104 60 32 13 4 2 0
16	7 11 21 56 83 77 46 54 94	11 56 66 48 103 128 60 16 13 8 2 0
16	7 25 97 42 82 77 62 118 125	12 40 80 88 70 88 80 40 12 0 0 0 1
16	7 11 21 97 56 83 46 93 126	12 46 68 81 93 88 68 38 14 2 0 1 0
16	7 11 49 82 100 29 45 106 124	12 46 68 81 93 88 68 38 14 2 0 1 0
16	7 11 21 41 100 51 78 86 120	12 46 68 81 93 88 68 38 14 2 0 1 0

Appendix C

Fortran Program Isocheck.f90

```
    PROGRAM alldesign
!-----------------------------------------------------------------------
!get the all the non-isomorphic design for the two level
!diffdesign : save the non-isomorphic designs for add k-th step
!diffmatrix:ith column is the position of non-isomorphic design 's basic
!effect for ith step
!idiff : the number of old non-isomorphic designs
!new_idiff : the number of new non-isomorphic designs
!MM : the number of basic factors
!KK : the number of added factors
!nn:  the number of runs
!lenvec: the cumulative number of words for i-length
!new_ii is from 1 to new_idiff
!design2 is a design from the exiting non-isomorphic design
!pick up a new design from the search table,denoted as design1
!-----------------------------------------------------------------
INTEGER::jstep,idiff,new_ii,new_idiff,nn,ii,jii,ifile,p,infor
INTEGER::infor2,infor3,infor4
LOGICAL::diff,first
CHARACTER(20)::filenames
CHARACTER*10::b(3)
INTEGER,PARAMETER::MM=7,diffrows=3530,kcols=20,ncol=260
INTEGER,PARAMETER::lwork=130
INTEGER:: KK,nrow,jjii,k,i,j
INTEGER,DIMENSION(8)::date_time
INTEGER,DIMENSION(MM-1)::lenvec
INTEGER,DIMENSION(kcols)::addeff
INTEGER,DIMENSION(5*kcols):: basiceff
REAL,DIMENSION(lwork)::work,work2,work3,work4
INTEGER,DIMENSION(2,kcols)::design1,design2
INTEGER,DIMENSION(diffrows,kcols)::diffmatrix
INTEGER,DIMENSION(diffrows,2,kcols)::diffdesign,new_diffdesign
INTEGER,DIMENSION(2,kcols)::wlp
REAL,DIMENSION(2,kcols)::eigenval,eigenval2,eigenvalc
```

```
REAL,DIMENSION(2,kcols)::eigenval22,eigenval32,eigenval42
INTEGER,DIMENSION(2,ncol,kcols)::wmatrix
INTEGER,DIMENSION(2,ncol,kcols)::pparr,pparr2,pparr3,pparr4,cpparr
INTEGER,DIMENSION(2,kcols,ncol)::tpp,tpp2,tpp3,tpp4,tcpp
INTEGER,DIMENSION(2,kcols,kcols)::tpppp,tpppp2,tpppp3,tpppp4,tcpppp

  OPEN(150,file="searchtable128.txt")
  nn=2**MM
  OPEN(1,file="design128_1.out")
  READ(150,*)(lenvec(i),i=1,MM-2)
  DO i=1,lenvec(MM-2)
    READ(150,*)basiceff(i)
  END DO
  DO i=1,kcols
  addeff(i)=MM+i
  END DO
!the first step is to get the non-isomorphic 28-1  designs
  DO i=1,MM-2
    IF(i .EQ. 1) THEN
     diffdesign(i,1,1)=basiceff(1)
     diffdesign(i,2,1)=addeff(1)
     diffmatrix(i,1)=1
    ELSE
      diffdesign(i,1,1)=basiceff(lenvec(i-1)+1)
      diffdesign(i,2,1)=addeff(1)
      diffmatrix(i,1)=lenvec(i-1)+1
    END IF
  END DO
  call date_and_time(b(1),b(2),b(3),date_time)
  WRITE(1,*) "The current time :: ", b(1),b(2)
  jstep=1
  idiff=MM-2
  DO i=1,idiff
   WRITE(1,*)diffdesign(i,1,1)
  END DO
  close(1)
  jstep=jstep+1
  idiff=MM-3
  DO WHILE(jstep .LT. 9)
      ifile=jstep
      p=jstep+MM
      SELECT CASE(ifile)
       CASE(2)
        filenames="design128_2.out"
       CASE(3)
        filenames="design128_3.out"
       CASE(4)
        filenames="design128_4.out"
       CASE(5)
```

```
       filenames="design128_5.out"
      CASE(6)
       filenames="design128_6.out"
      CASE(7)
       filenames="design128_7.out"
      CASE(8)
       filenames="design128_8.out"
      CASE(9)
       filenames="design128_9.out"
   END SELECT
   open(ifile,file=filenames)
   call date_and_time(b(1),b(2),b(3),date_time)
   WRITE(ifile,*) "The current time :: ", b(1),b(2)
   DO ii=1,idiff
      first=.TRUE.
      DO jii=diffmatrix(ii,(jstep-1))+1,lenvec(MM-2)
        IF(ii .EQ. 1 .AND. first)THEN
          first=.FALSE.
          new_diffdesign(1,:,1:(jstep-1))=diffdesign(1,:,1:(jstep-1))
          new_diffdesign(1,1,jstep)=basiceff(jii)
          new_diffdesign(1,2,jstep)=addeff(jstep)
          diffmatrix(1,jstep)=diffmatrix(1,jstep-1)+1
          new_idiff=1
        END IF
        design1(:,1:(jstep-1))=diffdesign(ii,:,1:(jstep-1))
        design1(1,jstep)=basiceff(jii)
        design1(2,jstep)=addeff(jstep)
        new_ii=1
      IF(jstep .LE. 11)THEN
call lpattern(design1(:,1:jstep),MM,jstep,wlp(1,:),pparr(1,:,1:p),&
     pparr2(1,:,1:p),pparr3(1,:,1:p),pparr4(1,:,1:p),wmatrix(1,:,1:p))
      END IF
      IF(wlp(1,3) .NE.0)THEN
         go to 998
      END IF
      if(wlp(1,4).ne.0)then
      call xtransp2(pparr(1,1:wlp(1,4),1:p),wlp(1,4),p,&
           tpp(1,1:p,1:wlp(1,4)))
      call multxy2(tpp(1,1:p,1:wlp(1,4)),pparr(1,1:wlp(1,4),1:p),&
           p,wlp(1,4),p,tpppp(1,1:p,1:p))
      call SSYEV('N','U',p,real(tpppp(1,1:p,1:p)),p,&
           eigenval2(1,1:p),work,lwork,infor)
      end if
      if(wlp(1,5).NE.0)then
      call xtransp2(pparr2(1,1:wlp(1,5),1:p),wlp(1,5),&
           p,tpp2(1,1:p,1:wlp(1,5)))
      call multxy2(tpp2(1,1:p,1:wlp(1,5)),pparr2(1,1:wlp(1,5),1:p),&
           p,wlp(1,5),p,tpppp2(1,1:p,1:p))
      call SSYEV('N','U',p,real(tpppp2(1,1:p,1:p)),p,&
```

```
      eigenval22(1,1:p),work2,lwork,infor2)
  end if
  if(wlp(1,6).NE.0)then
  call xtransp2(pparr3(1,1:wlp(1,6),1:p),wlp(1,6),&
       p,tpp3(1,1:p,1:wlp(1,6)))
  call multxy2(tpp3(1,1:p,1:wlp(1,6)),pparr3(1,1:wlp(1,6),1:p),&
       p,wlp(1,6),p,tpppp3(1,1:p,1:p))
  call SSYEV('N','U',p,real(tpppp3(1,1:p,1:p)),p,&
       eigenval32(1,1:p),work3,lwork,infor3)
  end if
  if(wlp(1,7).NE.0)then
  call xtransp2(pparr4(1,1:wlp(1,7),1:p),wlp(1,7),&
       p,tpp4(1,1:p,1:wlp(1,7)))
  call multxy2(tpp4(1,1:p,1:wlp(1,7)),pparr4(1,1:wlp(1,7),1:p),&
       p,wlp(1,7),p,tpppp4(1,1:p,1:p))
  call SSYEV('N','U',p,real(tpppp4(1,1:p,1:p)),p,&
      eigenval42(1,1:p),work4,lwork,infor4)
  end if
  DO
    diff=.FALSE.
    design2(:,1:jstep)=new_diffdesign(new_ii,:,1:jstep)
  IF(jstep .LE.11)THEN
call lpattern(design2(:,1:jstep),MM,jstep,wlp(2,:),pparr(2,:,1:p)&
     ,pparr2(2,:,1:p),pparr3(2,:,1:p),pparr4(2,:,1:p),wmatrix(2,:,1:p))
  END IF
  IF(wlp(2,3) .NE.0)THEN
       go to 998
  END IF
  jjii=0
  IF(sum(abs(wlp(1,1:p)-wlp(2,1:p))) .GT. 1.91) THEN
   diff=.TRUE.
  ELSE
  if(wlp(2,4).ne.0 .and. (.not. diff))then
  call xtransp2(pparr(2,1:wlp(2,4),1:p),wlp(2,4),&
       p,tpp(2,1:p,1:wlp(2,4)))
  call multxy2(tpp(2,1:p,1:wlp(2,4)),pparr(2,1:wlp(2,4),1:p),&
       p,wlp(2,4),p,tpppp(2,1:p,1:p))
  call SSYEV('N','U',p,real(tpppp(2,1:p,1:p)),p,&
       eigenval2(2,1:p),work,lwork,infor)
IF(sum(abs(eigenval2(1,1:p)-eigenval2(2,1:p))) .GT. 1) THEN
        diff=.TRUE.
else
 do i=jjii+1,jjii+wlp(1,4)
  do k=1,p
    cpparr(1,i,k)=pparr(1,i-jjii,k)
    cpparr(2,i,k)=pparr(2,i-jjii,k)
  end do
 end do
 jjii=jjii+wlp(1,4)
```

```
     end if
 end if
 if(wlp(2,5).ne.0 .and. (.not. diff))then
    call xtransp2(pparr2(2,1:wlp(2,5),1:p),wlp(2,5),p,&
         tpp2(2,1:p,1:wlp(2,5)))
    call multxy2(tpp2(2,1:p,1:wlp(2,5)),pparr2(2,1:wlp(2,5),1:p),&
         p,wlp(2,5),p,tpppp2(2,1:p,1:p))
    call SSYEV('N','U',p,real(tpppp2(2,1:p,1:p)),p,&
         eigenval22(2,1:p),work2,lwork,infor2)
    IF(sum(abs(eigenval22(1,1:p)-eigenval22(2,1:p))) .GT. 0.5) THEN
            diff=.TRUE.
    else
       do i=jjii+1,jjii+wlp(1,5)
        do k=1,p
          cpparr(1,i,k)=pparr2(1,i-jjii,k)
          cpparr(2,i,k)=pparr2(2,i-jjii,k)
        end do
       end do
      jjii=jjii+wlp(1,5)
   end if
end if
if(wlp(1,4).ne.0 .and. wlp(1,5).ne.0 .and. (.not.diff))then
 call xtransp2(cpparr(1,1:jjii,1:p),jjii,p,tcpp(1,1:p,1:jjii))
 call multxy2(tcpp(1,1:p,1:jjii),cpparr(1,1:jjii,1:p),&
      p,jjii,p,tcpppp(1,1:p,1:p))
 call SSYEV('N','U',p,real(tcpppp(1,1:p,1:p)),p,& eigenvalc(1,1:p),work4,lwork,infor4)
 call xtransp2(cpparr(2,1:jjii,1:p),jjii,p,tcpp(2,1:p,1:jjii))
 call multxy2(tcpp(2,1:p,1:jjii),cpparr(2,1:jjii,1:p),p,&
      jjii,p,tcpppp(2,1:p,1:p))
 call SSYEV('N','U',p,real(tcpppp(2,1:p,1:p)),p,&
      eigenvalc(2,1:p),work4,lwork,infor4)
 IF(sum(abs(eigenvalc(1,1:p)-eigenvalc(2,1:p))) .GT. 0.5) THEN
          diff=.TRUE.
   end if
  end if
 if(wlp(2,6).ne.0 .and. (.not.diff))then
   call xtransp2(pparr3(2,1:wlp(2,6),1:p),wlp(2,6),&
        p,tpp3(2,1:p,1:wlp(2,6)))
   call multxy2(tpp3(2,1:p,1:wlp(2,6)),pparr3(2,1:wlp(2,6),1:p),&
        p,wlp(2,6),p,tpppp3(2,1:p,1:p))
   call SSYEV('N','U',p,real(tpppp3(2,1:p,1:p)),p,&
        eigenval32(2,1:p),work,lwork,infor)
  IF(sum(abs(eigenval32(1,1:p)-eigenval32(2,1:p))) .GT. 0.5) THEN
              diff=.TRUE.
    else
   do i=jjii+1,jjii+wlp(1,6)
   do k=1,p
     cpparr(1,i,k)=pparr3(1,i-jjii,k)
     cpparr(2,i,k)=pparr3(2,i-jjii,k)
```

```
     end do
    end do
   jjii=jjii+wlp(1,6)
  call xtransp2(cpparr(1,1:jjii,1:p),jjii,p,tcpp(1,1:p,1:jjii))
  call multxy2(tcpp(1,1:p,1:jjii),cpparr(1,1:jjii,1:p),&
       p,jjii,p,tcpppp(1,1:p,1:p))
  call SSYEV('N','U',p,real(tcpppp(1,1:p,1:p)),&
       p,eigenvalc(1,1:p),work4,lwork,infor4)
  call xtransp2(cpparr(2,1:jjii,1:p),jjii,p,tcpp(2,1:p,1:jjii))
  call multxy2(tcpp(2,1:p,1:jjii),cpparr(2,1:jjii,1:p),&
       p,jjii,p,tcpppp(2,1:p,1:p))
  call SSYEV('N','U',p,real(tcpppp(2,1:p,1:p)),p,&
       eigenvalc(2,1:p),work4,lwork,infor4)
  IF(sum(abs(eigenvalc(1,1:p)-eigenvalc(2,1:p))) .GT. 0.5) THEN
          diff=.TRUE.
   end if
  end if
end if
if(wlp(2,7).ne.0 .and. (.not. diff))then
  call xtransp2(pparr4(2,1:wlp(2,7),1:p),wlp(2,7),&
       p,tpp4(2,1:p,1:wlp(2,7)))
  call multxy2(tpp4(2,1:p,1:wlp(2,7)),pparr4(2,1:wlp(2,7),1:p),&
       p,wlp(2,7),p,tpppp4(2,1:p,1:p))
  call SSYEV('N','U',p,real(tpppp4(2,1:p,1:p)),p,&
       eigenval42(2,1:p),work,lwork,infor)
 IF(sum(abs(eigenval42(1,1:p)-eigenval42(2,1:p))) .GT. 0.5) THEN
              diff=.TRUE.
    else
        do i=jjii+1,jjii+wlp(1,7)
         do k=1,p
         cpparr(1,i,k)=pparr4(1,i-jjii,k)
         cpparr(2,i,k)=pparr4(2,i-jjii,k)
         end do
        end do
       jjii=jjii+wlp(1,7)
   call xtransp2(cpparr(1,1:jjii,1:p),jjii,p,tcpp(1,1:p,1:jjii))
   call multxy2(tcpp(1,1:p,1:jjii),cpparr(1,1:jjii,1:p),&
        p,jjii,p,tcpppp(1,1:p,1:p))
   call SSYEV('N','U',p,real(tcpppp(1,1:p,1:p)),&
        p,eigenvalc(1,1:p),work4,lwork,infor4)
   call xtransp2(cpparr(2,1:jjii,1:p),jjii,&
        p,tcpp(2,1:p,1:jjii))
   call multxy2(tcpp(2,1:p,1:jjii),cpparr(2,1:jjii,1:p),&
        p,jjii,p,tcpppp(2,1:p,1:p))
   call SSYEV('N','U',p,real(tcpppp(2,1:p,1:p)),&
        p,eigenvalc(2,1:p),work4,lwork,infor4)
   IF(sum(abs(eigenvalc(1,1:p)-eigenvalc(2,1:p))) .GT. 0.5) THEN
              diff=.TRUE.
    else
```

```
      if(jstep .LE. MM)then
      call wcheck(wmatrix(1,1:(2**jstep-1),1:p),&
           wmatrix(2,1:(2**jstep-1),1:p),MM,p,2**jstep-1,diff)
      else
      call deseq2(design1(:,1:jstep),design2(:,1:jstep),MM,p,diff)
           end if
        end if
        END IF
       END IF
      END IF
      IF(.NOT. diff) THEN
             go to 998
       ELSE
           new_ii=new_ii+1
      END IF
      IF(new_ii .EQ. (new_idiff+1)) THEN
           new_idiff=new_idiff+1
           new_diffdesign(new_idiff,1:2,1:jstep)=design1(1:2,1:jstep)
           diffmatrix(new_idiff,jstep)=jii
           go to 998
       END IF
      END DO
998  END DO
   END DO
     idiff=new_idiff
     diffdesign=new_diffdesign
     DO i=1,idiff
      WRITE(ifile,*)diffdesign(i,1,1:jstep)
     END DO
     call date_and_time(b(1),b(2),b(3),date_time)
     WRITE(ifile,*) "The current time :: ", b(1),b(2)
     jstep=jstep+1
     close(ifile)
     END DO
999 END PROGRAM alldesign
!**********************************************************************
!this is the program to produce wlp
!gen is the matrix with row for generators
!wmat[i,] is ith generator 's letters

 SUBROUTINE lpattern(design,MM,addnum,wlp,ppmat,ppmat2,&
              ppmat3,ppmat4,ppmatall)
  INTEGER::i,j,bits,sm,b,bb,r,k,jj,l,addnum,jfive,jsix,jseven
  INTEGER::numb,numb2,numb3,numb4
  CHARACTER(20)::lpfile,ppfile
  INTEGER,PARAMETER::ncols=20,nrows=260,ncol=260
  INTEGER,DIMENSION(2,addnum)::design
  INTEGER,DIMENSION(ncols)::wlp
  INTEGER,DIMENSION(nrows)::digit
```

```
INTEGER,DIMENSION(addnum,ncols)::wmat
INTEGER,DIMENSION(nrows,ncols)::gen
INTEGER,DIMENSION(ncol,addnum+MM)::ppmat,ppmat2,ppmat3,ppmat4,ppmatall

DO i=1,nrows
  digit(i)=0
END DO
DO i=1,nrows
  DO j=1,ncols
     gen(i,j)=0
  END DO
 END DO
do i=1,addnum
  do j=1,ncols
     wmat(i,j)=0
   end do
 end do
 DO i=1,ncol
   DO j=1,addnum+MM
    ppmat(i,j)=0
    ppmat2(i,j)=0
    ppmat3(i,j)=0
    ppmat4(i,j)=0
    ppmatall(i,j)=0
   END DO
 END DO
 Do i=1,addnum
    bits=FLOOR(LOG10(real(design(1,i))))+1+1
   DO b=bits,2,-1
     IF(b .EQ. bits)THEN
      wmat(i,1)=FLOOR(real(design(1,i))/real(10**(b-1-1)))
     ELSE
      sm=SUM((/(wmat(i,bits-bb+1)*10**(bb-1-1),bb=bits,b+1,-1)/))
      wmat(i,bits-b+1)=FLOOR(real(design(1,i)-sm)/real(10**(b-2)))
     END IF
   END DO
      wmat(i,bits)=design(2,i)
  END DO
  gen(1,:)=wmat(1,:)
  gen(2,:)=wmat(2,:)
  call gene(gen(1,:),gen(2,:),gen(3,:))
  r=3
  IF( addnum .GE. 3)THEN
    DO i=3,addnum
      r=r+1
      gen(r,:)=wmat(i,:)
    DO jj=r+1,r+2**(i-1)-1
      if(jj .EQ. r+2**(i-1)-1)THEN
       call gene(gen(jj-r,:),gen(r,:),gen(jj,:))
```

```
      r=r+2**(i-1)-1
     ELSE
      call gene(gen(jj-r,:),gen(r,:),gen(jj,:))
     END IF
    END DO
   END DO
 END IF
DO i=1,r
 j=1
 DO WHILE(gen(i,j) .NE. 0)
   j=j+1
 END DO
   digit(i)=j-1
END DO
DO i=1,ncols
  wlp(i)=0
END DO
DO i=1,r
 wlp(digit(i))=wlp(digit(i))+1
END DO
numb=0
  DO i=1,r
     j=1
     IF(digit(i) .EQ.4)THEN
       numb=numb+1
     END IF
     DO WHILE(digit(i).EQ.4 .AND. gen(i,j).NE. 0)
       ppmat(numb,gen(i,j))=1
       j=j+1
     END DO
  END DO
  numb2=0
  DO i=1,r
     jfive=1
     IF(digit(i) .EQ.5)THEN
       numb2=numb2+1
     END IF
     DO WHILE(digit(i).EQ.5 .AND. gen(i,jfive).NE. 0)
       ppmat2(numb2,gen(i,jfive))=1
       jfive=jfive+1
     END DO
  END DO
  numb3=0
  DO i=1,r
     jsix=1
     IF(digit(i) .EQ.6)THEN
       numb3=numb3+1
     END IF
     DO WHILE(digit(i).EQ.6 .AND. gen(i,jsix).NE. 0)
```

```
          ppmat3(numb3,gen(i,jsix))=1
          jsix=jsix+1
        END DO
      END DO
      numb4=0
       DO i=1,r
        jseven=1
        IF(digit(i) .EQ.7)THEN
          numb4=numb4+1
        END IF
        DO WHILE(digit(i).EQ.7 .AND. gen(i,jseven).NE. 0)
          ppmat4(numb4,gen(i,jseven))=1
          jseven=jseven+1
        END DO
       END DO
       Do i=1,r
        j=1
         DO WHILE(gen(i,j).NE.0)
          ppmatall(i,gen(i,j))=1
          j=j+1
         END DO
       END DO
  END SUBROUTINE lpattern
!************************************************************************
!program to generate a new generator which is product of two generators
   SUBROUTINE gene(r1,r2,r1r2)
     INTEGER::i,j
     INTEGER,PARAMETER::ncols=20
     INTEGER,DIMENSION(ncols)::r1,r2,r1r2,csum
      INTEGER,DIMENSION(2,ncols)::imat

       DO i=1,2
         DO j=1,ncols
          imat(i,j)=0
         END DO
       END DO
       i=1
       DO WHILE(r1(i) .NE. 0)
         imat(1,r1(i))=1
         i=i+1
       END DO
       i=1
       DO WHILE(r2(i) .NE. 0)
         imat(2,r2(i))=1
         i=i+1
       END DO
       DO i=1,ncols
          csum(i)=imat(1,i)+imat(2,i)
       END DO
```

```
       j=1
       DO i=1,ncols
         IF(csum(i) .EQ. 1)THEN
           r1r2(j)=i
           j=j+1
         END IF
       END DO

      END SUBROUTINE gene
!**********************************************************************
!*     subroutine to calculate transpose of a matrix
      subroutine xtransp2(x,n,p,xt)
      integer::i,j,n,p
      integer x(n,p), xt(p,n)

      do i=1,n
         do j=1,p
            xt(j,i)=x(i,j)
         end do
      end do
      return
      end
!**********************************************************************
!*     subroutine to multiply two matrices
      subroutine multxy2(x,y,nx,px,py,xy)
      integer::nx,px,py,i,j,prod
      integer,dimension(nx,px)::x
      integer,dimension(px,py)::y
      integer,dimension(nx,py)::xy

      do i=1,nx
         do j=1,py
            prod=0.
            do k=1,px
               prod=prod+x(i,k)*y(k,j)
            end do
            xy(i,j)=prod
         end do
      end do
      return
    end
!**********************************************************************
       subroutine deseq2(des1,des2,bm,p,diff)
!*     Copyright James C Clark.  May 7th 1988.
!*     James.B.Clark@aero.org  (Jim Clark)
!*     dean.9@osu.edu    (Angela Dean)
!*     deseq1.f and deseq2.f are used to determine design equivalence.
!*     deseq1.f should be run first - it performs several tests for
!*     equivalence.  deseq2.f can be run concurrently - it tries to find
```

```
!*     the permutations showing the equivalence.  Much time may be wasted
!*     if this second search is done without starting the tests of deseq1.f.
!*     deseq2.f
!*     Given matrices x1 and x2 (stored in design1 and design2,
!*     with the first line containing the number of rows and the
!*     number of columns) a row permutation matrix r and a column
!*     permutation matrix c are found such that x1=rx2cl, where
!*     l is a diagonal matrix of +/- 1's.  l can be found by then
!*     equating the first row of x1 and rx2c.
!*     The program has been adapted.
!*     First an r matrix is found by comparing x1x1' and x2x2':
      integer,parameter::nrow=128
      integer n,i,j,row,col,curcol(nrow),maxstage,bits,b,bb,sm1,sm2,bm,p
      integer x1x1p(nrow,nrow),x2x2p(nrow,nrow),r(nrow,nrow)
      integer ri(0:nrow,0:nrow,0:nrow),x2t(nrow,nrow)
      integer x1(nrow,nrow),x2(nrow,nrow),x1t(nrow,nrow)
      integer,dimension(2,p-bm)::des1,des2
      integer,dimension(bm)::bitsvec
      logical diff,done
      done=.FALSE.
      n=2**bm
      DO i=1,bm
         DO j=0,(2**(bm-i)-1)*2**(i-1),2**(i-1)
            x1((1+2*j):(2**(i-1)+2*j),i)=1
            x1((1+2**(i-1)+2*j):(2**i+2*j),i)=-1
          END DO
      END DO
      DO j=bm+1,p
!    bits is the digits of design1(1,j-bm)=basiceff
          bits=FLOOR(LOG10(real(des1(1,j-bm))))+1
          DO b=bits,1,-1
             IF(b .EQ. bits)THEN
                bitsvec(b)=FLOOR(real(des1(1,j-bm))/real(10**(b-1)))
             ELSE
                sm1=SUM((/(bitsvec(bb)*10**(bb-1),bb=bits,b+1,-1)/))
                bitsvec(b)=FLOOR(real(des1(1,j-bm)-sm1)/real(10**(b-1)))
             END IF
          END DO
        x1(1:2**bm,j)=product(x1(1:2**bm,bitsvec(1:bits)),DIM=2)
      END DO

       x2(1:2**bm,1:bm)=x1(1:2**bm,1:bm)
       DO j=bm+1,p
! bits+1 is the digits of design2(j-bm)
        bits=FLOOR(LOG10(real(des2(1,j-bm))))+1
        DO b=bits,1,-1
          IF(b .EQ. bits)THEN
            bitsvec(b)=FLOOR(real(des2(1,j-bm))/real(10**(b-1)))
          ELSE
```

```
          sm2=SUM((/(bitsvec(bb)*10**(bb-1),bb=bits,b+1,-1)/))
          bitsvec(b)=FLOOR(real(des2(1,j-bm)-sm2)/real(10**(b-1)))
        END IF
       END DO
        x2(1:2**bm,j)=product(x2(1:2**bm,bitsvec(1:bits)),DIM=2)
       END DO

!*     Calculate x1x1' and x2x2'

      call xtransp(x1,n,p,x1t,nrow)
      call multxy(x1,x1t,n,p,n,x1x1p,nrow)
      call xtransp(x2,n,p,x2t,nrow)
      call multxy(x2,x2t,n,p,n,x2x2p,nrow)

!*     Clear the vector of the current row permutation
      do i=1,n
         curcol(i)=0
      end do

!*     Initialize the potential r matrices.
      call rcand(x1,x2,n,p,r,nrow)
      do i=1,n
         do j=1,n
            ri(0,i,j)=r(i,j)
         end do
      end do

!*     Find an r matrix
      row=1
      do while (row.ge.1)
         col = curcol(row)+1
         do while ((abs(r(row,col)).lt.0.1).and.(col.le.n))
            col = col+1
         end do
         if (col.eq.n+1) then
            do i=row-1,n
               do j=1,n
                  r(i,j)=ri(row-2,i,j)
               end do
            end do
            row=row-1
         else
            curcol(row)=col
            do i=row+1,n
               curcol(i)=0
            end do
            do j=1,n
               if(j.ne.row) r(j,col)=0.
               if(j.ne.col) r(row,j)=0.
```

```
            end do
            do j=row+1,n
               do i=1,n
                  if(abs(x1x1p(row,j)-x2x2p(col,i)).gt.0.1) r(j,i)=0.
               end do
            end do
            do i=1,n
               do j=1,n
                  ri(row,i,j)=r(i,j)
               end do
            end do
            if(row.eq.n) then
               row = row-1
               call findc(x1,x2,curcol,r,n,p,maxstage,nrow,done)
               if(done)then
                go to 99
               end if
            else
               row=row+1
            end if
          end if
       end do
      diff=.TRUE.
99    end subroutine deseq2

!**********************************************************************
!x1 and x2 are word matrices of design1 and design2,respectively
!the same principle as deseq2
      subroutine wcheck(w1,w2,bm,p,wl,diff)
      logical diff,done
      integer, parameter::nrow=64
      integer p,wl
      integer i,j,row,col,curcol(nrow)
      integer n,maxstage
      integer w1(wl,p),w2(wl,p)
      integer x1x1p(nrow,nrow),x2x2p(nrow,nrow),r(nrow,nrow)
      integer ri(0:nrow,0:nrow,0:nrow),x2t(nrow,nrow)
      integer x1(nrow,nrow),x2(nrow,nrow),x1t(nrow,nrow)
      n=wl
!*     Calculate x1x1' and x2x2'
      done=.false.
      x1(1:n,1:p)=w1
      x2(1:n,1:p)=w2
      call xtransp(x1,n,p,x1t,nrow)
      call multxy (x1,x1t,n,p,n,x1x1p,nrow)
      call xtransp(x2,n,p,x2t,nrow)
      call multxy (x2,x2t,n,p,n,x2x2p,nrow)

!*     Clear the vector of the current row permutation
```

```
      do i=1,n
         curcol(i)=0
      end do

!*     Initialize the potential r matrices.
      call rcand(x1,x2,n,p,r,nrow)
      do i=1,n
         do j=1,n
            ri(0,i,j)=r(i,j)
         end do
      end do

!*     Find an r matrix
      row=1
      do while (row.ge.1)
         col = curcol(row)+1
         do while ((abs(r(row,col)).lt.0.1).and.(col.le.n))
            col = col+1
         end do
         if (col.eq.n+1) then
            do i=row-1,n
               do j=1,n
                  r(i,j)=ri(row-2,i,j)
               end do
            end do
            row=row-1
         else
            curcol(row)=col
            do i=row+1,n
               curcol(i)=0
            end do
            do j=1,n
               if(j.ne.row) r(j,col)=0.
               if(j.ne.col) r(row,j)=0.
            end do
            do j=row+1,n
               do i=1,n
                  if(abs(x1x1p(row,j)-x2x2p(col,i)).gt.0.1) r(j,i)=0.
               end do
            end do
            do i=1,n
               do j=1,n
                  ri(row,i,j)=r(i,j)
               end do
            end do
            if(row.eq.n) then
               row = row-1
               call findc (x1,x2,curcol,r,n,p,maxstage,nrow,done)
                if(done)then
```

```
                go to 9999
               end if
             else
               row=row+1
             end if
           end if
      end do
      diff=.TRUE.

9999      end subroutine wcheck

!*****************************************************************
!*     Subroutine to find a c matrix for a given r matrix
      subroutine findc (x1,x2,curcol,r,n,p,maxstage,nrow,done)
      integer nrow,done
      integer  x1(nrow,nrow), x2(nrow,nrow),r(nrow,nrow)
      integer stage,p,n,del_cols(nrow),j,maxstage,curcol(nrow)
      logical valid,test_fails

      do j=1,p
         del_cols(j)=0
      end do

!*     Search through the tree to find a c:
      maxstage=1
      stage=1
      do while(stage.lt.p)
         del_cols(stage)=p+1
         test_fails=.true.
         do while(test_fails)
            del_cols(stage)=del_cols(stage)-1
            do while(del_cols(stage).eq.0)
               stage=stage-1
               if(stage.eq.0) then
                  return
               end if
               del_cols(stage)=del_cols(stage)-1
            end do
            valid = .false.
            do while((.not.valid).and.(del_cols(stage).gt.0))
               valid=.true.
               do j=stage-1,1,-1
                  if(del_cols(stage).eq.del_cols(j)) valid=.false.
               end do
               if (.not.valid) then
                  del_cols(stage)=del_cols(stage)-1
               end if
            end do
            if (del_cols(stage).eq.0) then
```

```
               del_cols(stage)=1
            else
               call test(x1,x2,n,p,del_cols,r,stage,test_fails,nrow)
            end if
         end do
         stage=stage+1
         if (stage.gt.maxstage) maxstage=stage
      end do
      call equal(r,del_cols,n,p,maxstage,nrow,done)
      end
!*************************************************************
!*     If the last stage is successful, designs are equivalent

      subroutine equal(r,del_cols,n,p,maxstage,nrow,done)
      integer nrow
      integer r(nrow,nrow)
      integer del_cols(nrow),maxstage
      integer i,n,p,rperm(nrow),cperm(nrow)
      logical done
!*     Determine the row permutation
      do i=1,n
         j=1
         do while (nint(real(r(i,j))).eq.0)
            j=j+1
         end do
         rperm(i)=j
      end do

!*     Determine the column permutation
      cperm(1)=0
      do i=p,2,-1
         cperm(i)=del_cols(p-i+1)
         cperm(1)=cperm(1)+cperm(i)
      end do
      cperm(1)=P*(p+1)/2-cperm(1)
      done=.TRUE.
!      write(*,*) 'Passes stage ',maxstage
!      write(*,*) 'Equal'
!      write(*,*) 'Column permutation',(cperm(i),i=1,p)
!      write(*,*) 'Row Permutation',(rperm(i),i=1,n)
      end
!*************************************************************
!*     At each stage, tests if x1s1x1'= rx2sx2'r', where s1 and s
!*     are diagonal matrices of 0's and 1's that select the
!*     appropriate columns for each stage.

      subroutine test(x1,x2,n,p,del_cols,r,stage,test_fails,nrow)
      integer nrow
      integer x1(nrow,nrow), x2(nrow,nrow), r(nrow,nrow),s(nrow,nrow)
```

```
      integer x1t(nrow,nrow), x2t(nrow,nrow), mx1(nrow,nrow)
      integer  m2(nrow,nrow),mx2(nrow,nrow),m3(nrow,nrow),rt(nrow,nrow)
      integer dif
      integer n,p, del_cols(nrow),stage
      logical test_fails

!*     Calculate mx1 = x1s1x1
      test_fails=.true.
      call xtransp(x1,n,p-stage,x1t,nrow)
      call multxy(x1,x1t,n,p-stage,n,mx1,nrow)

!*     Calculate s according to which columns were deleted
      do i=1,p
         do j=1,p
            s(i,j)=0.
         end do
         s(i,i)=1.
      end do

      i=1
      do while(del_cols(i).ne.0)
         s(del_cols(i),del_cols(i))=0.
         i=i+1
      end do

!*     Calculate  mx2 = rx2sx2'r'
      call multxy(r,x2,n,n,p,m2,nrow)
      call multxy(m2,s,n,p,p,m3,nrow)
      call xtransp(x2,n,p,x2t,nrow)
      call multxy(m3,x2t,n,p,n,m2,nrow)
      call xtransp(r,n,n,rt,nrow)
      call multxy(m2,rt,n,n,n,mx2,nrow)

!*     Test if mx1 = mx2.  Looks for large sum of squares of differences.
      dif=0.
      do i=1,n
         do j=1,n
            dif=dif+(mx1(i,j)-mx2(i,j))**2
         end do
      end do
      if (dif.lt.1.) test_fails=.false.
      return
      end
!************************************************************
!*     Find candidate r matrices

      subroutine rcand(x1,x2,n,p,r,nrow)
      integer nrow
      integer x1(nrow,nrow), x2(nrow,nrow),r(nrow,nrow)
```

```
      integer x1t(nrow,nrow), x2t(nrow,nrow)
      integer x1x1t(nrow,nrow), x2x2t(nrow,nrow)
      integer countsx1(nrow,nrow), countsx2(nrow,nrow)
      integer k,jj,i,j,n,p,match(nrow,nrow)

      call xtransp(x1,n,p,x1t,nrow)
      call multxy (x1,x1t,n,p,n,x1x1t,nrow)
      call xtransp(x2,n,p,x2t,nrow)
      call multxy (x2,x2t,n,p,n,x2x2t,nrow)
      call rowcounts(x1x1t,n,p,countsx1,nrow)
      call rowcounts(x2x2t,n,p,countsx2,nrow)

      do i=1,n
         do j=1,n
            match(i,j) =0
            r(i,j)=0
         end do
      end do

      do i=1,n
         jj=1
         do j=1,n
            k=1
            do while (countsx1(i,k).eq.countsx2(j,k).and.(k.le.p+1))
               k=k+1
            end do
            if(k.ge.p+1) then
               match(i,jj)=j
               jj=jj+1
            end if
         end do
      end do

      do i=1,n
         do j=1,n
            if(match(i,j).ne.0) r(i,match(i,j))=1
         end do
      end do
      return
      end
!*********************************************************************
!*     Subroutine to calculate the frequency of the integers in each
!*     row of XX';  e.g., if first row is [3 1 -1 -3], the row count
!*     for that row is [1 1 1 1].

      subroutine rowcounts (xxt, n, p, counts,nrow)
      integer nrow
      integer i, j, jj, n, p, counts(nrow,nrow)
      integer xxt(nrow,nrow)
```

```
      do i=1,n
         do j=1,p+1
            counts(i,j) = 0
         end do
        do j=1,n
            jj=(xxt(i,j)+p)/2+1
            counts(i,jj) = counts(i,jj)+1
         end do
      end do
      return
      end
!*****************************************************************
!*     subroutine to calculate transpose of a matrix
      subroutine xtransp(x,n,p,xt,nrow)
      integer nrow
      integer x(nrow,nrow), xt(nrow,nrow)
      integer i,j,n,p

      do i=1,n
         do j=1,p
            xt(j,i)=x(i,j)
         end do
      end do
      return
      end
!*****************************************************************
!*     subroutine to multiply two matrices
      subroutine multxy (x,y,nx,px,py,xy,nrow)
      integer nrow
      integer x(nrow,nrow), y(nrow,nrow), xy(nrow,nrow)
      integer nx,px,py,i,j
      integer prod

      do i=1,nx
         do j=1,py
            prod=0.
            do k=1,px
               prod=prod+x(i,k)*y(k,j)
            end do
            xy(i,j)=prod
         end do
      end do
      return
      end
```

Bibliography

Bingham, D. & Sitter, R. R. (1999) Minimum aberration fractional factorial split-plot designs. *Technometrics,* **41**, 62–70.

Block, M. R. & Mee, W. R. (2004) Resolution *iv* designs with 128 runs. *Unknown,* **41**.

Box, G. E. P. & Hunter, J. S. (1961) The 2^{kp} fractional factorial design. *Technometrics,* **3**, 311–449.

Chen, J. (1992) Some results on 2^{n-k} fractional factorial designs and search for minimum aberration designs. *Ann. Statist.,* **20**, 2124–2141.

Chen, J. & Lin, D. K. J. (1990) On the identity relationships of a 2^{k-p} design. *J. Statist. Plan. Infer.,* **28**, 95–98.

Chen, J., Sun, D. X. & Wu, C. F. J. (1993) A catalogue of two-level and three-level fractional factorial designs with small runs. *International Statistical Review,* **61** (1), 131–145.

Clark, J. B. & Dean, A. M. (2001) Equivalence of fractional factorial designs. *Statist. Sinica,* **11**, 537–547.

Draper, N. R. & Mitchell, T. J. (1968) Construction of the set of 256-run designs of resolution ≥ 5 and the set of even 512-run designs of resolution ≥ 6 with special reference to the unique saturated designs. *Ann. Math. Statist.,* **39**, 246–255.

Draper, N. R. & Mitchell, T. J. (1970) Construction of the set of 512-run designs of resolution ≥ 5 and the set of even 1024-run designs of resolution ≥ 6. *Ann. Math. Statist.,* **41**, 876–887.

Franklin, M. F. (1985) Selecting defining contrasts and confounded effects in p^{n-m} factorial experiments. *Technometrics,* **27**, 165–172.

Fries, A. & Hunter, W. (1980) Minimum aberration 2^{k-p} designs. *Technometrics,* **22**, 601–608.

Ma, C. X., Fang, K. T. & Lin, D. K. J. (2001) On the isomorphism of fractional factorial designs. *Journal of Complexity,* **17**, 86–97.

Montgomery, C. D. (2001) *Experiment Design.* John Wiley And Sun. INC, NEW YORK, U.S.

Sun, D. X., Li, W. & Ye, Q. (2002). An algorithm for sequentially constructing non-isomorphic orthogonal designs and its applications. Technical report SUNYSB-AMS.

Wu, C. F. J. & Hamada, M. (2000) *Experiments Planning, Analysis, and Parameter Designs Optimization.* NEW YORK, U.S: Wiley-Interscience Publication.