INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
LAMBDA–DESIGNS WITH SMALL LAMBDA ARE TYPE–1

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

Iván Weisz, M.S., M.A.

* * * * *

The Ohio State University

1995

Dissertation Committee:  
Professor Ákos Seress
Professor Dijen Ray-Chaudhuri
Professor Thomas Dowling

Approved by

Adviser
Department of Mathematics
ACKNOWLEDGEMENTS

I would like to express my appreciation to Dr. Ákos Seress, my adviser, for his guidance, encouragement and patience throughout my studies. I would also like to thank Professors Dijen Ray-Chaudhuri, Thomas Dowling and Gerald Newsom for their time and suggestions provided as members of my dissertation committee.
VITA

May 4, 1962 ............................................................... Born - Budapest, Hungary

1985 ................................................................. M.S. Mathematics

1985 ................................................................. M.A. Mathematics Education

1986 ................................................................. M.A. Computer Education

1985-87 ............................................................. Head of Computer Department, St.
Stephen High School, Budapest, Hungary

1990 ................................................................. M.S. Computer and Information Science

1992 ................................................................. M.S. Mathematics

1987-present ...................................................... Graduate Teaching Associate,
The Ohio State University.

Publications

Research Publications

I. Weisz, An Implementation of the RSA Algorithm M.S. Thesis, Eötvös University,
Budapest, Prof. Miklós Simonovits adviser, 1985 (Hungarian)

Á. Seress, I. Weisz, PERM: A Program Computing Strong Generating Sets, Groups
and Computations, DIMACS Series in Discrete Mathematics and Theoretical Com-

Á. Seress, I. Weisz, *λ-designs with small λ are type-1*, in preparation

Fields of Study

Major Field: Mathematics

Studies in:
- Combinatorics  Prof. Eiichi Bannai
- Combinatorics  Prof. Igor Križ
- Combinatorics  Prof. Dijen Ray-Chaudhuri
- Combinatorics  Prof. Neil Robertson
- Combinatorics  Prof. Ákos Seress
# Table of Contents

ACKNOWLEDGEMENTS ................................................................. ii  
VITA .................................................................................................... iii  
LIST OF TABLES ................................................................................ xi  
LIST OF SYMBOLS ........................................................................... xii  

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Introduction .......................................................... 1</td>
</tr>
<tr>
<td></td>
<td>1.1 Statement of the Basic Result ............................. 1</td>
</tr>
<tr>
<td></td>
<td>1.2 The Method Used in the Dissertation ..................... 3</td>
</tr>
<tr>
<td>II</td>
<td>Some Previous Results on λ-designs .......................... 4</td>
</tr>
<tr>
<td></td>
<td>2.1 Fisher Inequality ............................................... 4</td>
</tr>
<tr>
<td></td>
<td>2.2 The Ryser-Woodall Theorem ................................. 6</td>
</tr>
<tr>
<td></td>
<td>2.3 Elementary Relations Among the Parameters ............ 10</td>
</tr>
<tr>
<td></td>
<td>2.4 Some Characterizations of Type-1 λ-Designs .......... 30</td>
</tr>
<tr>
<td>III</td>
<td>The Case of Small Lambda ......................................... 31</td>
</tr>
<tr>
<td></td>
<td>3.1 The Basic Strategy ............................................. 31</td>
</tr>
<tr>
<td></td>
<td>3.2 Tests on the Basic Parameters ............................. 33</td>
</tr>
<tr>
<td></td>
<td>3.3 Construction of the Table ................................... 35</td>
</tr>
<tr>
<td></td>
<td>3.3.1 Estimates for $U$ ........................................... 35</td>
</tr>
<tr>
<td></td>
<td>3.3.2 Computing the $k_m - \lambda$ Values .................... 41</td>
</tr>
<tr>
<td></td>
<td>3.3.3 Estimates on the Number of Blocks of a Particular Size 46</td>
</tr>
<tr>
<td></td>
<td>3.4 Tests Eliminating Particular $k_m - \lambda$ Values .......... 50</td>
</tr>
</tbody>
</table>
3.5 Tests Eliminating a Line ............................................................... 55
3.6 Some Global Tests ........................................................................ 56
3.7 Tests Eliminating Solutions of the Diophantic Equations .......... 60

IV Implementation and Description of the Program ....................... 65

4.1 General Description of the Program ........................................... 65
  4.1.1 The Language ........................................................................ 65
  4.1.2 Overview of the Algorithm .................................................... 65
  4.1.3 The Data Structure ............................................................... 67

4.2 Utility Functions ........................................................................ 71
  4.2.1 Ceil ....................................................................................... 71
  4.2.2 Floor .................................................................................... 71
  4.2.3 RecursTestSum .................................................................... 72
  4.2.4 SubSumTest ......................................................................... 73
  4.2.5 MakeList ............................................................................. 74
  4.2.6 MakeFakeList ...................................................................... 75
  4.2.7 TouchList ............................................................................ 75
  4.2.8 ListRemove .......................................................................... 75
  4.2.9 SumPossibleNums ................................................................ 76
  4.2.10 IsSumPossible ..................................................................... 76
  4.2.11 IsSumPossibleLim .............................................................. 76
  4.2.12 FindUnique ......................................................................... 77
  4.2.13 FindNecess ......................................................................... 77

4.3 Top Level Functions .................................................................... 78
  4.3.1 PreProcess ........................................................................... 78
  4.3.2 Rounding ............................................................................. 79
  4.3.3 SubRecipSum ...................................................................... 80
  4.3.4 SubChangeMax ................................................................... 81
  4.3.5 ChangeMaxRed ................................................................... 82
  4.3.6 IsPairOK ............................................................................. 83
  4.3.7 IsShortOk ............................................................................ 83
  4.3.8 IsListOK ............................................................................. 84
  4.3.9 Weeding ............................................................................. 85
  4.3.10 CleanUp ............................................................................ 86
  4.3.11 Reduction ........................................................................... 86
  4.3.12 PrintDesign ....................................................................... 87
  4.3.13 Test ................................................................................... 88
  4.3.14 ReTest ............................................................................... 90
4.4 Tests Removing one $k_m - \lambda$ value .................................................. 90
  4.4.1 PrimeRed ................................................................. 90
  4.4.2 Trimming ................................................................. 91
  4.4.3 SubTrimming .......................................................... 91
  4.4.4 CommonRed ........................................................... 92
  4.4.5 RLConstRed ............................................................ 92
  4.4.6 RLSumRed .............................................................. 93
  4.4.7 MaxRed ................................................................. 94
  4.4.8 LastSumRed ............................................................ 94
  4.4.9 SubCardRed ............................................................ 95
  4.4.10 CardRed ............................................................... 95
  4.4.11 U2EitherRed .......................................................... 96
  4.4.12 SubEither ............................................................... 96
  4.4.13 U2PairRed ............................................................. 97

4.5 Tests Removing a Line .............................................................. 97
  4.5.1 RijU2inRed ............................................................... 97
  4.5.2 RijU1outRed ............................................................. 98
  4.5.3 ConstRed ............................................................... 99
  4.5.4 SubConstr .............................................................. 99
  4.5.5 SinRecipSum ............................................................ 99
  4.5.6 RecipRed ............................................................... 100

4.6 Global Tests ............................................................................ 100
  4.6.1 SolveGauss .............................................................. 101
  4.6.2 SquareRed .............................................................. 101
  4.6.3 GlobalSoluRed ........................................................ 102
  4.6.4 UniquePrime1Red ................................................... 103
  4.6.5 UniquePrime2Red ................................................... 103
  4.6.6 RLG lobalRed ........................................................ 104
  4.6.7 ListMax ................................................................. 105

4.7 Tests Eliminating Some Solutions of the Equations .................. 105
  4.7.1 Solu1 ................................................................. 105
  4.7.2 Solu2 ................................................................. 106
  4.7.3 Solu3 ................................................................. 106
  4.7.4 Solu4 ................................................................. 107
  4.7.5 Solu5 ................................................................. 107
  4.7.6 Solu6 ................................................................. 107
A.4 Tests Removing a $k_m - \lambda$ Value
   A.4.1 PrimeRed ................................................. 175
   A.4.2 Trimming ................................................. 176
   A.4.3 SubTrimming ............................................. 176
   A.4.4 CommonRed ............................................ 177
   A.4.5 RLConstRed ........................................... 179
   A.4.6 RLSumRed ................................................ 183
   A.4.7 MaxRed ..................................................... 184
   A.4.8 LastSumRed ............................................ 187
   A.4.9 SubCardRed ........................................... 188
   A.4.10 CardRed .................................................. 190
   A.4.11 U2EitherRed ......................................... 191
   A.4.12 SubEither .............................................. 192
   A.4.13 U2PairRed ............................................. 197

A.5 Tests Removing a Line ........................................ 203
   A.5.1 RijU2inRed .............................................. 203
   A.5.2 RijU1outRed ........................................... 204
   A.5.3 ConstRed .................................................. 206
   A.5.4 SubConstr ................................................ 207
   A.5.5 SinRecipSum ......................................... 207
   A.5.6 RecipRed .................................................. 210

A.6 Global Tests ................................................... 213
   A.6.1 SolveGauss .............................................. 213
   A.6.2 SquareRed ............................................... 214
   A.6.3 GlobalSoluRed ....................................... 218
   A.6.4 UniquePrime1Red ................................. 223
   A.6.5 UniquePrime2Red ................................... 225
   A.6.6 RLGloabalRed ......................................... 227
   A.6.7 ListMax .................................................... 232

A.7 Tests Eliminating Some Solutions of the Equations 233
   A.7.1 Solu1 ...................................................... 233
   A.7.2 Solu2 ...................................................... 235
   A.7.3 Solu3 ...................................................... 238
   A.7.4 Solu4 ...................................................... 242
   A.7.5 Solu5 ...................................................... 244
   A.7.6 Solu6 ...................................................... 246
B Tables .................................................. 249

BIBLIOGRAPHY ........................................... 252
## List of Tables

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The number of possible $(\lambda, d, \rho)$ triples</td>
</tr>
<tr>
<td>2</td>
<td>The number of $(\lambda, d, \rho)$ triples remaining after the first set of tests</td>
</tr>
<tr>
<td>3</td>
<td>The number of $(\lambda, d, \rho)$ triples remaining after the second set of tests</td>
</tr>
</tbody>
</table>
# List of Symbols

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>λ</td>
<td>2</td>
</tr>
<tr>
<td>ρ</td>
<td>11</td>
</tr>
<tr>
<td>A</td>
<td>28</td>
</tr>
<tr>
<td>B_i</td>
<td>2</td>
</tr>
<tr>
<td>C_1, C_2</td>
<td>10</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>c_1, c_2</td>
<td>10</td>
</tr>
<tr>
<td>I(p, B_i)</td>
<td>6</td>
</tr>
<tr>
<td>j_{011}, j_{212}, j_{021}, j_{022}</td>
<td>41</td>
</tr>
<tr>
<td>k_m</td>
<td>6</td>
</tr>
<tr>
<td>k'_{m}</td>
<td>12</td>
</tr>
<tr>
<td>k''_{m}</td>
<td>12</td>
</tr>
<tr>
<td>L</td>
<td>2</td>
</tr>
<tr>
<td>m_{11}, m_{12}, m_{21}, m_{22}</td>
<td>42</td>
</tr>
<tr>
<td>neg_{1}, neg_{2}</td>
<td>42</td>
</tr>
<tr>
<td>\hat{p}</td>
<td>6</td>
</tr>
<tr>
<td>pos_{1}, pos_{2}</td>
<td>42</td>
</tr>
<tr>
<td>R</td>
<td>13</td>
</tr>
<tr>
<td>R_{ij}</td>
<td>13</td>
</tr>
<tr>
<td>r_{ij}</td>
<td>20</td>
</tr>
<tr>
<td>r_{p}</td>
<td>7</td>
</tr>
<tr>
<td>r_1, r_2</td>
<td>8</td>
</tr>
<tr>
<td>s_1, s_2</td>
<td>28</td>
</tr>
<tr>
<td>s_{1(i)}, s_{2(i)}</td>
<td>42</td>
</tr>
<tr>
<td>t_{1}, t_{2}</td>
<td>28</td>
</tr>
<tr>
<td>U_i</td>
<td>19</td>
</tr>
<tr>
<td>U_{1}, U_{2}</td>
<td>31</td>
</tr>
<tr>
<td>u_0</td>
<td>46</td>
</tr>
<tr>
<td>V_i</td>
<td>20</td>
</tr>
</tbody>
</table>
CHAPTER I

Introduction

1.1 Statement of the Basic Result

We are interested in the study of set systems with the property that all pairwise intersections are of equal size \( \lambda \). The fundamental result in this area is the Generalized Fisher Inequality (see Theorem 2.1.1), which states that the number of sets (blocks) is at most as much as the cardinality of the underlying set.

We will concentrate on the extremal case, when the two numbers are equal. This can be divided into two subcases, when all sets are of the same size and when there are sets of different sizes.

Definition 1.1.1 A \((v,k,\lambda)\) symmetric block design is a pair \((X,L)\) such that

(a) \( X = \{1,2,\ldots,v\}, |L| = v, \) and the elements of \( L \) are subsets of \( X \). (The elements of \( X \) are called points, and the elements of \( L \) are blocks.)

(b) For all \( B_i, B_j \in L, i \neq j, |B_i \cap B_j| = \lambda \).

(c) For all \( B_j \in L, |B_j| = k > \lambda \).

Definition 1.1.2 A \( \lambda \)-design \( D \) is a pair \((X,L)\) such that
(a) $X = \{1, 2, \ldots, v\}, |L| = v$, and the elements of $L$ are subsets of $X$. (The elements of $X$ are called points, and the elements of $L$ are blocks.)

(b) For all $B_i, B_j \in L, i \neq j, |B_i \cap B_j| = \lambda$.

(c) For all $B_j \in L, |B_j| = k_j > \lambda$, and not all $k_j$ are equal.

$\lambda$-designs were first defined by Ryser [Ry] and Woodall [Wo70]. The only known examples are obtained from symmetric block designs by the following complementation procedure. Let $(X, S)$ be a symmetric $(v, k, \lambda')$ design with block set $S = \{C_1, C_2, \ldots, C_v\}$. Then

$$L = \{B \subset X : B = C_1 \text{ or } B = (C_i \setminus C_1) \cup (C_1 \setminus C_i) \text{ for some } 2 \leq i \leq v\}$$

is the block system of a $(k-\lambda')$-design. $\lambda$-designs obtained by this procedure are called type-1 designs. Type-1 designs have numerous special properties. For example there are only two different block sizes and only two different degrees in the design. (Degree meaning the number of blocks containing the given point.) The Ryser-Woodall Theorem (see Theorem 2.2.3) states that the second one, that only two different degrees possible, is true for any $\lambda$-design, not just for type-1 designs.

The main interest of this thesis is the $\lambda$-design conjecture [Ry], [Wo70], which states that all $\lambda$-designs are type-1. The conjecture was proven by deBruijn and Erdős [BE] for $\lambda = 1$, by Ryser [Ry] for $\lambda = 2$, by Bridges and Kramer [Br70], [Kr69], [BK], for $3 \leq \lambda \leq 9$, and by Seress [Se90] for $\lambda = 10$. Shrikhande and Singhi [SS76] showed the validity of the conjecture for prime $\lambda$. Seress [Se95] proved the conjecture for $\lambda = 2p$, $p$ prime.
Our main result is the following:

**Theorem 1.1.3** *The $\lambda$-design conjecture is true for $\lambda \leq 34$.*

### 1.2 The Method Used in the Dissertation

The method used is an extension of the technique in [Se90] and [Se95]. For a fixed value of $\lambda$ there are finitely many possible combinations of the other parameters. There are numerous inequalities and other conditions presented in this dissertation that must be satisfied by the parameters in order to possibly result in a non type-1 $\lambda$-design. These tests are incorporated into a computer program written in the GAP language [Sch].

Chapter II gives an overview of some previous results in the field of $\lambda$-designs. Chapter III introduces some new lemmas and tests used in our program. Chapter IV gives the implementation details of the program and the complete listing of the code. Chapter V contains some closing remarks.
CHAPTER II

Some Previous Results on $\lambda$-designs

2.1 Fisher Inequality

**Theorem 2.1.1** (Nonuniform Fisher Inequality) Let $B_1, \ldots, B_b$ be distinct subsets of a set of $v$ elements such that for every $i \neq j$, $|B_i \cap B_j| = \lambda$ where $1 \leq \lambda < v$. Then $b \leq v$.

*Proof.* The proof of this result is based on the one found by Majumdar [Ma] and rediscovered by Isbell [Is].

First let us consider the case when one of the sets has $\lambda$ elements. Then all other sets contain this set and are disjoint otherwise. Hence $b \leq v + 1 - \lambda \leq v$.

If there are no sets of size $\lambda$, then all numbers $\gamma_i = |B_i| - \lambda$ are positive. Let $M$ be the incidence matrix of the set system. The intersection condition can be expressed as

$$A = MM^T = \lambda J + C$$

where $J$ is the $m \times m$ all-one matrix and $C$ is the diagonal matrix $C = \text{diag}(\gamma_1, \ldots, \gamma_m)$.

Let us compute the determinant of $A$:
We can use the “bordering trick”, adding a first row of all ones and a first column of all zeros except in the first position. This change obviously will not affect the value of the determinant.

\[
\begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
0 & \lambda + \gamma_1 & \lambda & \cdots & \lambda \\
0 & \lambda & \lambda + \gamma_2 & \cdots & \lambda \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \lambda & \lambda & \cdots & \lambda + \gamma_m
\end{pmatrix}
\]  
(2.2)

Next we can subtract \( \lambda \) times the first row from all other rows.

\[
\begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
-\lambda & \gamma_1 & 0 & \cdots & 0 \\
-\lambda & 0 & \gamma_2 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
-\lambda & 0 & 0 & \cdots & \gamma_m
\end{pmatrix}
\]  
(2.3)

Finally using the diagonal entries we can kill all the ones in the first row except the leading one, bringing the matrix to a lower triangular form.

\[
\begin{pmatrix}
1 + \frac{\lambda}{\gamma_1} + \frac{\lambda}{\gamma_2} + \cdots + \frac{\lambda}{\gamma_m} & 0 & 0 & \cdots & 0 \\
-\lambda & \gamma_1 & 0 & \cdots & 0 \\
-\lambda & 0 & \gamma_2 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
-\lambda & 0 & 0 & \cdots & \gamma_m
\end{pmatrix}
\]  
(2.5)
From this form we can easily find the determinant:

$$\gamma_1 \gamma_2 \cdots \gamma_m \left(1 + \lambda \left(\frac{1}{\gamma_1} + \frac{1}{\gamma_2} + \cdots + \frac{1}{\gamma_m}\right)\right).$$

(2.6)

Since all $\gamma_i$'s are positive this shows, that the determinant of $A$ is positive, hence the rank of $A$ is $b$. Observing $b = rkA \leq rkM \leq v$ completes our proof. □

2.2 The Ryser–Woodall Theorem

From this point on, we are investigating $\lambda$-designs, where $X = \{1, 2, \ldots, v\}$, $L = \{B_1, B_2, \ldots, B_v\}$ is a set of subsets of $X$, $|L| = v$ and there is more than one different block size.

Let $V$ be the free $\mathbb{Q}$-vector space generated by the vectors $\hat{B}_1, \ldots, \hat{B}_v$, where $\hat{B}_i$ is a vector associated with $B_i$, i.e. $V = \{\sum_{i=1}^{v} c_i \hat{B}_i : c_i \in \mathbb{Q}\}$. To each point $p$ we assign

$$\hat{p} = \left\{ \sum B_i : p \in B_i \right\} = \sum_{B_i \in L} T(p, B_i) \hat{B}_i,$$

(2.7)

where $T(p, B_i) = \begin{cases} 1 & p \in B_i \\ 0 & \text{otherwise} \end{cases}$

First we prove some lemmas about the relations of the vectors $\hat{B}_i, \hat{p}$.

Lemma 2.2.1

$$\forall B_i \in L \quad \sum_{p \in B_i} \hat{p} = (k_i - \lambda) \hat{B}_i + \lambda \hat{L}$$

(2.8)

where $k_i = |B_i|$, and $\hat{L} = \sum_{B_i \in L} \hat{B}_i$.

Proof. By definition,

$$k_i = \sum_{p \in X} T(p, B_i).$$

(2.9)
Let us also observe that
\[ \sum_{p \in X} I^2(p, B_i) = \sum_{p \in X} I(p, B_i) = k_i \]  
(2.10)
and that if \( i \neq j \) then
\[ \sum_{p \in X} I(p, B_i)I(p, B_j) = \lambda. \]  
(2.11)

So
\[ \sum_{p \in \hat{B}_i} \hat{p} = \sum_{p \in X} I(p, B_i)\hat{p} = \sum_{p \in X} I(p, B_i) \left( \sum_{B_j \in L} I(p, B_j) \hat{B}_j \right) = \sum_{B_j \in L} \sum_{p \in \hat{B}_i} I(p, B_i)I(p, B_j) \hat{B}_j = \sum_{B_j \in L} \hat{B}_j \sum_{p \in X} I(p, B_i)I(p, B_j) = \hat{B}_i \sum_{p \in X} I^2(p, B_i) + \sum_{B_j \in L, B_i \neq B_j} \hat{B}_j \left( \sum_{p \in X} I(p, B_i)I(p, B_j) \right) = \hat{B}_i k_i + \sum_{B_j \in L, B_i \neq B_j} \lambda \hat{B}_j = (k_i - \lambda) \hat{B}_i + \lambda \hat{L}. \]  
(2.12)

\[ \square \]

**Lemma 2.2.2**

\[ \sum_{p \in X} (r_p - 1) \hat{p} = \lambda (v - 1) \hat{L} \]  
(2.13)

where \( r_p \) is the number of blocks containing \( p \).

**Proof.**

\[ \sum_{p \in X} (r_p - 1) \hat{p} = \sum_{p \in X} \left( \sum_{B_i \in L} I(p, B_i) - 1 \right) \left( \sum_{B_j \in L} I(p, B_j) \hat{B}_j \right) = \sum_{B_i \in L} \left( \sum_{p \in X} I(p, B_i) \sum_{B_j \in L} I(p, B_j) \hat{B}_j - \sum_{B_j \in L} I(p, B_j) \hat{B}_j \right) = \sum_{B_i \in L} \sum_{p \in X} I(p, B_i)I(p, B_j) \hat{B}_j - \sum_{B_j \in L} \sum_{p \in X} I(p, B_j) \hat{B}_j = \]  
(2.14)
The following theorem was independently proved by Ryser [Ry] and Woodall [Wo70].

**Theorem 2.2.3** If $D$ is a $\lambda$-design then there exist integers $r_1 > r_2$ such that $r_1 + r_2 = v + 1$, and every point occurs either in $r_1$ blocks or in $r_2$ blocks.

**Proof.** From Lemma 2.2.1 we get

$$
\frac{1}{k_i - \lambda} \sum_{p \in X} I(p, B_i) \hat{p} = \hat{B}_i + \frac{\lambda}{k_i - \lambda} \hat{L}.
$$

(2.16)

Summing over $B_i$,

$$
\sum_{B_i \in L} \sum_{p \in X} \frac{1}{k_i - \lambda} I(p, B_i) \hat{p} = \left(1 + \sum_{B_i \in L} \frac{\lambda}{k_i - \lambda}\right) \hat{L} = \left(1 + \sum_{B_i \in L} \frac{\lambda}{k_i - \lambda}\right) \hat{L} = \left(1 + \sum_{B_i \in L} \frac{\lambda}{k_i - \lambda}\right) \frac{1}{\lambda(v - 1)} \sum_{p \in X} (r_p - 1) \hat{p}.
$$

(2.17)

using Lemma 2.2.2

$$
\left(1 + \sum_{B_i \in L} \frac{\lambda}{k_i - \lambda}\right) \frac{1}{\lambda(v - 1)} \sum_{p \in X} (r_p - 1) \hat{p}.
$$

(2.18)

\{\hat{p} : p \in X\} is a basis of the vector-space, so for each $\hat{p}$, the coefficients on both sides must agree. Let us define $C_p = \sum_{B_i \in L} \frac{I(p, B_i)}{k_i - \lambda}$ and $c = \frac{1}{\lambda(v - 1)} \left(1 + \sum_{B_i \in L} \frac{\lambda}{k_i - \lambda}\right)$. We obtained

$$
C_p = (r_p - 1)c.
$$

(2.19)
Let \( p_0 \) be a distinguished point. We multiply equation (2.16) by \( I(p_0, B_i) \), and sum over \( B_i \):

\[
\sum_{p \in X} \sum_{B_i \in L} \frac{1}{k_i - \lambda} I(p, B_i) I(p_0, B_i) \hat{\phi} = \sum_{B_i \in L} (p_0, B_i) \hat{B}_i + \lambda \hat{\mathcal{L}} \sum_{B_i \in L} \frac{I(p_0, B_i)}{k_i - \lambda} = (2.20)
\]

using Lemma 2.2.2

\[
\hat{\phi}_0 + \frac{\lambda C_{p_0}}{\lambda (v - 1)} \sum_{p \in X} (r_p - 1) \hat{\phi}.
\]

(2.21)

The coefficient of \( \hat{\phi}_0 \) on the right-hand side of the equation is

\[
1 + \frac{C_{p_0}}{(v - 1)(r_{p_0} - 1)}.
\]

(2.22)

On the left-hand side of the equation the coefficient is \( C_{p_0} \). Hence

\[
C_{p_0} = 1 + \frac{C_{p_0}}{(v - 1)(r_{p_0} - 1)},
\]

(2.23)

or

\[
C_{p_0} \left(1 - \frac{r_{p_0} - 1}{v - 1}\right) = 1,
\]

(2.24)

or

\[
C_{p_0} \frac{v - r_{p_0}}{v - 1} = 1.
\]

(2.25)

From equation (2.19),

\[
c(r_{p_0} - 1)(v - r_{p_0}) = v - 1.
\]

(2.26)

So \( r_{p_0} \) is the root of a quadratic equation, whose coefficients depend only on \( D \):

\[
(v - x)(x - 1) = \frac{v - 1}{d},
\]

(2.27)
or

\[-x^2 + (v + 1)x - v - \frac{v - 1}{c} = 0.\]  \hspace{1cm} (2.28)

Therefore, there are only two possible values for the degree: \(r_1, r_2\), satisfying \(r_1 + r_2 = v + 1\). □

2.3 Elementary Relations Among the Parameters

Without a loss of generality we can assume that \(r_1 > r_2\). Let us denote the set of points with degree \(r_1\) by \(C_1\) and call them class 1 points; we denote the set of points with degree \(r_2\) by \(C_2\) and call them class 2 points. Let \(e_1 = |C_1|\) and \(e_2 = |C_2|\). By Theorem 2.2.3, we have

\[e_1 + e_2 = v = r_1 + r_2 - 1.\]  \hspace{1cm} (2.29)

Rearranging the two sides, we get

\[(r_1 - e_2) - (e_1 - r_2) = 1.\]  \hspace{1cm} (2.30)

Lemma 2.3.1

\[e_1r_1(r_1 - 1) + e_2r_2(r_2 - 1) = \lambda v(v - 1)\]  \hspace{1cm} (2.31)

Proof. Let us count the pairs of blocks containing a fixed point. There are \(v\) blocks, so there are \(\binom{v}{2}\) ways to pick the two blocks. Any two blocks intersect in \(\lambda\) points. On the other hand, there are \(e_1\) class-1 points, each of them are covered by \(r_1\) blocks.
So we can select the two blocks in \( \binom{r_1}{2} \) ways. Similarly, there are \( e_2 \) class-2 points, each of them are covered by \( r_2 \) blocks. The two blocks covering the point can be selected in \( \binom{r_2}{2} \) ways. Hence,

\[
e_1 \binom{r_1}{2} + e_2 \binom{r_2}{2} = \lambda \binom{v}{2}.
\]

Multiplying by 2, we get equation (2.31). □

Using (2.29) and (2.31) we have

\[
(e_1 - r_2)(r_1 - e_2) = \lambda(v - 1) - e_1 e_2.
\]

(2.33)

On the left-hand-side of (2.33), according to (2.30), there is a product of two consecutive integers, hence

\[
e_1 e_2 \leq \lambda(v - 1).
\]

(2.34)

Let us define

\[
\rho = \frac{r_1 - 1}{r_2 - 1} > 1.
\]

(2.35)

We can also define

\[
d = e_1 - r_2
\]

(2.36)

Notice that according to (2.30)

\[
r_1 - e_2 = d - 1.
\]

(2.37)

The parameters of a design are frequently expressed in terms of \( v, \lambda, \rho \). For our computations it will be more practical to use \( d, \lambda, \rho \), and express the other parameters...
in terms of these three. Using the definitions and the identities above, we find

\[ e_1 = \lambda + \frac{\lambda + d}{\rho}, \]

(2.38)

\[ e_2 = \lambda \rho - d \rho - \rho + \lambda, \]

(2.39)

\[ r_1 = \lambda \rho - d \rho - \rho + \lambda + d + 1, \]

(2.40)

\[ r_2 = \lambda - d + \frac{\lambda + d}{\rho}, \]

(2.41)

\[ \frac{r_1}{\rho + 1} = \frac{r_1(r_2 - 1)}{v - 1} = \lambda - (d + 1)\frac{\rho - 1}{\rho + 1}, \]

(2.42)

\[ \frac{r_1 \rho}{\rho + 1} = \frac{r_1^2 - r_1}{\rho - 1} = \lambda \rho - (d + 1)\frac{\rho - 1}{\rho + 1} = \lambda + e_2 \frac{\rho - 1}{\rho + 1}, \]

(2.43)

\[ \frac{r_2}{\rho + 1} = \frac{r_2^2 - r_2}{v - 1} = \lambda - d \frac{\rho - 1}{\rho + 1} = \lambda - e_1 \frac{\rho - 1}{\rho + 1}, \]

(2.44)

\[ \frac{r_2 \rho}{\rho + 1} = \frac{r_2(r_1 - 1)}{v - 1} = \lambda - d \frac{\rho - 1}{\rho + 1}. \]

(2.45)

Let \( k_m = |B_m| \). Denote by \( k'_m \) the number of class 1 points in \( B_m \) and by \( k^*_m \) the number of class 2 points. Obviously \( k_m = k'_m + k^*_m \).

**Lemma 2.3.2**

\[ k'_m = \lambda - \frac{k_m - 2\lambda}{\rho - 1} \]

(2.46)

\[ k^*_m = \lambda + \frac{(k_m - 2\lambda)\rho}{\rho - 1} \]

(2.47)
Proof. Let us count the points with multiplicity in the intersection of $B_m$ with other blocks. There are $v - 1$ more blocks, and any two blocks intersect in $\lambda$ points. On the other hand there are $k'_m$ points of degree $r_1$ and $k''_m$ points of degree $r_2$. Therefore

$$(v - 1)\lambda = k'_m(r_1 - 1) + k''_m(r_2 - 1). \quad (2.48)$$

We also know

$$k_m = k'_m + k''_m. \quad (2.49)$$

Solving the two equations for $k'_m$ and $k''_m$ we get (2.46) and (2.47). □

Let

$$R_{ij} = \sum_{\{m:i,j \in B_m\}} \frac{1}{k_m - \lambda}. \quad (2.50)$$

Then $R_{ij}$ depends only on the class of $i$ and $j$. Hence we can use the following notation: $R_{ij} = R(1)$ if $i = j$ and $i \in C_1$; $R_{ij} = R(1,1)$ if $i \neq j$ and $i,j \in C_1$; etc.

Lemma 2.3.3

(a)

$$R = \sum_{B_m \in L} \frac{1}{k_m - \lambda} = \rho + 2 + \frac{1}{\rho} - \frac{1}{\lambda}. \quad (2.51)$$

(b) For all $i \in C_1$,

$$R(1) = \sum_{\{B_m:i \in B_m\}} \frac{1}{k_m - \lambda} = 1 + \rho. \quad (2.52)$$
(c) For all \( i \in \mathcal{C}_2 \),
\[
R(2) = \sum_{\{B_m : i \in B_m, j \in B_m\}} \frac{1}{k_{ij} - \lambda} = 1 + \frac{1}{\rho}. \tag{2.53}
\]

(d) For all \( i, j \in \mathcal{C}_1 \),
\[
R(1, 1) = \sum_{\{B_m : i \in B_m, j \in B_m\}} \frac{1}{k_{ij} - \lambda} = \rho. \tag{2.54}
\]

(e) For all \( i \in \mathcal{C}_1, j \in \mathcal{C}_2 \),
\[
R(1, 2) = \sum_{\{B_m : i \in B_m, j \not\in B_m\}} \frac{1}{k_{ij} - \lambda} = 1. \tag{2.55}
\]

(f) For all \( i, j \in \mathcal{C}_2 \),
\[
R(2, 2) = \sum_{\{B_m : i \not\in B_m, j \in B_m\}} \frac{1}{k_{ij} - \lambda} = \frac{1}{\rho}. \tag{2.56}
\]

(g) For all \( i, j \in \mathcal{C}_1 \),
\[
R(1, \bar{1}) = \sum_{\{B_m : i \in B_m, j \not\in B_m\}} \frac{1}{k_{ij} - \lambda} = 1. \tag{2.57}
\]

(h) For all \( i \in \mathcal{C}_1, j \in \mathcal{C}_2 \),
\[
R(1, \bar{2}) = \sum_{\{B_m : i \in B_m, j \not\in B_m\}} \frac{1}{k_{ij} - \lambda} = \rho. \tag{2.58}
\]

(i) For all \( i \in \mathcal{C}_1, j \in \mathcal{C}_2 \),
\[
R(\bar{1}, 2) = \sum_{\{B_m : i \not\in B_m, j \in B_m\}} \frac{1}{k_{ij} - \lambda} = \frac{1}{\rho}. \tag{2.59}
\]

(j) For all \( i, j \in \mathcal{C}_2 \),
\[
R(2, \bar{2}) = \sum_{\{B_m : i \not\in B_m, j \not\in B_m\}} \frac{1}{k_{ij} - \lambda} = 1. \tag{2.60}
\]
(k) For all $i, j \in C_1$,

$$R(\bar{1}, \bar{1}) = \sum_{\{B_m : i \notin B_m, j \notin B_m\}} \frac{1}{k_m - \lambda} = 1 - \frac{1}{\lambda}. \quad (2.61)$$

(l) For all $i \in C_1, j \in C_2$,

$$R(\bar{1}, \bar{2}) = \sum_{\{B_m : i \notin B_m, j \notin B_m\}} \frac{1}{k_m - \lambda} = 1 - \frac{1}{\lambda}. \quad (2.62)$$

(m) For all $i, j \in C_2$,

$$R(\bar{2}, \bar{2}) = \sum_{\{B_m : i \notin B_m, j \notin B_m\}} \frac{1}{k_m - \lambda} = \rho - \frac{1}{\lambda}. \quad (2.63)$$

(n) For all $i \in C_1$,

$$R(\bar{1}) = \sum_{\{B_m : i \notin B_m\}} \frac{1}{k_m - \lambda} = 1 + \frac{1}{\rho} - \frac{1}{\lambda}. \quad (2.64)$$

(o) For all $i \in C_2$,

$$R(\bar{2}) = \sum_{\{B_m : i \notin B_m\}} \frac{1}{k_m - \lambda} = 1 + \rho - \frac{1}{\lambda}. \quad (2.65)$$

Proof.

(a) According to (2.19),

$$C_{p_0} = (r_{p_0} - 1) \frac{1}{\lambda (v - 1)} \left(1 + \sum_{B_i \in L} \frac{\lambda}{k_i - \lambda}\right). \quad (2.66)$$

Solving for $\sum_{B_i \in L} \frac{1}{k_i - \lambda}$ we get

$$\sum_{B_i \in L} \frac{1}{k_i - \lambda} = \frac{C_{p_0} (v - 1)}{r_{p_0} - 1} - \frac{1}{\lambda}. \quad (2.67)$$
Using (2.25), (2.29) and (2.35) we get

\[ \sum_{B_i \in L} \frac{1}{k_i - \lambda} = \frac{(v - 1)^2}{(v - r_{po})(r_p - 1)} - \frac{1}{\lambda} = (r_1 - 1)^2 + (r_2 - 1)^2 + 2(r_1 - 1)(r_2 - 1) \frac{1}{(r_1 - 1)(r_2 - 1)} - \frac{1}{\lambda} = \frac{r_1 - 1}{r_2 - 1} + \frac{r_2 - 1}{r_1 - 1} + 2 + \frac{1}{\lambda} = \rho + \frac{1}{\rho} + 2 - \frac{1}{\lambda}. \]  

(2.68)

(b) Let us take another look at equations (2.20) and (2.21):

\[ \sum_{p \in X} \sum_{B_i \in L} \frac{1}{k_i - \lambda} \mathcal{I}(p, B_i) \mathcal{I}(p_0, B_i) \dot{p} = \dot{p}_0 + \frac{\lambda C_{po}}{\lambda(v - 1)} \sum_{p \in X} (r_p - 1) \dot{p}. \]

(2.69)

According to (2.25),

\[ C_{po} = \frac{v - 1}{v - r_{po}}. \]

(2.70)

Let us check the coefficient of \( p \) on the two sides of (2.69). If \( p \neq p_0 \) then the coefficients on the two sides give us

\[ \sum_{p, p_0 \in B_i} \frac{1}{k_i - \lambda} = \frac{C_{po}}{v - 1} (r_p - 1) = \frac{r_p - 1}{v - r_{po}}. \]

(2.71)

Similarly, if \( p = p_0 \) then we find

\[ \sum_{p_0 \in B_i} \frac{1}{k_i - \lambda} = 1 + \frac{r_{po} - 1}{v - r_{po}}. \]

(2.72)

If \( p = p_0 \in C_1 \) then

\[ \sum_{p_0 \in B_i} \frac{1}{k_i - \lambda} = 1 + \frac{r_1 - 1}{v - r_1} = 1 + \rho. \]

(2.73)

(c) If \( p = p_0 \in C_2 \) then

\[ \sum_{p_0 \in B_i} \frac{1}{k_i - \lambda} = 1 + \frac{r_2 - 1}{v - r_2} = 1 + \frac{1}{\rho}. \]

(2.74)
(d) If \( p \neq p_0 \) and \( p, p_0 \in C_1 \) then

\[
\sum_{p, p_0 \in B_i} \frac{1}{k_i - \lambda} = \frac{r_1 - 1}{v - r_1} = \rho. \tag{2.75}
\]

(e) If \( p \in C_1 \) and \( p_0 \in C_2 \) then

\[
\sum_{p, p_0 \in B_i} \frac{1}{k_i - \lambda} = \frac{r_1 - 1}{v - r_2} = 1. \tag{2.76}
\]

(f) If \( p \neq p_0 \) and \( p, p_0 \in C_2 \) then

\[
\sum_{p, p_0 \in B_i} \frac{1}{k_i - \lambda} = \frac{r_2 - 1}{v - r_2} = \frac{1}{\rho}. \tag{2.77}
\]

(g)

\[
R(1, \bar{1}) = R(1) - R(1, 1) = 1. \tag{2.78}
\]

(h)

\[
R(1, \bar{2}) = R(1) - R(1, 2) = \rho. \tag{2.79}
\]

(i)

\[
R(\bar{1}, 2) = R(2) - R(1, 2) = \frac{1}{\rho}. \tag{2.80}
\]

(j)

\[
R(2, \bar{2}) = R(2) - R(2, 2) = 1. \tag{2.81}
\]

(k)

\[
R(\bar{1}, \bar{1}) = R - 2R(1) + R(1, 1) = \frac{1}{\rho} - \frac{1}{\lambda}. \tag{2.82}
\]
(l) \[ R(\bar{1}, \bar{2}) = R - R(1) - R(2) + R(1, 2) = 1 - \frac{1}{\lambda}. \quad (2.83) \]

(m) \[ R(\bar{2}, \bar{2}) = R - 2R(2) + R(2) = \rho - \frac{1}{\lambda}. \quad (2.84) \]

(n) \[ R(\bar{1}) = R - R(1) = 1 + \frac{\rho}{\lambda} - \frac{1}{\lambda}. \quad (2.85) \]

(o) \[ R(\bar{2}) = R - R(2) = 1 + \rho - \frac{1}{\lambda}. \quad (2.86) \]

Combining the inequality between the arithmetic and harmonic means and Lemma 2.3.3 we get the following inequalities:

\[ \sum_{ \{B_m: i \in B_m, i \in C_1 \} } \frac{(k_{m} - \lambda)}{r_1} \geq \frac{r_1}{\sum_{ \{B_m: i \in B_m, i \in C_1 \} } \frac{1}{k_{m} - \lambda}} = \frac{r_1}{\rho + 1} \quad (2.87) \]

which yields to

\[ \sum_{ \{B_m: i \in B_m, i \in C_1 \} } (k_{m} - \lambda) \geq \frac{r_1^2}{\rho + 1}. \quad (2.88) \]

\[ \sum_{ \{B_m: i \in B_m, i \in C_2 \} } \frac{(k_{m} - \lambda)}{r_2} \geq \frac{r_2}{\sum_{ \{B_m: i \in B_m, i \in C_2 \} } \frac{1}{k_{m} - \lambda}} = \frac{r_2^2}{1 + \frac{1}{\rho}} \quad (2.89) \]
which yields to

\[ \sum_{\{B_m:i\in B_m,i\in C_1\}} (k_m - \lambda) \geq \frac{r_2^2 \rho}{\rho + 1}. \]  \hspace{1cm} (2.90)

\[ \sum_{\{B_m:i\notin B_m,i\in C_1\}} \frac{(k_m - \lambda)}{r_2 - 1} \geq \frac{r_2 - 1}{\sum_{\{B_m:i\notin B_m,i\in C_1\}} \frac{1}{k_m - \lambda}} \frac{1}{1 + \frac{1}{\rho} - \frac{1}{\lambda}} \]  \hspace{1cm} (2.91)

which yields to

\[ \sum_{\{B_m:i\notin B_m,i\in C_1\}} (k_m - \lambda) \geq \frac{(r_2 - 1)^2}{1 + \frac{1}{\rho} - \frac{1}{\lambda}}. \]  \hspace{1cm} (2.92)

\[ \sum_{\{B_m:i\notin B_m,i\in C_2\}} \frac{(k_m - \lambda)}{r_1 - 1} \geq \frac{r_1 - 1}{\sum_{\{B_m:i\notin B_m,i\in C_2\}} \frac{1}{k_m - \lambda}} \frac{1}{1 + \rho - \frac{1}{\lambda}} \]  \hspace{1cm} (2.93)

which yields to

\[ \sum_{\{B_m:i\notin B_m,i\in C_2\}} (k_m - \lambda) \geq \frac{(r_1 - 1)^2}{1 + \rho - \frac{1}{\lambda}}. \]  \hspace{1cm} (2.94)

These inequalities suggest the introduction of the following quantities:

\[ U_i = \sum_{\{m:i\in B_m\}} \left( k_m - \lambda - \frac{r_1}{\rho + 1} \right), \quad i \in C_1, \]  \hspace{1cm} (2.95)

\[ U_i = \sum_{\{m:i\in B_m\}} \left( k_m - \lambda - \frac{r_2 \rho}{\rho + 1} \right), \quad i \in C_2, \]  \hspace{1cm} (2.96)

\[ x_1 = \frac{d(\lambda \rho - \lambda)}{\lambda \rho + \lambda - \rho}. \]  \hspace{1cm} (2.97)
\[ x_2 = \frac{(d + 1)(\lambda \rho - \lambda)}{\lambda \rho + \lambda - 1}, \quad (2.98) \]

\[ V_i = \sum_{\{B_m : i \notin B_m\}} (k_m - 2\lambda + x_1), \quad i \in C_1, \quad (2.99) \]

\[ V_i = \sum_{\{B_m : i \notin B_m\}} (k_m - 2\lambda + x_2), \quad i \in C_2. \quad (2.100) \]

Let us denote by \( r_{ij} \) the number of blocks containing the points \( i \) and \( j \).

**Lemma 2.3.4** Let \( i \in C_1 \). Then

(a)

\[ \sum_{j \in C_1, \ j \neq i} r_{ij} = \frac{r_1 (e_1 - 1) \rho}{\rho + 1} - \frac{U_i}{\rho - 1}, \quad (2.101) \]

or equivalently,

\[ \sum_{j \in C_1, \ j \neq i} \left( r_{ij} - \frac{r_1 \rho}{\rho + 1} \right) = - \frac{U_i}{\rho - 1}. \quad (2.102) \]

(b)

\[ \sum_{j \in C_2} r_{ij} = \frac{r_1 e_2}{\rho + 1} + \frac{U_i \rho}{\rho - 1}, \quad (2.103) \]

or equivalently,

\[ \sum_{j \in C_2} \left( r_{ij} - \frac{r_1}{\rho + 1} \right) = \frac{U_i \rho}{\rho - 1}. \quad (2.104) \]
\( (c) \)

\[ \sum_{j \neq i} r_{ij}^2 = r_i^2 \frac{\rho + \lambda - \rho}{\rho + 1} + U_i, \quad (2.105) \]

or equivalently,

\[ \sum_{j \in C_1 \setminus \{i\}} \left( r_{ij} - \frac{r_i \rho}{\rho + 1} \right)^2 + \sum_{j \in C_2 \setminus \{i\}} \left( r_{ij} - \frac{r_i \rho}{\rho + 1} \right)^2 = U_i. \quad (2.106) \]

**Proof.**

**(a)**

\[ \sum_{j \in C_1 \setminus \{i\}} \sum_{j \neq i} 1 = \sum_{\{m : i, j \in B_m\}} \sum_{j \neq i} 1 = \quad (2.107) \]

using the definition of \( k'_m \) and (2.46),

\[ \sum_{\{m : i \in B_m\}} (k'_m - 1) = \sum_{\{m : i \in B_m\}} \left( \lambda - \frac{k_m - 2\lambda}{\rho - 1} - 1 \right) = \quad (2.108) \]

\[ r_1 (\lambda - 1) - \frac{1}{\rho - 1} \sum_{\{m : i \in B_m\}} (k_m - \lambda) + \frac{r_1 \lambda}{\rho - 1} = \quad (2.109) \]

\[ r_1 \left( \lambda - 1 + \frac{\lambda}{\rho - 1} \right) - \frac{1}{\rho - 1} \sum_{\{m : i \in B_m\}} \left( k_m - \lambda - \frac{r_1}{\rho + 1} \right) - \frac{r_1}{\rho - 1} \frac{r_1}{\rho + 1} = \quad (2.110) \]

by (2.95) and (2.42),

\[ r_1 \left( \lambda - 1 + \frac{\lambda}{\rho - 1} - \frac{1}{\rho - 1} \left( \lambda - (d + 1) \frac{\rho - 1}{\rho + 1} \right) \right) - \frac{1}{\rho - 1} U_i = \quad (2.111) \]

and by (2.38) we get,

\[ \frac{r_1(e_1 - 1) \rho - U_i}{\rho + 1} = \quad (2.112) \]

Subtracting \( \frac{r_1(e_1 - 1) \rho}{\rho + 1} \) from both sides of the equation, we get (2.102).
(b) \[
\sum_{j \in C_2} \sum_{j \in C_2 : \{m \in B_m, j \in B_m\}} 1 = \sum_{\{m \in B_m\}} \sum_{\{j, j \in C_2, j \in B_m\}} 1 = (2.113)
\]
using the definition of \( k_m^* \) and (2.47),
\[
\sum_{\{m \in B_m\}} k_m^* = \sum_{\{m \in B_m\}} \left( \lambda + \frac{(k_m - 2\lambda)\rho}{\rho - 1} \right) = (2.114)
\]
\[
r_1 \lambda + \frac{\rho}{\rho - 1} \sum_{\{m \in B_m\}} (k_m - \lambda) - \frac{r_1 \lambda \rho}{\rho - 1} = (2.115)
\]
\[
r_1 \left( \lambda - \frac{\lambda \rho}{\rho - 1} \right) + \frac{\rho}{\rho - 1} \sum_{\{m \in B_m\}} \left( k_m - \lambda - \frac{r_1}{\rho + 1} \right) + \frac{r_1 \rho}{\rho - 1} \frac{r_1}{\rho + 1} = (2.116)
\]
by (2.95) and (2.43),
\[
r_1 \left( \lambda - \frac{\lambda \rho}{\rho - 1} + \frac{1}{\rho - 1} \left( \lambda \rho - (d + 1) \frac{\rho - 1}{\rho + 1} \right) \right) + \frac{\rho}{\rho - 1} U_i = (2.117)
\]
and by (2.39) we get,
\[
\frac{r_1 c_2}{\rho + 1} + \frac{U_i \rho}{\rho - 1}. (2.118)
\]
Subtracting \( \frac{r_1 c_2}{\rho + 1} \) from both sides of the equation, we get (2.104).

(c) Let \( B_m \) be a fixed block, such that \( i \in B_m \).
\[
\sum_{j \in B_m} \sum_{j \in B_m : \{n \notin B_n, j \in B_n\}} 1 = \sum_{\{n \notin B_n\}} \sum_{j \in B_n : j \notin B_m} 1 = (2.119)
\]
\[
(r_1 - 1)(\lambda - 1) + k_m - 1. (2.120)
\]
Hence,

\[ \sum_{j \neq i} r_{ij}^2 = \sum_{\{m : i \in B_m \}} \sum_{j \in B_m \setminus \{i\}} r_{ij} = \] (2.121)

\[ r_1((r_1 - 1)(\lambda - 1)) + \sum_{\{m : i \in B_m \}} (k_m - 1) = \] (2.122)

\[ r_1^2(\lambda - 1) + \sum_{\{m : i \in B_m \}} (k_m - \lambda) = \] (2.123)

by (2.95),

\[ r_1^2(\lambda - 1) + U_i + \frac{r_1^2}{\rho + 1} = r_1^2 \frac{\lambda \rho + \lambda - \rho}{\rho + 1} + U_i. \] (2.124)

To prove the second form:

\[ \sum_{j \in \mathcal{C}_1 \setminus \{i\}} \left( r_{ij} - \frac{r_1 \rho}{\rho + 1} \right)^2 + \sum_{j \in \mathcal{C}_2} \left( r_{ij} - \frac{r_1}{\rho + 1} \right)^2 = \] (2.125)

using (2.101) and (2.103) we have,

\[ \sum_{j \neq i} r_{ij}^2 - 2 \frac{r_1 \rho}{\rho + 1} \left( \frac{r_1(c_1 - 1)\rho}{\rho + 1} - \frac{U_i}{\rho - 1} \right) + (r_1 - 1) \left( \frac{r_1 \rho}{\rho + 1} \right)^2 \]

\[ -2 \frac{r_1}{\rho + 1} \left( \frac{r_1 e_2}{\rho + 1} + \frac{U_i \rho}{\rho - 1} \right) + r_2 \left( \frac{r_1}{\rho + 1} \right)^2 = \] (2.126)

canceling the \( U_i \) terms, substituting (2.105), (2.38) and (2.39),

\[ r_1^2 \frac{\lambda \rho + \lambda - \rho}{\rho + 1} + U_i - 2 \frac{r_1 \rho}{\rho + 1} \frac{r_1(\lambda + \Delta d - 1)\rho}{\rho + 1} + \]

\[ (r_1 - 1) \left( \frac{r_1 \rho}{\rho + 1} \right)^2 - 2 \frac{r_1}{\rho + 1} \frac{r_1(\lambda \rho - d \rho - \rho + \lambda)}{\rho + 1} + r_2 \left( \frac{r_1}{\rho + 1} \right)^2 \] (2.127)

after simplification every term cancels, except \( U_i \), proving (2.106).
Lemma 2.3.5 Let $i \in C_2$. Then

(a) 

\[ \sum_{j \in C_1} r_{ij} = \frac{r_2 e_{i+1}}{\rho + 1} - \frac{U_i}{\rho - 1}, \tag{2.128} \]

or equivalently,

\[ \sum_{j \in C_1} \left( r_{ij} - \frac{r_2 \rho}{\rho + 1} \right) = -\frac{U_i}{\rho - 1}. \tag{2.129} \]

(b) 

\[ \sum_{\substack{j \in C_2 \backslash j \neq i}} r_{ij} = \frac{r_2 (c_2 - 1)}{\rho + 1} + \frac{U_i \rho}{\rho - 1}, \tag{2.130} \]

or equivalently,

\[ \sum_{\substack{j \in C_2 \backslash j \neq i}} \left( r_{ij} - \frac{r_2}{\rho + 1} \right) = \frac{U_i \rho}{\rho - 1}. \tag{2.131} \]

(c) 

\[ \sum_{j \neq i} r_{ij}^2 = r_2^2 \frac{\lambda \rho + \lambda - 1}{\rho + 1} + U_i, \tag{2.132} \]

or equivalently,

\[ \sum_{j \in C_1} \left( r_{ij} - \frac{r_2 \rho}{\rho + 1} \right)^2 + \sum_{\substack{j \in C_2 \backslash j \neq i}} \left( r_{ij} - \frac{r_2}{\rho + 1} \right)^2 = U_i. \tag{2.133} \]

Proof.
\[ \sum_{j \in C_1} r_{ij} = \sum_{j \in C_1 \setminus \{m : i,j \in B_m\}} 1 = \sum_{\{m \in B_m\}} \sum_{\{j : j \in C_1, i,j \in B_m\}} 1 = \]  

using the definition of \( k_m' \) and (2.46),

\[ \sum_{\{m \in B_m\}} k_m' = \sum_{\{m \in B_m\}} \left( \lambda - \frac{k_m - 2\lambda}{\rho - 1} \right) = \]  

\[ r_2 \lambda - \frac{1}{\rho - 1} \sum_{\{m \in B_m\}} (k_m - \lambda) + \frac{r_2\lambda}{\rho - 1} = \]  

\[ r_2 \left( \lambda + \frac{\lambda}{\rho - 1} \right) - \frac{1}{\rho - 1} \sum_{\{m \in B_m\}} \left( k_m - \lambda - \frac{r_2\rho}{\rho + 1} \right) - \frac{r_2}{\rho - 1} \frac{r_2\rho}{\rho + 1} = \]  

by (2.95) and (2.42),

\[ r_2 \left( \lambda + \frac{\lambda}{\rho - 1} - \frac{\rho}{\rho - 1} \left( \frac{\lambda}{\rho} - \frac{d \rho - 1}{\rho + 1} \right) \right) - \frac{1}{\rho - 1} U_i = \]  

and by (2.38) we get,

\[ \frac{r_1c_1\rho}{\rho + 1} - \frac{U_i}{\rho - 1}. \]  

Subtracting \( \frac{r_1c_1\rho}{\rho + 1} \) from both sides of the equation, we get (2.129).

(b)

\[ \sum_{j \in C_2 \setminus i} \sum_{j \neq i} 1 = \sum_{\{m \in B_m\}} \sum_{\{j : j \in C_2, j \neq i, i,j \in B_m\}} 1 = \]  

\[ \sum_{j \in C_2 \setminus i} \sum_{j \neq i} 1 = \]  

\[ \sum_{\{m \in B_m\}} \sum_{\{j : j \in C_2, j \neq i, i,j \in B_m\}} 1 = \]  

\[ (2.140) \]
using the definition of $k_m^*$ and (2.47),

\[ \sum_{\{m : i \in B_m\}} (k_m^* - 1) = \sum_{\{m : i \in B_m\}} \left( \lambda + \frac{(k_m - 2\lambda)}{\rho - 1} - 1 \right) = \] (2.141)

\[ r_2(\lambda - 1) + \frac{\rho}{\rho - 1} \sum_{\{m : i \in B_m\}} (k_m - \lambda) - \frac{r_2\lambda\rho}{\rho - 1} = \] (2.142)

\[ r_2 \left( \lambda - 1 - \frac{\lambda\rho}{\rho - 1} \right) + \frac{\rho}{\rho - 1} \sum_{\{m : i \in B_m\}} \left( k_m - \lambda - \frac{r_2\rho}{\rho + 1} \right) + \frac{r_2\rho}{\rho - 1}\frac{r_2\rho}{\rho + 1} = \] (2.143)

by (2.96) and (2.45),

\[ r_2 \left( \lambda - 1 - \frac{\lambda\rho}{\rho - 1} + \frac{\rho}{\rho - 1} \left( \lambda - \frac{d\rho - 1}{\rho + 1} \right) \right) + \frac{\rho}{\rho - 1} U_i = \] (2.144)

and by (2.39) we get,

\[ \frac{r_2(c_2 - 1)}{\rho + 1} + \frac{U_i\rho}{\rho - 1}. \] (2.145)

Subtracting $\frac{r_2(c_2 - 1)}{\rho + 1}$ from both sides of the equation, we get (2.131).

(c) Let $B_m$ be a fixed block, such that $i \in B_m$.

\[ \sum_{j \in B_m} r_{ij} = \sum_{j \in B_m} \sum_{\{n : i, j \in B_n\}} 1 = \sum_{\{n : i \in B_n\}} \sum_{j \in B_n \cap B_m} 1 = \] (2.146)

\[ (r_2 - 1)(\lambda - 1) + k_m - 1. \] (2.147)

Hence,

\[ \sum_{j \neq i} r_{ij}^2 = \sum_{\{m : i \in B_m\}} \sum_{j \in B_m \setminus \{i\}} r_{ij} = \] (2.148)
\[ r_2((r_2 - 1)(\lambda - 1)) + \sum_{\{m: n \in B_m\}} (k_m - 1) = \]  \hspace{1cm} (2.149)

\[ r_2^2(\lambda - 1) + \sum_{\{m: n \in B_m\}} (k_m - \lambda) = \]  \hspace{1cm} (2.150)

by (2.95),

\[ r_2^2(\lambda - 1) + U_i + \frac{r_2^2 \rho}{\rho + 1} = r_2^2 \frac{\lambda \rho + \lambda - 1}{\rho + 1} + U_i. \]  \hspace{1cm} (2.151)

To prove the second form:

\[ \sum_{j \in C_1} \left( r_{ij} - \frac{r_2 \rho}{\rho + 1} \right)^2 + \sum_{j \in C_2, j \neq i} \left( r_{ij} - \frac{r_2}{\rho + 1} \right)^2 = \]  \hspace{1cm} (2.152)

using (2.128) and (2.130) we have,

\[ \sum_{j \neq i} r_{ij}^2 - 2 \frac{r_2 \rho}{\rho + 1} \left( \frac{r_2 e_1 \rho}{\rho + 1} - \frac{U_i}{\rho - 1} \right) + r_1 \left( \frac{r_2 \rho}{\rho + 1} \right)^2 \]
\[ -2 \frac{r_2}{\rho + 1} \left( \frac{r_1 (e_2 - 1)}{\rho + 1} + \frac{U_i \rho}{\rho - 1} \right) + (r_2 - 1) \left( \frac{r_2}{\rho + 1} \right)^2 = \]  \hspace{1cm} (2.153)

canceling the \( U_i \) terms, substituting (2.132), (2.38) and (2.39),

\[ r_2^2 \frac{\rho + \lambda - 1}{\rho + 1} + U_i - 2 \frac{r_2 \rho}{\rho + 1} \frac{r_2(\lambda + \frac{\lambda + d}{\rho}) \rho}{\rho + 1} + r_1 \left( \frac{r_2 \rho}{\rho + 1} \right)^2 \]
\[ -2 \frac{r_2}{\rho + 1} \frac{r_2(\lambda \rho - d \rho - \rho + \lambda - 1)}{\rho + 1} + (r_2 - 1) \left( \frac{r_2}{\rho + 1} \right)^2 \]  \hspace{1cm} (2.154)

after simplification every term cancels, except \( U_i \), proving (2.133).

\[ \square \]

**Theorem 2.3.6 [Se89]**

(a) Let \( i \in C_1 \). Then more than half of the numbers \( \{ r_{ij} : j \in C_2 \} \) are equal to \( \left\lfloor \frac{r_1 (r_2 - 1)}{v - 1} \right\rfloor \).
(b) Let \( i \in C_2 \). Then more than half of the numbers \( \{ r_{ij} : j \in C_2 \} \) are equal to \( \left\lceil \frac{r_2(r_2 - 1)}{r - 1} \right\rceil \).

In [SS76] Singhi and Shrikhande proved the following:

**Theorem 2.3.7** If \( \lambda > 1 \), \( \rho = x/y \), \( \gcd(x, y) = 1 \), then \( 0 < x - y < \lambda \) and \( y < \lambda \).

Let us define

\[
A = \frac{x - y}{\gcd(\lambda, x - y)}.
\]

In [Se95] Seress showed that

**Lemma 2.3.8** For all \( i, j \in X \), \( i \neq j \), \( r_{ij} = \lambda + m_{ij}A \) for some integer \( m_{ij} \).

In [Se90] Seress proved:

**Theorem 2.3.9** \( d \geq -\frac{1}{2}\lambda \rho - \frac{1}{2}\lambda + \frac{\lambda(p-1)}{4\lambda - 2} \).

Let us define four new quantities:

\[
t_1 = \left\lceil \frac{r_1}{\rho + 1} \right\rceil - \frac{r_1}{\rho + 1},
\]

\[
t_2 = \left\lceil \frac{r_2}{\rho + 1} \right\rceil - \frac{r_2}{\rho + 1},
\]

\[
s_1 = \left\lceil \frac{\lambda - x_1}{\rho} \right\rceil - \frac{\lambda - x_1}{\rho},
\]

\[
s_2 = [\lambda - x_2] - (\lambda - x_2).
\]
Lemma 2.3.10 [Wo71], [Sc89]

\[ t_1 + t_2 = \frac{\rho - 1}{\rho + 1}. \]  

(2.160)

Lemma 2.3.11 [Sc90]

\[ s_1 + t_1 = \frac{\rho - 1}{\rho + 1} \frac{\lambda \rho + \lambda - \rho + d}{\lambda \rho + \lambda - \rho}. \]  

(2.161)

Lemma 2.3.12 [Sc90]

\[ s_2 + t_2 = \frac{\rho - 1}{\rho + 1} \frac{\lambda \rho + \lambda + d}{\lambda \rho + \lambda - 1}. \]  

(2.162)

Lemma 2.3.13 [Sc90]

\[ \frac{\rho}{\rho - 1} \sum_{i \in C_1} U_i + \frac{1}{\rho - 1} \sum_{i \in C_2} U_i = e_1 e_2 \frac{\rho - 1}{\rho + 1}. \]  

(2.163)

Proof. Summing the two sides of equation (2.103) for all \( i \in C_1 \) and the two sides of (2.128) for all \( j \in C_2 \) we get two different counts of the \( r_{ij} \) values, for pairs with one class-1 and one class-2 point. Setting the two right hand sides equal, we get (2.163). □

Lemma 2.3.14 [Sc90]

For all \( i \in C_1 \),

\[ U_i + V_i = \frac{(\rho - 1)^2 (e_1 - 1)e_2}{(\rho + 1)(\lambda \rho + \lambda - \rho)}. \]  

(2.164)

Lemma 2.3.15 For all \( i \in C_2 \),

\[ U_i + V_i = \frac{(\rho - 1)^2 (e_2 - 1)e_1}{(\rho + 1)(\lambda \rho + \lambda - 1)}. \]  

(2.165)
2.4 Some Characterizations of Type-1 $\lambda$-Designs

In [SS76] Singhi and Shrikhande proved the following:

**Theorem 2.4.1** The $\lambda$-design conjecture is true when $\lambda$ is a prime number.

In [Se95] Seress showed the following two theorems:

**Theorem 2.4.2** If $\gcd(\lambda, x - y) = 2$ then the design is type-1.

**Theorem 2.4.3** If $\gcd(\lambda, x - y) = \frac{\lambda}{2}$ then the design is type-1.

In [SS84] Singhi and Shrikhande showed:

**Theorem 2.4.4** If $\gcd(\lambda, x - y) = 1$ then the design is type-1.

An immediate corollary of the previous three is:

**Theorem 2.4.5** The $\lambda$-design conjecture is true when $\lambda$ is twice a prime number.

In [Kr74] Kramer proved that

**Theorem 2.4.6** If $\rho \geq \lambda - 1$ then the design is type-1.

In [Wo71] and [Se89] it is shown that

**Theorem 2.4.7** $D$ is type-1 if and only if $\frac{t_x - r_1}{v-1}$ or $\frac{t_x - r_2}{v-1}$ is integer.

The above is equivalent to: $D$ is type-1 if and only if $\frac{r_1}{\rho + 1}$ or $\frac{r_2}{\rho + 1}$ is integer.

**Theorem 2.4.8** $D$ is type-1 if and only if $\frac{\lambda - r_1}{\rho}$ or $x_2$ is an integer.
CHAPTER III

The Case of Small Lambda

3.1 The Basic Strategy

In this chapter we describe the method used to prove the conjecture for a particular \( \lambda \) value.

Let us first observe that by Lemma 3.2.2 and Theorems 2.3.7 and 2.3.9, for a fixed \( \lambda \) value there are finitely many possible \((\lambda, d, \rho)\) triples. There are 565,816 triples \((\lambda, d, \rho)\) with \(2 < \lambda < 34\). First we can list these and check some simple conditions that the parameters of non type–1 designs have to satisfy. For example, \((2.38)-(2.41)\) have to give integer values for \(e_1, e_2, r_1, r_2\); by Theorem 2.4.7, \(r^*_i\) and \(r^*_j\) should not be integers; etc. After these tests, 311 triples \((\lambda, d, \rho)\) remain with \(2 < \lambda < 34\). For the remaining triples, our main tool is Lemma 2.3.13. In Subsection 3.3.1, we give estimates for \(U_i\); if these estimates contradict \((2.163)\), then the appropriate triple \((\lambda, d, \rho)\) can be eliminated. There are 48 triples which survive this test. For these, we list the possible \(U\) values. For each fixed \(U\), in Subsection 3.3.2 we compute a list of possible block sizes for blocks which contain a point \(i\) with \(U = U_i\), and a list of possible block sizes for blocks which do not contain \(i\). We will refer to the \(U\) values of class–1 and class–2 points as \(U_1\) and \(U_2\) values, respectively. The lists of possible
block sizes are called the in-list and out-list of $U$. For practical reasons, instead of listing the block sizes ($k_m$ values), we will list the $k_m - \lambda$ values, and call these $kml$ values for short. We are going to refer to these lists as $kml$-lists. We will call a $U$ value and the two associated $kml$-lists a line. The set of lines is called the table belonging to $(\lambda, d, \rho)$.

In Subsection 3.3, we also give estimates for the multiplicities of blocks of size $k_m$ that can contain a point $i$ with $U = U_i$.

Our next goal is to eliminate some $U$ values from the table we built in Section 3.3.3, until we get contradiction with (2.163). The main tool here is Lemma 2.3.3. We check some number theoretical conditions the $k_m - \lambda$ values must satisfy, so that the reciprocal sums are as required in Lemma 2.3.3. These tests are described in Sections 3.4 and 3.5. In Section 3.6, we describe some global tests the design must satisfy (for example, $\det(MM^T)$, as defined in the proof of Theorem 2.1.1, must be a square number). These global tests can eliminate a triple $(\lambda, d, \rho)$ immediately. For the remaining triples, we consider a fixed value $U$ and the corresponding in-list. For a point $i$ with $U = U_i$, we know the sum and reciprocal sum of the $k_m - \lambda$ values for the blocks containing $i$, and the number of such blocks. This gives three Diophantic equations for the block sizes, and we compute all possible solutions. Section 3.7 describes some tests that eliminate some of these solutions.

We apply these tests and reductions repeatedly until either (in the desired case) the design is eliminated or (in the unlucky case) none of the tests find contradictions and none of the reductions yield any change in the table.
Since these tests and reductions involve a very large amount of computations, these were done with a computer program. Chapter IV gives the implementation details of the program and Appendix A contains the complete listing of the program.

3.2 Tests on the Basic Parameters

We start with one particular $\lambda$ value. First we can list all the possible $(\lambda, d, \rho)$ triples. As it was mentioned in section 3.1, there are finitely many such triples. For fixed $\lambda$, Theorem 2.3.7 tells us the list of possible $\rho$ values. $d$ is an integer by definition, and Theorem 2.3.9 gives a lower estimate for its value. The upper estimate for $d$ is given by the following two lemmas.

**Lemma 3.2.1** Let $D$ be a non type–1 design of the smallest possible $\lambda$ value. Then all block sizes are at least $2\lambda$.

*Proof.* Let us assume that the non type–1 $\lambda$–design with the minimal $\lambda$ value contains some block, say $B_m$, that has a size $k_m < 2\lambda$. Then we can construct a new design by keeping $B_m$, and replacing all other blocks by their symmetric difference with $B_m$. The resulting new design is clearly a non type–1 $\lambda$–design, and any two blocks intersect in $k_m - \lambda$ points. Hence if $k_m < 2\lambda$, the new design has a lower $\lambda$ value, contradicting our assumption. □

Based on the Lemma above, from here on we assume that all block sizes are at least $2\lambda$.

**Lemma 3.2.2** In a non type–1 $\lambda$–design $d \leq -2$. 
Proof. According to equation (2.52):

\[
\sum_{\{B_m : i \in B_m, t \in C_t\}} \frac{1}{k_m - \lambda} = 1 + \rho.
\]  

(3.1)

On the left hand side there are \( r_1 \) terms, and each denominator is at least \( \lambda \). Therefore

\[
\frac{r_1}{\lambda} \geq \sum_{\{B_m : i \in B_m, t \in C_t\}} \frac{1}{k_m - \lambda} = 1 + \rho.
\]  

(3.2)

Multiplying by \( \lambda \) and expanding \( r_1 \), using equation (2.40), we find,

\[
\lambda \rho - d \rho - \rho + \lambda + d + 1 \geq \lambda \rho + \lambda.
\]  

(3.3)

Factoring the above inequality we get,

\[-(d + 1)(\rho - 1) \geq 0.\]

(3.4)

Since \( \rho > 1 \), it means that \( d \leq -1 \). Let us notice that \( d = -1 \) is equivalent to \( \frac{1}{\rho + 1} = \lambda \) and according to Theorem 2.4.7 this means that the design is type-1. Hence for non type-1 designs \( d \leq -2 \). □

At this point we have a finite list of \((\lambda, d, \rho)\) triples. We can eliminate all the cases where \( \lambda \) is a prime or twice a prime, based on Theorems 2.4.1 and 2.4.5. We can delete all the cases where \( \gcd(\lambda, x - y) \in \{1, 2, \frac{1}{2}\} \), since according to Theorems 2.4.2, 2.4.3 and 2.4.4 those designs are type-1. Next we check if \( \rho < \lambda - 1 \); otherwise according to Theorem 2.4.6 we have a type-1 design. Then we compute \( e_1 \) and \( e_2 \) using (2.38) and (2.39) and make sure that both of them are integers. We compute \( r_1 \) and \( r_2 \) using (2.40) and (2.41). Then we make sure that they are integers and that the conditions of Theorem 2.4.7 are not met. Next we compute \( t_1, t_2, s_1, s_2 \), according
to (2.156)-(2.159). We check if these values satisfy the conditions of Lemmas 2.3.10–2.3.12. We also check the conditions of Theorem 2.4.8. Finally we check the conditions of Lemma 2.3.8, namely that the difference of $\left\lfloor \frac{r_2 \rho}{\rho+1} \right\rfloor$ and $\lambda$ is a multiple of $A$. Note that by Theorem 2.3.6, there are $i, j$ with $r_{ij} = \left\lfloor \frac{r_2 \rho}{\rho+1} \right\rfloor$.

The tests and computations described above are implemented in the function PreProcess. (See section 4.3.1)

Table 2 in Appendix B, shows how many parameter triples survive the tests above for the first few $\lambda$ values.

### 3.3 Construction of the Table

For each of the remaining parameter triples we build a table described in Section 3.1. The construction of the table is implemented in the function Test. (see 4.3.13)

#### 3.3.1 Estimates for $U$

First we have to obtain some lower and upper estimates for the $U$ values.

The estimates in Lemmas 3.3.1–3.3.8 are based on equations (2.106) and (2.133). They are extensions of Lemma 4.1 of [Se95].

**Lemma 3.3.1** Let $D$ be a non type-1 $\lambda$-design. Then for all $i \in C_1$,

$$U_i \geq \frac{(\rho - 1)(At_1(e_2 - e_1 + 1) - (v - 1)t_1^2)}{(\rho - 1)(1 + A) - 2t_1(\rho + 1)}. \quad (3.5)$$
Proof. Recall that $A$ is defined in equation (2.155). Let us observe, that for an integer $x$, that is a multiple of $A$, $x^2 + A|x| \geq 0$ holds. Based on this we know that

\begin{align}
\sum_{j \in C_1, i \neq j} \left( \left( r_{ij} - \frac{r_1 \rho}{\rho + 1} + t_1 \right)^2 + A \left( r_{ij} - \frac{r_1 \rho}{\rho + 1} + t_1 \right) \right) + \\
\sum_{j \in C_2} \left( \left( r_{ij} - \frac{r_1}{\rho + 1} - t_1 \right)^2 + A \left( r_{ij} - \frac{r_1}{\rho + 1} - t_1 \right) \right) \geq 0
\end{align}

Solving the inequality for $U_i$, we obtain (3.5). □

Lemma 3.3.2 Let $D$ be a non type-1 $\lambda$-design. Then for all $i \in C_1$,

$$U_i \geq \frac{(\rho - 1)(At_1(e_1 + e_2 - 1) - (v - 1)t_1^2)}{(\rho - 1) + (A - 2t_1)(\rho + 1)}.$$  (3.7)

Proof.

\begin{align}
\sum_{j \in C_1, i \neq j} \left( \left( r_{ij} - \frac{r_1 \rho}{\rho + 1} + t_1 \right)^2 - A \left( r_{ij} - \frac{r_1 \rho}{\rho + 1} + t_1 \right) \right) + \\
\sum_{j \in C_2} \left( \left( r_{ij} - \frac{r_1}{\rho + 1} - t_1 \right)^2 + A \left( r_{ij} - \frac{r_1}{\rho + 1} - t_1 \right) \right) \geq 0
\end{align}

Solving the inequality for $U_i$, we obtain (3.7). □

Lemma 3.3.3 Let $D$ be a non type-1 $\lambda$-design. Then for all $i \in C_1$,

$$U_i \leq \frac{(\rho - 1)(At_1(e_1 + e_2 - 1) + (v - 1)t_1^2)}{(A + 2t_1)(\rho + 1) - (\rho - 1)}.$$  (3.9)

Proof.

\begin{align}
\sum_{j \in C_1, i \neq j} \left( \left( r_{ij} - \frac{r_1 \rho}{\rho + 1} + t_1 \right)^2 + A \left( r_{ij} - \frac{r_1 \rho}{\rho + 1} + t_1 \right) \right) + \\
\sum_{j \in C_2} \left( \left( r_{ij} - \frac{r_1}{\rho + 1} - t_1 \right)^2 - A \left( r_{ij} - \frac{r_1}{\rho + 1} - t_1 \right) \right) \geq 0
\end{align}

Solving the inequality for $U_i$, we obtain (3.9). □
Lemma 3.3.4 Let $D$ be a non type-1 $\lambda$-design. Then for all $i \in C_1$,

$$U_i \leq \frac{(\rho - 1)((v - 1)\ell) - A\ell (e_1 - e_2 - 1)}{(\rho - 1)(A - 1) + 2\ell (\rho + 1)}.$$  \hfill (3.11)

Proof.

$$
\sum_{j \in C_1, i \neq j} \left( \left( r_{ij} - \frac{r_1}{\rho + 1} + \ell \right)^2 - A \left( r_{ij} - \frac{r_1}{\rho + 1} + \ell \right) \right) + \\
\sum_{j \in C_2} \left( \left( r_{ij} - \frac{r_1}{\rho + 1} - \ell \right)^2 - A \left( r_{ij} - \frac{r_1}{\rho + 1} - \ell \right) \right) \geq 0
$$  \hfill (3.12)

Solving the inequality for $U_i$, we obtain (3.11). $\square$

Lemma 3.3.5 Let $D$ be a non type-1 $\lambda$-design. Then for all $i \in C_2$,

$$U_i \geq \frac{(\rho - 1)(A\ell (e_2 - e_1 - 1) - (v - 1)\ell^2)}{(\rho - 1)(1 + A) - 2\ell (\rho + 1)}.$$  \hfill (3.13)

Proof.

$$
\sum_{j \in C_1} \left( \left( r_{ij} - \frac{r_2}{\rho + 1} + \ell \right)^2 + A \left( r_{ij} - \frac{r_2}{\rho + 1} + \ell \right) \right) + \\
\sum_{j \in C_2, i \neq j} \left( \left( r_{ij} - \frac{r_2}{\rho + 1} - \ell \right)^2 + A \left( r_{ij} - \frac{r_2}{\rho + 1} - \ell \right) \right) \geq 0
$$  \hfill (3.14)

Solving the inequality for $U_i$, we obtain (3.13). $\square$

Lemma 3.3.6 Let $D$ be a non type-1 $\lambda$-design. Then for all $i \in C_2$,

$$U_i \geq \frac{(\rho - 1)(A\ell (e_1 + e_2 - 1) - (v - 1)\ell^2)}{(\rho - 1) + (A - 2\ell)(\rho + 1)}.$$  \hfill (3.15)

Proof.

$$
\sum_{j \in C_1} \left( \left( r_{ij} - \frac{r_2}{\rho + 1} + \ell \right)^2 - A \left( r_{ij} - \frac{r_2}{\rho + 1} + \ell \right) \right) + \\
\sum_{j \in C_2, i \neq j} \left( \left( r_{ij} - \frac{r_2}{\rho + 1} - \ell \right)^2 + A \left( r_{ij} - \frac{r_2}{\rho + 1} - \ell \right) \right) \geq 0
$$  \hfill (3.16)

Solving the inequality for $U_i$, we obtain (3.15). $\square$
Lemma 3.3.7 Let $D$ be a non type-1 $\lambda$-design. Then for all $i \in C_2$,

$$U_i \leq \frac{(\rho - 1)(At_2(e_1 + e_2 - 1) + (v - 1)t_2^2)}{(A + 2t_2)(\rho + 1) - (\rho - 1)}. \quad (3.17)$$

Proof.

$$\sum_{j \in C_1} \left( \left( r_{ij} - \frac{r_2}{\rho + 1} + t_2 \right)^2 + A \left( r_{ij} - \frac{r_2}{\rho + 1} + t_2 \right) \right) +$$

$$\sum_{j \in C_2, i \neq j} \left( \left( r_{ij} - \frac{r_2}{\rho + 1} - t_2 \right)^2 - A \left( r_{ij} - \frac{r_2}{\rho + 1} - t_2 \right) \right) \geq 0 \quad (3.18)$$

Solving the inequality for $U_i$, we obtain (3.17). □

Lemma 3.3.8 Let $D$ be a non type-1 $\lambda$-design. Then for all $i \in C_2$,

$$U_i \leq \frac{(\rho - 1)((v - 1)t_2^2 - At_2(e_1 - e_2 + 1))}{(A - 1)(\rho - 1) + 2t_2(\rho + 1)}. \quad (3.19)$$

Proof.

$$\sum_{j \in C_1} \left( \left( r_{ij} - \frac{r_2}{\rho + 1} + t_2 \right)^2 - A \left( r_{ij} - \frac{r_2}{\rho + 1} + t_2 \right) \right) +$$

$$\sum_{j \in C_2, i \neq j} \left( \left( r_{ij} - \frac{r_2}{\rho + 1} - t_2 \right)^2 - A \left( r_{ij} - \frac{r_2}{\rho + 1} - t_2 \right) \right) \geq 0 \quad (3.20)$$

Solving the inequality for $U_i$, we obtain (3.19). □

Lemma 3.3.9

$$k_m = 2\lambda + c(x - y) \quad (3.21)$$

where $c$ is a non-negative integer.
Proof. According to Lemma 3.2.1, \( k_m \geq 2\lambda \). Using (2.46) and that \( \rho = \frac{x}{y} \), we get

\[
k'_m = \lambda - \frac{y(k_m - 2\lambda)}{x - y}.
\]

(3.22)

Since in this equation all the variables are integers and \( \gcd(y, x - y) = 1 \), \( k_m - 2\lambda \) must be a multiple of \( x - y \). □

The definition of \( U_i \) (equations (2.95) and (2.96)) provide another estimate for \( U_i \). Recall that the motivation behind defining \( U_i \) was the arithmetic-harmonic mean of the \( k_m - \lambda \) values for blocks containing \( i \). As Lemma 3.3.9 shows, there are gaps among the possible \( k_m - \lambda \) values. Utilizing that, we can give a lower estimate for the difference between the arithmetic and harmonic mean.

The following is Lemma 4.3 from [Se95], and the next three are analogous to it.

Lemma 3.3.10 Let \( D \) be a non type-1 \( \lambda \)-design. Let \( C = (x - y) \left[ \frac{r_1 - \lambda}{x - y} \right] \). Then for all \( i \in C_1 \),

\[
U_i \geq (1 + \rho) \left( \frac{r_1}{\rho + 1} - \lambda - C \right) \left( \lambda + C + x - y - \frac{r_1}{\rho + 1} \right).
\]

(3.23)

Proof. If \( U_i \geq r_1 \left( \lambda + C + x - y - \frac{r_1}{\rho + 1} \right) \) then the inequality clearly holds. Otherwise by (2.95)

\[
\sum_{\{m : i \in B_m\}} (k_m - \lambda) = r_1 \frac{r_1}{\rho + 1} + U_i = r_1 (\lambda + C) + K(x - y)
\]

(3.24)

for some integer \( 0 \leq K < r_1 \). Let us remember that \( k_m - \lambda \) is of the form \( \lambda + a_m(x - y) \) for some integer \( a_m \), and \( \frac{1}{\lambda + v(x - y)} + \frac{1}{\lambda + u(x - y)} > \frac{1}{\lambda + (v+1)(x - y) + (u+1)(x - y)} \) if \( u + 1 \leq v - 1 \). Therefore

\[
\rho + 1 = \sum_{\{m : i \in B_m\}} \left( \frac{1}{k_m - \lambda} \right) \geq \frac{K}{\lambda + C + x - y} + \frac{r_1 - K}{\lambda + C}.
\]

(3.25)
Solving (3.24) for $K$ and substituting into (3.25) we get an inequality for $U_i$. Solving it for $U_i$ we get (3.23). □

**Lemma 3.3.11** Let $D$ be a non type-1 $\lambda$-design. Let $C = (x - y) \left\{ \frac{r_2\rho - \lambda}{x - y} \right\}$. Then for all $i \in C_2$,

\[ U_i \geq \left(1 + \frac{1}{\rho}\right) \left(\frac{r_2\rho}{\rho + 1} - \lambda - C\right) \left(\lambda + C + x - y - \frac{r_2\rho}{\rho + 1}\right). \] (3.26)

**Lemma 3.3.12** Let $D$ be a non type-1 $\lambda$-design. Let $C = (x - y) \left\{ \frac{r_1 - 1}{x - y} \right\}$. Then for all $i \in C_1$,

\[ V_i \geq (r_2 - 1)(x_1 + 2C + \lambda + x - y) - \left(1 + \frac{1}{\rho} - \frac{1}{\lambda}\right)(\lambda + C)(\lambda + C + x - y). \] (3.27)

**Lemma 3.3.13** Let $D$ be a non type-1 $\lambda$-design. Let $C = (x - y) \left\{ \frac{r_1 - 1}{x - y} \right\}$. Then for all $i \in C_2$,

\[ V_i \geq (r_1 - 1)(x_2 + 2C + \lambda + x - y) - \left(1 + \rho - \frac{1}{\lambda}\right)(\lambda + C)(\lambda + C + x - y). \] (3.28)

The above twelve lemmas combined with equations (2.164) and (2.165) provide us with upper and lower estimates for $U_1$ and $U_2$.

These estimates can be somewhat sharpened by observing that the left hand sides of the equations (2.101), (2.103), (2.105), (2.128), (2.130) and (2.132) are obviously integers, and therefore the estimates for $U$ must be such that the right hand sides of the equations are integers, too. This observation is implemented in the function `Rounding` (see 4.3.2).
At this point we can check if the estimates we have obtained are consistent with Lemma 2.3.13. If we find a contradiction, the case can be discarded, otherwise we proceed with building the table.

Table 3 in Appendix B, shows how many parameter triples remain at this point for the first few \( \lambda \) values.

As a corollary of Lemma 3.3.9 and the definition of \( U \) (equations (2.95) and (2.96)), \( U \) can grow in increments of \( x - y \), too. Hence we can list the possible \( U_1 \) and \( U_2 \) values between the lower and upper estimates by incrementing the lower estimate by multiples of \( x - y \).

### 3.3.2 Computing the \( k_m - \lambda \) Values

Next we compute the following quantities. Note that in type-1 designs, \( j_{ab} = r_{ij} \) for all \( i \in C_a, j \in C_b, i \neq j \). We shall call the numbers \( j_{ab} \) the good values of \( r_{ij} \).

\[
\begin{align*}
    j_{o_{11}} &= \left\lfloor \frac{r_1 \rho}{\rho + 1} \right\rfloor, \\
    j_{o_{12}} &= \left\lfloor \frac{r_1}{\rho + 1} \right\rfloor, \\
    j_{o_{21}} &= \left\lfloor \frac{r_2 \rho}{\rho + 1} \right\rfloor, \\
    j_{o_{22}} &= \left\lfloor \frac{r_2}{\rho + 1} \right\rfloor.
\end{align*}
\]

For each value \( U = U_i \) we compute the following quantities, which measure how much do the \( r_{ij} \) differ from their good values.
\[ m_{11}^{(i)} = \sum_{j \in c_1, j \neq i} (r_{ij} - j_o_{11}), \]  
\[ m_{12}^{(i)} = \sum_{j \in c_2} (r_{ij} - j_o_{12}), \]  
\[ sq_1^{(i)} = \sum_{j \in c_1, j \neq i} (r_{ij} - j_o_{11})^2 + \sum_{j \in c_2} (r_{ij} - j_o_{12})^2, \]  
\[ m_{21}^{(i)} = \sum_{j \in c_1} (r_{ij} - j_o_{21}), \]  
\[ m_{22}^{(i)} = \sum_{j \in c_2, j \neq i} (r_{ij} - j_o_{22}), \]  
\[ sq_2^{(i)} = \sum_{j \in c_1, j \neq i} (r_{ij} - j_o_{21})^2 + \sum_{j \in c_2} (r_{ij} - j_o_{22})^2. \]

Note that by Lemmas 2.3.4, 2.3.5, these quantities can be expressed in terms of \( \lambda, d, \rho, U_i, t_1, t_2 \): 

\[ m_{11}^{(i)} = -\frac{U_i}{\rho - 1} + (e_1 - 1)t_1, \quad sq_1^{(i)} = \left(1 - 2t_1^{(i)}\right)U_i + (v - 1)t_1^2, \text{ etc.} \]

Also, let

\[ \text{pos}_1^{(i)} = \frac{sq_1^{(i)} + m_{11}^{(i)} + m_{12}^{(i)}}{2}, \]  
\[ \text{neg}_1^{(i)} = \frac{sq_1^{(i)} - (m_{11}^{(i)} + m_{12}^{(i)})}{2}, \]  
\[ \text{pos}_2^{(i)} = \frac{sq_2^{(i)} + m_{21}^{(i)} + m_{22}^{(i)}}{2}, \]  
\[ \text{neg}_2^{(i)} = \frac{sq_2^{(i)} - (m_{21}^{(i)} + m_{22}^{(i)})}{2}. \]
Lemma 3.3.14 The sum of positive terms on the right hand side of (3.33) and (3.34) is at most \(pos_1^{(i)}\) and the absolute value of the sum of the negative terms is at most \(neg_1^{(i)}\).

Proof. Let \(a\) be the sum of positive terms and \(b\) be the sum of negative terms. Then \(a + b = m_{11}^{(i)} + m_{12}^{(i)}\) and, since all the terms are integers, \(a - b \leq sq_1^{(i)}\). This implies that \(a \leq \frac{sq_1^{(i)} + m_{11}^{(i)} + m_{12}^{(i)}}{2} = pos_1^{(i)}\) and \(-b \leq neg_1^{(i)}\). □

Lemma 3.3.15 The sum of positive terms on the right hand side of (3.36) and (3.37) is at most \(pos_2^{(i)}\) and the absolute value of the sum of the negative terms is at most \(neg_2^{(i)}\).

Proof. Similar to the proof of the previous lemma. □

Next we form the km\(\lambda\) lists, the lists of possible \(k_m - \lambda\) values, that can be associated with a particular \(U\) value. Each such list starts with \(\lambda\) (according to Lemma 3.2.1) or the lower bound found in the next four lemmas, whichever is higher. The numbers grow in increments of \(x - y\), according to Lemma 3.3.9. The upper bound for these lists is given by the following lemmas:

Lemma 3.3.16 If \(B_m\) is a block containing a class-1 point \(i\), then

\[
\lambda + (x - y) \frac{neg_1^{(i)} - r_1 + j_011 + \lambda}{x(j_012 - 1) - y(j_011 - 1)} \leq k_m - \lambda \leq \lambda + (x - y) \frac{\lambda - r_1 + j_011 - pos_1^{(i)}}{x(j_012 - 1) - y(j_011 - 1)}. \tag{3.43}
\]
Proof. For a fixed $i \in C_1$ and a fixed $B_m$ containing $i$, Lemmas 3.3.14 and 3.3.15 imply that,

$$-neg_1^{(i)} \leq \sum_{j \in B_m \cap C_1 \atop j \neq i} (r_{ij} - jo_{11}) + \sum_{j \in B_m \cap C_2} (r_{ij} - jo_{12}) \leq pos_1^{(i)}. \quad (3.44)$$

Since any two blocks intersect in $\lambda$ points, $\sum_{j \in B_m \atop j \neq i} r_{ij} = (r_1 - 1)\lambda + k_m - r_1$. Using equations (2.46) and (2.47) we get

$$neg_1^{(i)} \leq (r_1 - 1)\lambda + k_m - r_1 - \left( \lambda - \frac{k_m - 2\lambda}{\rho - 1} - 1 \right) jo_{11} - \left( \lambda + \frac{(k_m - 2\lambda)\rho}{\rho - 1} \right) jo_{12} \leq pos_1^{(i)}. \quad (3.45)$$

Solving this inequality for $k_m - \lambda$, we get (3.43) $\square$

Lemma 3.3.17 If $B_m$ is a block not containing a class-1 point $i$, then

$$\lambda + (x - y) \frac{-pos_1^{(i)}(x - y)}{xo_{12} - yo_{11}} \leq k_m - \lambda \leq \lambda + (x - y) \frac{neg_1^{(i)}(x - y)}{xo_{12} - yo_{11}}. \quad (3.46)$$

Proof. For a fixed $i \in C_1$ and a fixed $B_m$ not containing $i$,

$$-neg_1^{(i)} \leq \sum_{j \in B_m \cap C_1 \atop j \neq i} (r_{ij} - jo_{11}) + \sum_{j \in B_m \cap C_2} (r_{ij} - jo_{12}) \leq pos_1^{(i)} \leq \frac{r_1}{\lambda} \left( \lambda - \frac{k_m - 2\lambda}{\rho - 1} \right) jo_{11} - \left( \lambda + \frac{(k_m - 2\lambda)\rho}{\rho - 1} \right) jo_{12} \leq pos_1^{(i)}. \quad (3.47)$$

Using equations (2.46) and (2.47) we get

$$neg_1^{(i)} \leq r_1 \lambda - \left( \lambda - \frac{k_m - 2\lambda}{\rho - 1} \right) jo_{11} - \left( \lambda + \frac{(k_m - 2\lambda)\rho}{\rho - 1} \right) jo_{12} \leq pos_1^{(i)}. \quad (3.48)$$

Solving this inequality for $k_m - \lambda$, we get (3.46) $\square$
Lemma 3.3.18 If $B_m$ is a block containing a class-2 point $i$, then

$$
(4.39)
$$

$$
\lambda + (x - y) \frac{\neg g_2^{(i)} - r_2 + j_022 + \lambda}{x(j_022 - 1) - y(j_021 - 1)} \leq k_m - \lambda \leq (x - y) \frac{-\pos g_2^{(i)} - r_2 + j_022 + \lambda}{x(j_022 - 1) - y(j_021 - 1)}.
$$

Proof. For a fixed $i \in C_2$ and a fixed $B_m$ containing $i$,

$$
-\neg g_2^{(i)} \leq \sum_{j \in B_m \cap C_1} (r_{ij} - j_021) + \sum_{j \in B_m \cap C_2} (r_{ij} - j_022) \leq \pos g_2^{(i)}. 
$$

Using equations (2.46) and (2.47) we get

$$
(4.50)
$$

$$
(4.51)
$$

Solving this inequality for $k_m - \lambda$, we get (4.49) □

Lemma 3.3.19 If $B_m$ is a block not containing a class-2 point, then

$$
(4.52)
$$

$$
\lambda + (x - y) \frac{-\pos g_2^{(i)}(x - y)}{xj_022 - yj_021} \leq k_m - \lambda \leq (x - y) \frac{\neg g_2^{(i)}(x - y)}{xj_022 - yj_021}.
$$

Proof. For a fixed $i \in C_2$ and a fixed $B_m$ not containing $i$,

$$
(4.53)
$$

Using equations (2.46) and (2.47) we get

$$
-\neg g_2^{(i)} \leq r_2 \lambda - \left(\lambda - \frac{k_m - 2\lambda}{\rho - 1}\right) j_021 - \left(\lambda + \frac{(k_m - 2\lambda)\rho}{\rho - 1}\right) j_022 \leq \pos g_2^{(i)}. 
$$
Solving this inequality for $k_m - \lambda$, we get (3.52) \(\square\)

When we list the $k_m - \lambda$ values, we have to include only such values that satisfy the trivial $k'_m \leq e_1$ and $k^*_m \leq e_2$ inequalities. Furthermore, for the out-lists the strict form of one of those inequalities holds, since there must be at least one point outside $B_m$.

### 3.3.3 Estimates on the Number of Blocks of a Particular Size

As we mentioned earlier, with each $k_m - \lambda$ value we also store a set of numbers, telling us how many blocks of this particular size can possibly cover a fixed point.

If there are numbers $k_1 - \lambda, k_2 - \lambda, \ldots, k_n - \lambda$ in a km-l-list, let $x_i$ denote the number of blocks of size $k_i$ covering the point. These values are non-negative integers, obviously. There are three equations that these values have to satisfy.

\[
x_1 + x_2 + \cdots + x_n = u_0 \tag{3.55}
\]

\[
\frac{x_1}{k_1 - \lambda} + \frac{x_2}{k_2 - \lambda} + \cdots + \frac{x_n}{k_n - \lambda} = v_0 \tag{3.56}
\]

\[
x_1(k_1 - \lambda) + x_2(k_2 - \lambda) + \cdots + x_n(k_n - \lambda) = w_0 \tag{3.57}
\]

The first equation tells us the degree of a point, and $u_0$ equals $r_1$, $r_2 - 1$, $r_2$ and $r_1 - 1$ for $U_1$ in-list, out-list, $U_2$ in-list and out-list, respectively, simply from the definition of $r_1$ and $r_2$.

The second equation describes the sum of the reciprocals of the km-l-values, based on Lemma 2.3.3. The value of $v_0$ is $\rho + 1, 1 + \frac{1}{\rho} - \frac{1}{\lambda}, 1 + \frac{1}{\rho},$ and $\rho + 1 - \frac{1}{\lambda}$ respectively.
The third equation expresses the sum of the kml-values, based on the definition of $U$ (see (2.95) and (2.96)). The values of $w_0$ are $U_1 + \frac{r_1^2}{\rho\lambda+1}$, $e_1 r_1 + e_2 r_2 - v\lambda - U_1 - \frac{r_1^2}{\rho\lambda+1}$, $U_2 + \frac{r_2^2}{\rho\lambda+1}$, and $e_1 r_1 + e_2 r_2 - v\lambda - U_2 - \frac{r_2^2}{\rho\lambda+1}$ respectively for the four kinds of lists.

These equations will be used at several points in this method. Right now, we are going to use them to find an upper bound for the $x_1, x_2, \ldots, x_n$ values, i.e. the numbers of blocks of a particular size. The idea is the same as behind inequalities (3.23)–(3.28). Namely if we form a reciprocal sum with a list of given numbers such that the number of terms and the sum of terms is given, then we get the lowest possible reciprocal sum if we choose the terms as evenly as possible within the given list. Based on this observation for each $k_m - \lambda$ value we can find the largest number, such that including $k_m - \lambda$ this many times is still possible to find a reciprocal sum not larger than the appropriate $w_0$ value.

After every successful reduction of the kml-lists during the entire algorithm, we check the table to see if any of the kml-lists became short enough to solve the Diophantine equations (3.55)–(3.57). This is quite time consuming, so initially we solve the equations for lists of length three or less. If no solution is found, we can eliminate the corresponding line from the table. These tests are implemented in the functions IsShortOK and IsListOK (see 4.3.7 and 4.3.8).

The following lemma is almost trivial, but it is extremely useful and is used several times in the algorithm.

**Lemma 3.3.20** If $\sum \frac{1}{k_i - \lambda} = \text{frac}$ and there is some prime $p$ such that one particular $k_m - \lambda$ value is divisible by $p^\alpha$ and all other $k_i - \lambda$ values and frac are divisible by
at most $p^\beta$, where $\alpha > \beta$, then the number of terms with the denominator $k_m - \lambda$ is a multiple of $p^{\alpha-\beta}$.

Proof. Suppose there are $a$ terms of $\frac{1}{k_m - \lambda}$ on the left hand side. Then

$$\frac{a}{k_m - \lambda} = \frac{1}{k_i - \lambda} - \sum_{k_i \neq k_m} \frac{1}{k_i - \lambda}. \quad (3.58)$$

The denominator of the right hand side is clearly divisible by at most $p^\beta$, while the denominator on the left hand side is divisible by $p^\alpha$. For the two to be equal, $a$ must be a multiple of $p^{\alpha-\beta}$. □

The lemma above gives us a way to restrict how many times a particular $k_m - \lambda$ value can be used, if the $k_m - \lambda$ value contains a unique prime power.

The following five lemmas give us some further ways to limit these numbers.

**Lemma 3.3.21** If there are at least two blocks of size $k_m$ that intersect in $C_2$, then

$$\lambda - 2 * k'_m + e_1 - 1 \geq 0. \quad (3.59)$$

Proof.

Let us take two such blocks. Each has $k'_m$ points in $C_1$. Since the two blocks intersect in $C_2$, they have at most $\lambda - 1$ common points in $C_1$. On the other hand we know that the number of points in $C_1$ is $e_1$. Hence

$$2k'_m - (\lambda - 1) \leq e_1. \quad (3.60)$$

Rearranging the terms we get (3.59). □

Consequently, if $k_m - \lambda$ occurs on the in-list of a $U_2$ value and $k'_m$ does not satisfy (2.55), then the multiplicity of $k_m - \lambda$ is at most 1 on this list.
**Lemma 3.3.22** Suppose that for a fixed \( i \in C_2 \) and for every \( j \in C_2 \) there are at least \( b \) blocks of size \( k_m \) containing both points. Then the number of blocks of size \( k_m \) containing the fixed \( i \in C_2 \) is at least \( \frac{(b-1)(k_m^* - 1)}{\lambda - (2 \cdot k_m^* - e_1 + 1)} + 1 \).

**Proof.** Let us fix one such block \( B \). Any other block of size \( k_m \) can intersect \( B \cap C_2 \) in at most \( \lambda - (2 \cdot k_m^* - e_1 + 1) \) points, and the sum of the intersections is at least \( (b-1)(k_m^* - 1) \). The plus 1 stands for \( B \). \( \square \)

**Lemma 3.3.23** Suppose that for a fixed \( i \in C_2 \) and for every \( j \in C_2 \) there are exactly \( b \) blocks of size \( k_m \) containing both points. Then the number of blocks of size \( k_m \) containing the fixed \( i \in C_2 \) is \( \frac{(e_2-1)b}{k_m^* - 1} \). If the number of blocks of size \( k_m \) containing both points is at least \( b \), then the number of blocks of size \( k_m \) containing the fixed \( i \in C_2 \) is at least \( \frac{(e_2-1)b}{k_m^* - 1} \).

**Proof.** Let us fix \( i \in C_2 \) and suppose that there are \( c \) blocks of size \( k_m \) containing \( i \). Each such block contains \( k_m^* - 1 \) more class-2 points. On the other hand, there are \( e_2 - 1 \) class-2 points besides \( i \) and each of them is contained in \( b \) (or at least \( b \)) blocks containing \( i \). Hence \( c(k_m^* - 1) = b(e_2 - 1) \), or \( c(k_m^* - 1) \geq b(e_2 - 1) \), respectively. \( \square \)

**Lemma 3.3.24** Suppose that for a fixed \( i \in C_1 \) and for every \( j \in C_1 \) there are exactly \( b \) blocks of size \( k_m \) not containing either of the two points. Then the number of blocks of size \( k_m \) not containing the fixed \( i \in C_1 \) is \( \frac{b(e_1-1)}{e_1 - k_m^* - 1} \). If the number of blocks of size \( k_m \) not containing either of the two points is at least \( b \), then the number of blocks of size \( k_m \) not containing the fixed \( i \in C_1 \) is at least \( \frac{b(e_1-1)}{e_1 - k_m^* - 1} \).
Proof. Let us fix $i \in C_1$ and suppose that there are $c$ blocks of size $k_m$ not containing $i$. Each such block has $c_1 - k'_m - 1$ more class-1 points outside of it. On the other hand, there are $c_1 - 1$ class-1 points besides $i$ and each of them is outside of $b$ (or at least $b$) blocks of size $k_m$ not containing $i$. Hence $c(c_1 - k'_m - 1) = b(c_1 - 1)$ or $c(c_1 - k'_m - 1) \geq b(c_1 - 1)$ respectively. □

Lemma 3.3.25 If $\lambda = -d$ then every class-2 point is contained in at most one block of size $2\lambda$.

Proof. If $\lambda = -d$ then according to (2.38) $c_1 = \lambda$. (2.46) yields $k'_m = \lambda$. This means that any two such blocks intersect only in $C_1$ and are disjoint in $C_2$. □

The above estimates are implemented in functions SubRecipSum, SubChangeMax and ChangeMaxRed. (see 4.3.3, 4.3.4 and 4.3.5)

3.4 Tests Eliminating Particular $k_m - \lambda$ Values

According to Lemma 2.3.3 we have to be able to produce a reciprocal sum of $\frac{1}{\rho} - \frac{1}{\lambda}$ from the numbers of the $U_1$ out-list for which $k'_m \leq c_1 - 2$, and $\frac{1}{\rho}$ from the numbers of the $U_2$ in-list. If one of the numbers $k_m - \lambda$ in the list contains some unique prime power $p^\alpha$ and all other denominators are divisible by at most $p^\beta$, then according to Lemma 3.3.20 the multiplicity of $k_m - \lambda$ is 0 or at least $p^{\alpha - \beta}$. We can check if $\frac{p^{\alpha - \beta}}{k_m - \lambda}$ is greater than $\frac{1}{\rho} - \frac{1}{\lambda}$ or $\frac{1}{\rho}$ respectively. If this is the case, then this $k_m - \lambda$ value can not be part of any reciprocal sum, and hence it can be removed from the list. This observation is implemented in the functions PrimeRed, Trimming and SubTrimming (see 4.4.1, 4.4.2 and 4.4.3).
Lemma 3.4.1 Any $k_m - \lambda$ value must appear in all four kinds of lists: $U_1$ and $U_2$, in-list and out-list, with the possible exception of the following values: $(\rho - 1)(c_1 - \lambda) + \lambda, \frac{(c_2 - \lambda)(\rho - 1)}{\rho} + \lambda, \text{ and } \lambda \rho$.

Proof. For any block of size $k_m$, there are some points in $C_1$ that are inside the block, and some that are outside, and similarly there are some points in $C_2$ that are inside and some are outside. The only possible exceptions are the cases where

1. the block includes the entire $C_1$, i.e. $k'_m = c_1$. According to (2.46) this is equivalent to $k_m - \lambda = (\rho - 1)(c_1 - \lambda) + \lambda$.

2. the block includes the entire $C_2$ i.e. $k^*_m = c_2$. According to (2.47) this is equivalent to $k_m - \lambda = \frac{(c_2 - \lambda)(\rho - 1)}{\rho} + \lambda$.

3. the block is disjoint to $C_1$, i.e. $k'_m = 0$. According to (2.46) this is equivalent to $k_m - \lambda = \lambda \rho$.

4. notice that the fourth logical case, when the block is disjoint to $C_2$, i.e. $k^*_m = 0$ never happens, since $k^*_m \geq \lambda$.\[ \square \]

The test based on this lemma is implemented in the function CommonRed (see 4.4.4).

Checking whether a number can be part of a reciprocal sum can be done by direct computation, but that is extremely time consuming in most cases. Hence we check first some number theoretical conditions that can be checked much faster and may be able to exclude some of the cases.

The next two tests work again with the $U_1$ out-lists and with the fact that we have to be able to produce $\frac{1}{\rho} - \frac{1}{\lambda}$ as a reciprocal sum from the numbers in this list.
As before, we are working with $k_m - \lambda$ values where $k'_m \leq e_1 - 2$, i.e. there are at least two class-1 points outside the block. For each number $k_m - \lambda$ in the kml-list, we want to check whether this number can be included in a reciprocal sum. In the first test, in order to save time, we check only the following. For each prime factor $p$ of $k_m - \lambda$, we know that if we include $k_m - \lambda$ in the reciprocal sum, then the sum of all the terms sharing $p$, is a fraction $\frac{l}{i}$, where $l$ is the least common multiple of the $k_m - \lambda$ values divisible by $p$. We also know that $\frac{l}{i}$ must be such that the denominator of $\frac{1}{p} - \frac{1}{\lambda} - \frac{l}{i}$ is not divisible by $p$ any more. Therefore we form all such $\frac{l}{i}$ fractions and check whether $\frac{1}{p} - \frac{1}{\lambda} - \frac{l}{i}$ can be formed as a sum of the remaining $k_m - \lambda$ values (the ones not divisible by $p$). We check this by taking the least common multiple of the remaining terms and check if this is a multiple of the denominator of the difference $\frac{1}{p} - \frac{1}{\lambda} - \frac{l}{i}$. If not, then we can not produce this fraction. If we find such a contradiction for each multiple $\frac{l}{i}$, then we can conclude that the $k_m - \lambda$ value that we started with can not be part of any sum, and hence it can be removed. This test is implemented in the function RLConstRed (see 4.4.5).

The second test checks whether $k_m - \lambda$ can be involved in a reciprocal sum $\frac{1}{p} - \frac{1}{\lambda}$ with a brute force method. This time we check this by an exhaustive back-track algorithm. If we get a negative answer, the number can be removed. This test is implemented in the function RLSumRed (see 4.4.6).

In the next test we first collect all the $k_m - \lambda$ values found in any list. For each $k_m - \lambda$ value we find the highest and lowest $U_1$ and $U_2$ values, such that this $k_m - \lambda$ value appears in their in-list. These numbers give upper and lower estimates of $U$
for points within a block of size \( k_m \). We check if these estimates are consistent with Lemma 2.3.13. If a contradiction is found, it means that this block size is not possible and hence the \( k_m - \lambda \) value is removed from every \( km\)-list. This test is implemented in the function \textbf{MaxRed} (see 4.4.7).

In our next test we work with the \( U_2 \) in-lists. According to Lemma 2.3.3 we have to be able to produce a reciprocal sum of \( \frac{1}{\rho} \) from this list. We check with an exhaustive search, what is the smallest number of terms that is needed, if a particular \( k_m - \lambda \) value is included. If this number of terms is larger than \( j\sigma_{22} \) then we know that for a fixed class-2 point \( i \) and all other class-2 points \( j, r_{ij} > j\sigma_{22} \). However, by Theorem 2.3.6, there is \( j \) with \( r_{ij} = j\sigma_{22} \). This contradiction shows that the \( k_m - \lambda \) value can be removed. This test is implemented in the function \textbf{LastSumRed} (see 4.4.8).

The next test works with the \( U_1 \) out-lists and \( U_2 \) in-lists knowing that we have to be able to form reciprocal sums that add up to \( \frac{1}{\rho} \) and \( \frac{1}{\rho} - \frac{1}{\lambda} \) respectively. We take a \( k_m - \lambda \) value and check what other values can appear in the same sum. We do this by taking all possible pairs and checking if including both we can produce the required reciprocal sum. This way for each fixed value \( k_m - \lambda \) we can form a list of values that can appear together with it. If this list is reasonably short, than we can attempt to solve the equations (3.55)–(3.57). If no solution is found than \( k_m - \lambda \) can not appear in a sum, and hence it can be removed. This test is implemented in the functions \textbf{CardRed} and \textbf{SubCardRed} (see 4.4.9 and 4.4.10).
Lemma 3.4.2 If there are two blocks of size \( k_m \), intersecting in \( C_2 \), then there is some \( U_2 \) value such that \( k_m - \lambda \) appears both in the in-list and in the out-list.

Proof. Let us take two blocks \( B_1 \) and \( B_2 \) of the same size \( k_m \). Let us recall that the two blocks have \( \lambda \) common points, \( k_m^* \geq \lambda \), and that if \( k_m^* = \lambda \) then \( k_m' = \lambda \).

If \( k_m^* > \lambda \) then there is a class-2 point \( i \in B_1 \setminus B_2 \), and hence \( k_m - \lambda \) appears both in the in-list and in the out-list of \( i \).

If \( k_m^* = \lambda \) then the only way the two blocks can have empty symmetric difference in \( C_2 \) if they intersect in \( \lambda \) class-2 points. This would mean that their class-1 parts are disjoint. Hence the two blocks would contain \( 2\lambda \) class-1 points. This contradicts the fact that \( c_1 < 2\lambda \). \( \square \)

The test based on this lemma is implemented in the functions \texttt{U2EitherRed} and \texttt{SubEither} (see 4.4.11 and 4.4.12).

Lemma 3.4.3 If there are two blocks of size \( k_1 \) and \( k_2 \), \( k_1 > k_2 \), intersecting in \( C_2 \), then there must be a \( U_2 \) value such that \( k_1 - \lambda \) appears in the in-list and \( k_2 - \lambda \) in the out-list. Furthermore, if \( k_1^* > \lambda \) or if \( k_1' + k_2' > c_1 \), then there is also some \( U_2 \) value such that \( k_2 - \lambda \) appears in the in-list and \( k_1 - \lambda \) appears in the out-list.

Proof. If both \( k_1^* > \lambda \) and \( k_2^* > \lambda \) or if \( k_1' + k_2' > c_1 \), then there are some class-2 points in the first block that are not in the second and vice-versa. Otherwise the only thing that we can guarantee is that for the larger value \( k_1^* > \lambda \), and hence the larger block contains some class-2 points not included in the smaller block. \( \square \)

The test based on this lemma is implemented in the function \texttt{U2PairRed} (see 4.4.13).
3.5 Tests Eliminating a Line

Let us consider the in-lists for the $U_2$ values. According to Lemma 2.3.3 we have to be able to form a reciprocal sum whose value is $\frac{1}{\rho}$. Let us find the smallest number of terms that makes this possible. This is an obvious lower estimate for $r_{ij}$. Next we check if this lower estimate is consistent with (2.130). If we find a contradiction, this $U_2$ value can not appear in the design, so we can remove the line. This test is implemented in the function $\text{RijU2inRed}$ (see 4.5.1).

We can perform a similar test with the out-lists of $U_1$. Again, according to Lemma 2.3.3 we have to be able to form a reciprocal sum whose value is $\frac{1}{\rho} - \frac{1}{\lambda}$. In this case we have to make sure that we are using only $k_m - \lambda$ values where $k_m' \leq c_1 - 2$, i.e. there are at least two class-1 points outside the appropriate block. We can find out the smallest number of terms needed to form the required reciprocal sum. If this number is $n$, then $n - v + 2r_1$ is a lower estimate for $r_{ij}$. Next we check if this estimate is consistent with (2.101). If a contradiction is found, the line can be removed. This test is implemented in the function $\text{RijU1outRed}$ (see 4.5.2).

The next test uses again the fact that we have to be able to form a reciprocal sum of $\frac{1}{\rho} - \frac{1}{\lambda}$ from the numbers in the out-lists of $U_1$. If the least common multiple of the $k_m - \lambda$ values is not a multiple of the denominator of $\frac{1}{\rho} - \frac{1}{\lambda}$, then there is obviously no hope of producing such a sum, and hence the line can be removed. This test is implemented in the functions $\text{ConstRed}$ and $\text{SubConstr}$ (see 4.5.3 and 4.5.4).

The next tests can be performed for all four kinds of km-lists. We take equations (3.55), (3.56) and (3.57) and check the reciprocal sum corresponding to the (unique)
solution of (3.55) and (3.57) where at most two consecutive $k_m - \lambda$ values occur with nonzero coefficients. As we discussed earlier, this gives the smallest possible reciprocal sum. If this is larger than the right hand side of (3.56) then clearly we can not solve the system of equations, so the whole line can be removed.

The other test checks whether any of the $k_m - \lambda$ values contains a unique prime power. Suppose $p^\alpha | k_s - \lambda$ and $p^\beta$ is the largest power of $p$ occurring in the other $k_m - \lambda$ values and in the denominator of the right hand side of (3.56). Then according to Lemma 3.3.20, $x_s$ must be a multiple of $p^{\alpha - \beta}$. We subtract $p^{\alpha - \beta}$, $\frac{p^{\alpha - \beta}}{k_s - \lambda}$ and $p^{\alpha - \beta}(k_s - \lambda)$ from the right hand sides of (3.55), (3.56) and (3.57), respectively, and apply the previous test. If we find a contradiction, we can not remove the whole line, but at least we can remove $k_s - \lambda$ from the list. This observation is implemented in the functions SinRecipSum and RecipRed (see 4.5.5 and 4.5.6).

3.6 Some Global Tests

This section describes some tests that check some global properties of the table and in case of a contradiction, make possible to reject the $\lambda, d, p$ triple.

Lemma 3.6.1 The number $\frac{\lambda(p+1)^2}{\rho} \prod_{m=1}^{n} (k_m - \lambda)$ is a perfect square.

Proof. Let us take an incidence matrix $D$, showing which points are included in which blocks. Since $D$ is a 0–1 matrix, its determinant is an integer. Let us consider the matrix $DD^T$. The $(i, j)$ entry of the matrix shows the size of the intersection of blocks $B_i$ and $B_j$. This matrix has $\lambda$ in every entry except the diagonal one, since any two different blocks intersect in $\lambda$ points. The diagonal entries are $k_i$, since the
intersection of $B_i$ by itself is the whole block, whose size is $k_i$:

$$
\begin{vmatrix}
    k_1 & \lambda & \lambda & \cdots & \lambda \\
    \lambda & k_2 & \lambda & \cdots & \lambda \\
    \lambda & \lambda & k_3 & \cdots & \lambda \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    \lambda & \lambda & \lambda & \cdots & k_n
\end{vmatrix}
$$

(3.61)

In the proof of Theorem 2.1.1, we have computed the determinant:

$$
\det(DD^T) = \prod_{m=1}^{n} (k_m - \lambda) \left(1 + \lambda \sum_{m=1}^{n} \frac{1}{k_m - \lambda}\right).
$$

(3.62)

Using (2.51) we can rewrite it as:

$$
= \prod_{m=1}^{n} (k_m - \lambda) \left(1 + \lambda \left(\rho + 2 + \frac{1}{\rho} - \frac{1}{\lambda}\right)\right).
$$

(3.63)

Hence

$$
\det(D)^2 = \det(DD^T) = \frac{\lambda(\rho+1)^2}{\rho} \prod_{m=1}^{n} (k_m - \lambda),
$$

(3.64)

proving our claim. □

We apply Lemma 3.6.1 in two different settings. Let $k_1, k_2, \ldots, k_m$ be the possible block sizes in the design, with multiplicities $x_1, x_2, \ldots, x_m$. Since each prime in $\frac{M(r+1)^2}{\rho} \prod_{m=1}^{n} (k_m - \lambda)$ must occur on an even power, it gives a system of equations for the $x_i \mod 2$ values. If this system has no solutions, the triple $(\lambda, d, \rho)$ can be eliminated.

The test based on this lemma is implemented in the function SquareRed (see 4.6.2)

Let us notice that we have three Diophantine equations similar to (3.55)-(3.57) for the total number of blocks of given sizes. We know that there are $v$ blocks altogether (by the definition of $v$), that the sum of all $k_m - \lambda$ values is $e_1 r_1 + e_2 r_2 - v \lambda$ according
to the definitions of $e_1, e_2, r_1, r_2$, and that the sum of the reciprocals is $p + 2 + \frac{1}{\rho} - \frac{1}{\lambda}$ according to (2.51). We can solve this system of Diophantic equations. Next we can check for each solution if it satisfies the conditions of Lemma 3.6.1 and eliminate those that do not. If no solutions remain, than we can eliminate this $(\lambda, d, \rho)$ triple. If there are some solutions remaining, then for each $k_m - \lambda$ value, we can check what is the highest number in any solution, and this gives us an upper estimate of the global number of blocks of this size, and hence it is also an upper estimate for the number of blocks of this size covering any given point.

The test based on this lemma is implemented in the function GlobalSoluRed (see 4.6.3).

**Lemma 3.6.2** Suppose that there is a prime power $p^\alpha$ in the denominator of $\frac{1}{\rho}$ that does not appear in the denominator of $\frac{1}{\lambda}$ and appears in exactly one $k_s - \lambda$ value. Moreover, suppose that there is a unique positive integer $m$ such that $\frac{1}{\rho} - \frac{1}{\lambda} - \frac{m}{k_s - \lambda}$ is non-negative and its denominator does not contain the prime power $p^\alpha$. Then $x_s = \frac{m e_1 (e_1 - 1)}{(e_1 - k_s^2)(e_1 - k_s^2 - 1)}$ is an integer, and the denominator of $\frac{x_s}{k_s - \lambda} - \frac{1}{\rho}$ is not divisible by $p^\alpha$.

**Proof.** Let $x_s$ be the number of blocks of size $k_s$ in the design. According to the conditions, for each pair $i, j \in C_1$ there are exactly $m$ blocks of size $k_s$ containing neither $i$ nor $j$. Outside of a fixed block of size $k_s$, there are $\binom{e_1 - k_s^2}{2}$ pairs $i, j$. On the other hand there are $\binom{e_1}{2}$ pairs in $C_1$ altogether. Hence

$$x_s \binom{e_1 - k_s^2}{2} = m \binom{e_1}{2}. \quad (3.65)$$
Solving the equation for $x_s$, we get the first statement of the lemma. According to (2.51),

$$\sum_{B_m \in B} \frac{1}{k_m - \lambda} = \rho + 2 + \frac{1}{\rho} - \frac{1}{\lambda}. \quad (3.66)$$

If we rearrange the terms we get:

$$\frac{x}{k_s - \lambda} - \frac{1}{\rho} = \rho + 2 - \frac{1}{\lambda} + \sum_{k_i \neq k_s} \frac{1}{k_i - \lambda}. \quad (3.67)$$

The denominator of the right hand side clearly does not contain $p^n$, and hence the denominator of the left hand side can not be divisible by $p^n$ either. □

The test based on the above lemma is implemented in the function UniquePrime1Red (see 4.6.4).

**Lemma 3.6.3** Suppose that there is a prime power $p^n$ in the denominator of $\frac{1}{\rho}$ that divides exactly one $k_s - \lambda$ value and there is a unique positive integer $m$ such that $\frac{1}{\rho} - \frac{m}{k_s - \lambda}$ is non-negative and the denominator does not contain the prime power $p^n$. Then $x_s = \frac{m(r_2(r_2 - 1))}{k_s^2(k_s^2 - 1)}$ is an integer.

**Proof.** According to the conditions, for each pair $i, j \in C_2$ there are exactly $m$ blocks of size $k_s$ containing $i, j$. A fixed block of size $k_s$ contains $\binom{k_s^*}{2}$ pairs. On the other hand there are $\binom{r_2}{2}$ pairs in $C_2$ altogether. Hence

$$x_s \binom{k_s^*}{2} = m \binom{r_2}{2}. \quad (3.68)$$

Solving the equation for $x_s$, we get the statement of the lemma. □

The test based on the above lemma is implemented in the function UniquePrime2Red (see 4.6.5)
Lemma 3.6.4 For each \( m \), let \( x_m \) be an upper estimate on the number of blocks of size \( k_m \). Let \( p^\alpha \) be a prime power such that it is a divisor of the denominator of \( \frac{1}{\lambda} \) but not a divisor of the denominator of \( \frac{1}{\rho} \). Let \( l \) be the least common multiple of the \( k_m - \lambda \) values that are multiples of \( p^\alpha \). Let \( t \) be the smallest positive integer such that the denominator of \( \frac{1}{t} \) is not divisible by \( p^\alpha \) and \( \frac{1}{t} > \frac{1}{\lambda} \). Then

\[
\left( \binom{c_1}{2} \right) \left( \frac{t}{l} - \frac{1}{\lambda} \right) \leq \sum_{p^\alpha | k_m - \lambda} \frac{x_m}{k_m - \lambda} \left( \binom{c_1}{2} \frac{k_m'}{2} \right). \tag{3.69}
\]

Proof. According to Lemma 2.3.3 for each pair \( i, j \) of class-1 points, the reciprocal sum of the \( k_m - \lambda \) values for the blocks not containing \( i \) and \( j \) is \( \frac{1}{\rho} - \frac{1}{\lambda} \). Let us estimate two different ways the contribution to these sums of those \( k_m - \lambda \) values that are multiples of \( p^\alpha \). On one hand they must contribute at least \( \left( \binom{c_1}{2} \right) \left( \frac{t}{l} - \frac{1}{\lambda} \right) \) to each sum in order to eliminate \( p^\alpha \) from the denominator, and there are \( \binom{c_1}{2} \) pairs, therefore they contribute at least \( \left( \binom{c_1}{2} \right) \left( \frac{t}{l} - \frac{1}{\lambda} \right) \) to these sums. On the other hand for each block of size \( k_m \) there are \( \binom{c_1-k_m}{2} \) pairs of class-1 points outside the block, and there are at most \( x_m \) blocks of this size. Hence they can contribute at most \( \sum_{p^\alpha | k_m - \lambda} \frac{x_m}{k_m - \lambda} \left( \binom{c_1-k_m}{2} \right) \) to these sums. \( \square \)

The test based on this lemma is implemented in the function RLGlobalRed (see 4.6.6).

3.7 Tests Eliminating Solutions of the Diophantic Equations

In this section we are dealing with the solutions of the Diophantic equations described in (3.55), (3.56) and (3.57) for the \( U_2 \) in-list. Having a solution means that we have the exact number of blocks with different sizes that cover a class-2 point.
For each solution, we re-apply the tests described in the previous sections. Since some of the \( k_m - \lambda \) values on the in-list may appear with 0 coefficient in the solution, we may (and almost always do) work with shorter in-lists. The following lemmas describe the modified tests.

**Lemma 3.7.1** Suppose that the denominator of \( \frac{1}{\rho} \) contains a prime power \( p^n \) found in only one \( k_m \lambda \) value of the solution, in \( k_s - \lambda \), and let \( c \) be the smallest multiplicity that has to be included in a reciprocal sum to eliminate \( p^n \) from the denominator of \( \frac{1}{\rho} - \frac{c}{k_s - \lambda} \). If \( k_s - \lambda \) occurs with multiplicity \( n \) in the solution, then

\[
n(k_s^* - 1) \geq c(e_2 - 1).
\]  

(3.70)

*Proof.* Let us fix a class-2 point \( i \) and let us count the number of class-2 points \( j \) and blocks of size \( k_s \) covering the two. There are \( n \) blocks of size \( k_s \) containing \( i \), covering the other class-2 points \( n(k_s^* - 1) \) times. On the other hand there are \( e_2 - 1 \) other class-2 points and in each reciprocal sum we have to include at least \( c \) numbers \( k_s - \lambda \). □

The test based on this lemma and also on Lemmas 3.3.22 and 3.3.23 is implemented in the function `Solu1` (see 4.7.1).

If in a solution the number 0 appears, than we can take those \( k_m - \lambda \) values that appear with positive multiplicity and check if any of them contain a unique prime power, relative to this reduced list. If a unique prime power \( p^\alpha \) is found in \( k_s - \lambda \), and \( p^\beta \) is the highest power of \( p \) found in any other \( k_m - \lambda \) value, then using Lemma 3.3.20 we know that \( k_s - \lambda \) has to appear in the reciprocal sum at least \( p^{\alpha-\beta} \) times.
If \( \frac{p^\alpha - \alpha}{k_m - \lambda} > \frac{1}{\rho} \), then \( k_s - \lambda \) can not appear in the sum, so the solution can be removed.

This observation is implemented in the function Solution2 (see 4.7.2).

If there is some prime power, \( p^\alpha \), found in some \( k_m - \lambda \) values with positive multiplicities, but not dividing the denominator of \( \frac{1}{\rho} \), we can do the following computation.

Let us fix a point \( i \) and a block \( B_s \) containing \( i \) such that \( p^\alpha | k_s - \lambda \). Next we consider the size of the intersection of \( B_s \) with other blocks \( B_m \) with \( p^\alpha | k_m - \lambda \) and we can give two estimates on how much these \( \frac{1}{k_m - \lambda} \) values contribute in the reciprocal sums for \( j \in B_s \cap C_2 \). Let \( l = \text{lcm} \{ k_m - \lambda : p^\alpha | k_m - \lambda \} \). All such blocks contribute at least \( \frac{1}{l} \) at each point, where \( t \) is the smallest integer such that \( \frac{1}{l} \geq \frac{1}{k_s - \lambda} \) and after simplification the denominator of \( \frac{1}{l} \) does not contain \( p^\alpha \). Moreover \( \frac{1}{\rho} - \frac{1}{l} \) can be produced as a reciprocal sum of the \( k_m - \lambda \) values not divisible by \( p^\alpha \). There are \( k_s^* - 1 \) class-2 points in \( B_s \setminus \{ i \} \). The other blocks whose \( k_m - \lambda \) value includes \( p^\alpha \) contribute at least \( (k_s^* - 1)(t - \frac{1}{k_s - \lambda}) \) to the sums. On the other hand, if \( c_{ij} \) is an upper estimate on the number of common class-2 points of two blocks of sizes \( k_i \) and \( k_j \), and there are at most \( x_i \) further blocks of size \( k_i \), then \( \sum_{p^\alpha | k_m - \lambda} \frac{x_m c_{ms}}{k_m - \lambda} \) is an upper estimate of the contribution. Therefore

\[
\sum_{p^\alpha | k_m - \lambda} \frac{x_m c_{ms}}{k_m - \lambda} \geq (k_s^* - 1) \left( \frac{t}{l} - \frac{1}{k_s - \lambda} \right).
\] (3.71)

The test checking this condition is implemented in Solution3 (see 4.7.3).

**Lemma 3.7.2** Suppose that there are prime powers \( p^\alpha \) and \( \rho^\beta \), \( \alpha > \beta \) with the following properties. Exactly two \( k_m - \lambda \) values, \( k_1 - \lambda \) and \( k_2 - \lambda \), are divisible by \( p^\alpha \), and all other \( k_m - \lambda \) values appearing in the solution and in the denominator of \( \frac{1}{\rho} \) are
divisible by at most $p^\beta$. Moreover, the multiplicity of $k_1 - \lambda$ is 1, and $k_2 \neq 2\lambda$. Then, if $\frac{p^{n-\beta}}{k_2-\lambda} \geq \frac{1}{\rho}$, then the solution can be eliminated.

Proof. Suppose that the sizes of the blocks containing some $i \in C_2$ satisfy this solution. Let $B_1$ be the the unique block of size $k_1$ containing $i$, and let $B_2$ be a block of size $k_2$, $i \in B_2$. Then, since $|B_2 \cap C_2| > \lambda$, there exists $j \in (B_2 \cap C_2) \setminus B_1$. In the reciprocal sum $\sum_{m:j \in B_m} \frac{1}{k_m-\lambda}$ the value $\frac{1}{k_2-\lambda}$ must occur with multiplicity at least $p^{n-\beta}$, which leads to a contradiction. □

The test based on this lemma is implemented in the function $Sol4$ (see 4.7.4).

**Lemma 3.7.3** Suppose that $\rho$ is an integer, the $k_m - \lambda$ values $\lambda$ and $\lambda + \rho - 1$ appear with positive coefficients $n_1$ and $n_2$, respectively in the solution, and $\frac{1}{\rho} - \frac{1}{\lambda} - \frac{1}{\lambda + \rho - 1}$ cannot be written as a reciprocal sum of $k_m - \lambda$ values from the solution. Then

$$(\lambda - 1)n_1 \leq e_2 - 1 - (\lambda + \rho - 1) \quad (3.72)$$

and

$$n_1 \leq e_1 - (\lambda - 1). \quad (3.73)$$

**Proof.** Suppose that the sizes of blocks containing some $i \in C_2$ satisfy this solution. Let us fix a block $B_2$ of size $2\lambda + \rho - 1$. For any block $B_1$ of size $2\lambda$, $B_1 \cap B_2 \cap C_2 = \{i\}$ because of the reciprocal sum condition. Hence $|B_1 \cap B_2 \cap C_1| = \lambda - 1$ and, since $|B_2 \cap C_1| = \lambda - 1$, $B_1 \cap B_2 = \{i\} \cup (B_2 \cap C_1)$. This means that for any two blocks $B'_1$, $B'_2$ of size $2\lambda$, $B'_1 \cap B'_2 \cap C_1$ contains the $\lambda$-element set $\{i\} \cup (B_2 \cap C_1)$. Therefore, $B'_1 \cap B'_2 = \{i\} \cup (B_2 \cap C_1)$, and $B'_1$, $B'_2$ are disjoint outside $\{i\} \cup (B_2 \cap C_1)$. Since
\(|B'_1 \cap C_1 \setminus (B_2 \cap C_1)| = 1\) and \(|(B'_1 \cap C_2) \setminus (B_2 \cap C_2)| = \epsilon_2 - (\lambda + \rho)\), the stated inequalities follow. □

The test based on this lemma is implemented in function \texttt{Solu5} (see 4.7.5).

If for two blocks, \(B_1\) and \(B_2\), we know that they have at least two common class-2 points, then they have to appear in the same reciprocal sum of \(\frac{1}{\rho}\). Therefore we check whether \(\frac{1}{\rho} - \frac{1}{k_1 - \lambda} - \frac{1}{k_2 - \lambda}\) can be produced as a reciprocal sum. If not, then we know that this solution can be removed. This test is implemented in the function \texttt{Solu6} (see 4.7.6).
CHAPTER IV

Implementation and Description of the Program

4.1 General Description of the Program

4.1.1 The Language

The program is written in GAP. GAP stands for Groups, Algorithms and Programming, and was developed at the University of Aachen, Germany [Sch]. It is a computing environment that was designed with primarily group theoretical algorithms in mind. The system contains a small kernel written in the C programming language, and a large library of functions that are written in the GAP language and interpreted. The user can use this system interactively or write a program in a file and run the interpreter on this program. This chapter describes the implementation details of the program, written in this manner. The language itself is similar to most modern high level algorithmic programming languages, and hopefully the code will be easily understandable to anybody with some familiarity with such languages.

4.1.2 Overview of the Algorithm

The program takes a set of λ values as an input. First it calls a function (PreProcess) that generates the list of all possible parameter triples with λ being in the given set and
the triple satisfying all simple conditions that can be easily verified by computations using these parameters, without building any further data structures. The function returns to the main program the list of these parameter triples that survived the first, most basic elimination process. The main program takes one such triple at a time, and builds a data structure, called the table, from these. It computes several additional parameters that will be used in later computations and also creates a list of the possible \( U \) values with the corresponding possible \( k_m - \lambda \) values. Then it calls a function, called Reduction, that applies several reduction steps to this data structure repeatedly, until the case is completely eliminated, or till all the reduction steps return the data structure unchanged. There are three kinds of reductions. The first kind is global, i.e. it checks some condition, and if that tests false, the entire table is eliminated immediately. The second kind can remove one line of the database, i.e. it can eliminate one of the possible \( U \) values. The third kind can eliminate only one possible \( k_m - \lambda \) value at a time. If some case survives all the tests, then the data returned by Reduction are output by the main program, meaning that this case has not been eliminated by the tests implemented in the program. In an optimal case the program completes its run without any output, meaning that all the cases were eliminated. If some cases remain, the ReTest function can be called with some different parameters. This function reads as input the cases remaining after a previous run and calls the Reduction function with the new parameter values, in the hope that it can eliminate the case this time.

There are two global variables affecting the program: small.size determines what
is the largest list size, where we still try to solve the Diophantic equations for the parameters. (see 4.3.7). fast is a Boolean variable that determines if the most time consuming tests are run. (see 4.3.11).

4.1.3 The Data Structure

Most variables in the program are stored as ordinary variables or lists of variables. There are four kinds of data that are stored as records (or structures). Below we describe these records and the fields (or members) in them.

Representation of a single $k_m - \lambda$ value

With each $k_m - \lambda$ value (kml value for short), we store not just the numerical value, but also the prime factorization of the number, since this is used frequently, and also some previously obtained information on how many times this particular value can be used. The record contains the following fields:

- **avail** the numeric value of $k_m - \lambda$
- **fact** an ordered list of the prime factors of the number
- **pow** a list giving the exponents of the prime factors in the same order as the previous list
- **max** an integer giving the largest possible multiplicity of blocks with this $k_m - \lambda$ value covering a particular point
- **per** an integer showing that the multiplicities of blocks with this $k_m - \lambda$ value must be a multiple of this number
set a list of integers giving the possible multiplicities of blocks with this $k_m - \lambda$ value in the denominator

**Representation of lists of $k_m - \lambda$ values**

This data structure represents a list of $k_m - \lambda$ values associated with a possible $U$ value. We will refer to this kind of structure as a kml-list. The record contains the following fields:

- **leng** the number of numbers in the list
- **lst** a list of records, in increasing order of the numbers, where each record represents one $k_m - \lambda$ value by a record described above
- **trimmed** a Boolean variable showing if trimming has been performed on this list (see 4.4.2)
- **short** a Boolean variable showing if short list checking has been performed on this list (see 4.3.7)
- **nums** a list of the $k_m - \lambda$ values in increasing order
- **lastsum** a Boolean variable showing if the LastSumRed has been performed on this list (see 4.4.8)
- **u2in** a Boolean variable showing if the U2InRed function has been run on this list (see 4.5.1)
u1out a Boolean variable showing if the U1OutRed function has been run on this list (see 4.5.2)

rlsum a Boolean variable showing if the RLSumRed function has been run on this list (see 4.4.6)

maxcheck a Boolean variable showing if the MaxCheck function has been run on this list (see 4.3.4)

solu a list of solutions found for this list

Representation of a Line

A “line” is a representation of a possible $U$ value and the associated lists of $k_m - \lambda$ values for points inside the block and for points outside. The record contains the following fields:

au gives the $U$ value

inl gives the list of possible $k_m - \lambda$ values for points inside the block as a kmll-list

out gives the list of possible $k_m - \lambda$ values for points outside the block, as a kmll-list

Representation of a Design

In the program all the variables related to a design are stored in a record. This record is passed as a parameter to the reduction functions and the same record is returned, with some fields possibly modified. The record contains the following fields:

1 the value of $\lambda$
d  the value of $d$, see 2.36

r  the value of $\rho$, see 2.35

v  the value of $v$, the number of points and blocks

r1  the number $r_1$, the higher one of the two possible degrees

r2  the number $r_2$, the lower one of the two possible degrees

e1  the number $e_1$, the number of class-1 points

e2  the number $e_2$, the number of class-2 points

jo11  the value of $j_{o11}$, (see 3.29)

jo12  the value of $j_{o12}$, (see 3.30)

jo21  the value of $j_{o21}$, (see 3.31)

jo22  the value of $j_{o22}$, (see 3.32)

u1low  the lower estimate for $U$ in $C_1$

u1high  the higher estimate for $U$ in $C_1$

u2low  the lower estimate for $U$ in $C_2$

u2high  the higher estimate for $U$ in $C_2$

u2crit  the critical value for $U$ in $C_2$, i.e. if $u2low$ is pushed above this number, then

       this case can be eliminated
ch_count integer showing how many consecutive cheap tests have been performed without any change (see 4.3.11)

ex_count integer showing how many consecutive expensive tests have been performed without any change (see 4.3.11)

ulist a list of two lists, for $C_1$ and $C_2$ respectively; each list contains “lines”

small an integer showing what is the largest number of variables where we still try to solve the Diophantic equations

4.2 Utility Functions

The following functions perform some elementary operations needed by some other functions. They are written as separate functions for clarity and convenience.

4.2.1 Ceil

This function takes a number as input and returns the ceiling of it, i.e. the smallest integer larger than or equal to the input.

For the listing see A.2.1.

4.2.2 Floor

This function takes a number as input and returns the floor of it, i.e. the largest integer smaller than or equal to the input.

For the listing see A.2.2.
4.2.3  RecursTestSum

This is a recursive function that tests whether a number can be produced as a reciprocal sum with a prescribed number of terms using numbers from a given list. The function takes five input variables. The first, list, is a record of $k_m - \lambda$ values in km-list format, whose reciprocals can be used as terms in the sum. The algorithm uses the fact that these numbers are listed in an increasing order. The second variable, sum, is the number that we want to produce as a sum of reciprocals. The third, $k$, is the number of terms to be used. The fourth, result, is a list of two elements. The first element is a Boolean value, that eventually will show if the sum can be produced. The second element is a list of integers, where the $i^{th}$ number shows how many fractions are used already with the $i^{th}$ number from list as the denominator. The fifth variable, terms, is again a list of integers, where the $i^{th}$ position shows how many of the $i^{th}$ number from list are needed for some previous number theoretical considerations. The function itself uses a back-track algorithm to find a possible sum. It increments the numbers in the second component of result by one. If no completion is possible with the given starting segment of this list, the last position is increased by one, until the maximum is reached; after that it steps back to the previous position and increases that number by one, etc. The function first checks if there are any unchecked numbers left in the list. If there are, it checks if the required sum can be achieved with the remaining number of terms and remaining numbers. (i.e. if the remaining numbers are too large, hence their reciprocals are too small, or
there are too few terms are allowed, then this is a dead end, and we should back­
track.) If these two conditions check out true, we add a 0 in the next position. If the
remaining sum is exactly one over the remaining number and this number does not
exceed the maximum number allowed for that particular $k_m - \lambda$ value, than we have
found a solution: set the first entry in result to true and the second part of result
is filled up accordingly. If this is not the case, figure out what is an upper bound
for the next number and try every value from 0 to this bound. In order to check if
we can complete the sum with setting the $i^{th}$ number to a particular value, we call
RecursTestSum recursively. If we tried all the values, up to the bound and none of
them works out, than we have to backtrack, and we remove the last number from the
list in result. On completion the function returns result, whose first entry shows
if we succeeded, and the second part gives the coefficients for the sum we have found.

For the listing see A.2.3.

4.2.4 SubSumTest

This function does the preprocessing to RecursTestSum (see 4.2.3). It takes three
input variables. The first, 1st, is a list of the $k_m - \lambda$ values; the second, n_terms, is
the number of terms to be used in the sum; and the third, frac, is the number that
we want to produce as a reciprocal sum, with numbers in the denominators taken
from the list, and using the given number of terms. The main part of this function
is based on the following simple observation. If in the denominator of frac, there is
some prime power that is found in only one of the numbers in the list, than obviously
this number must appear in the sum and also we can give a minimum for the number
of such terms. To find this out, we list the prime factors of the denominator of \texttt{frac}
in \texttt{rfactset}. Then in a loop we take one prime divisor of the denominator of \texttt{frac}
at a time and put its exponent in \texttt{rpower}. Then we loop through all the numbers in the list, checking if there is only one, that has this prime with \texttt{rpower}, or higher exponent. If there is only one such number, then we can compute the minimum number of such terms needed in the sum and put the appropriate number in \texttt{terms}. Also we can subtract this multiple of the reciprocal from the sum that we have to produce. This reduced sum is stored in \texttt{remains}. If these conditions use up the entire sum or the number of terms allowed, then we know that we can not produce the required sum. Otherwise the function \texttt{RecursTestSum} is called with the appropriate arguments, to find out if this sum can be produced. The first component of \texttt{result} shows if \texttt{RecursTestSum} succeeded and hence this is the result for the entire function.

For the listing see A.2.4.

4.2.5 MakeList

This function produces a \texttt{kml}-list. It takes two inputs; the first, \texttt{nums}, is a list of the numbers (in increasing order) and the second, \texttt{card}, is the value of the reciprocal sum, used to make an upper estimate on how many times a particular $k_m - \lambda$ value can be used. It sets the \texttt{max}, \texttt{per} and \texttt{set} fields for each number according to this estimate, and sets all the Boolean flags for the list to \texttt{false}, showing that none of the tests were performed on this list, yet. The function returns a \texttt{kml}-list, representing the list.

For the listing see A.2.5.
4.2.6 MakeFakeList

This function produces a kml-list, just like the previous one. The difference is that this function takes a list of records of the kind described in 4.1.3, i.e. all the fields \( \text{max} \), \( \text{per} \) and \( \text{set} \) are supplied for each \( k_m - \lambda \) value, so there is no need to estimate these. The function returns a kml-list, representing the list.

For the listing see A.2.6.

4.2.7 TouchList

This function takes a kml-list and resets all the Boolean flags to \text{false}. This is used when some data are modified in the list, and hence all the previous tests must be redone. For technical reasons we also have to pass the data structure that represents the entire table, because after any modification to constituent lists, we have to reestablish the fact that the large lists in the table are indeed sets (contain no repetition in the list). This function does not return anything.

For the listing see A.2.7.

4.2.8 ListRemove

This function takes a kml-list and a number, one of the \( k_m - \lambda \) values in the list, and removes this value from the list, changing all the appropriate sub-lists and resetting the Boolean flags. For technical reasons we also have to pass the structure, representing the entire table. This function does not return anything.

For the listing see A.2.8.
4.2.9 SumPossibleNums

This function takes a kml-list and a number as input. It computes the cardinalities of the reciprocal sums formed using the numbers in the list whose sum is the given number. The function returns a list of integers in increasing order, representing the numbers that can be cardinalities of such sums.

For the listing see A.2.9.

4.2.10 IsSumPossible

This function takes a kml-list and a number as input. It answers a similar question to the previous one, except that instead of returning a list of numbers, it returns a Boolean value representing whether the given number can be produced at all as a reciprocal sum from the given list of numbers.

For the listing see A.2.10.

4.2.11 IsSumPossibleLim

This function is a modification of the previous one. It takes one more input, a limit. It also returns a Boolean value, answering the question whether the given sum can be produced as a reciprocal sum from the list using at most as many terms as the number given as the limit.

For the listing see A.2.11.
4.2.12 FindUnique

This function takes a kml-list and a fraction as input. It finds all the unique prime powers in each \( k_m - \lambda \) value. In other words, it finds if a \( k_m - \lambda \) value contains some prime power that is larger than the exponent of that prime in any other \( k_m - \lambda \) value, or in the denominator of the fraction. It finds the excess powers, i.e. if a \( p \) prime appears in this number with an exponent \( \alpha \) and the highest exponent anywhere in any other \( k_m - \lambda \) value, or in the denominator of the fraction is \( \beta \) then the excess power is \( p^{\alpha - \beta} \). The function returns a list of integers, where the length of the list is the same as the length of the input list. For each number in the input kml-list it contains the product of the excess powers found in that number.

For the listing see A.2.12.

4.2.13 FindNecess

This function takes a kml-list and a fraction as input. It finds if there is some prime power in the denominator of the fraction that appears in only one of the \( k_m - \lambda \) values. If such a power is found, it finds all the possible multiplicities that this \( k_m - \lambda \) value can appear with. The function returns a list of lists. If a \( k_m - \lambda \) value contains no unique prime powers, than the associated list is empty. If it contains some unique prime powers, then the associated list contains the possible multiplicities.

For the listing see A.2.13.
4.3 Top Level Functions

4.3.1 PreProcess

This function takes a set of \( \lambda \) values as input and generates the possible \( (\lambda, d, \rho) \) parameter triples that have their first parameter from the given set. Then it checks several simple conditions that exclude the triple from leading to a non type-1 design. First it checks if \( \lambda \) is a prime or twice a prime, since according to Theorems 2.4.1 and 2.4.5 the \( \lambda \)-design conjecture is true for these \( \lambda \) values. Next we use Theorem 2.3.7 to form \( \rho \) as a ratio of two integers. \( j \) is the denominator of \( \rho \) and \( i \) is the numerator minus the denominator. We make sure that they are relative primes and that \( i \) and \( \lambda \) are relative primes, since due to Theorem 2.4.4 otherwise we have a type-1 design. Next we check if \( \rho < \lambda - 1 \), since otherwise according to Theorem 2.4.6 we have a type-1 design. We take the possible \( d \) values, using the estimate in Theorem 2.3.9 and Lemma 3.2.2. Next we compute \( e_1 \) and \( e_2 \) using (2.38) and (2.39) and make sure that both of them are integers. We compute \( r_1 \) and \( r_2 \) using (2.40) and (2.41). Then we make sure that they are integers and that the conditions of Theorem 2.4.7 are not met. Next we compute \( t_1 \) and \( t_2 \) according to (2.156) and (2.157). We check Lemmas 2.3.10, 2.3.11 and 2.3.12. We compute \( x_1 \) and \( x_2 \) according to (2.97) and (2.98) and check the conditions of Theorem 2.4.8. Finally we check the conditions of Lemma 2.3.8, namely that the difference of \( \lceil \frac{r_2 n}{\rho + 1} \rceil \) and \( \lambda \) is a multiple of \( A \). If a \( (\lambda, d, \rho) \) parameter triple survived all these tests, then it is added to the list of values to be returned.

For the listing see A.3.1.
4.3.2 Rounding

This function is based on the observation that in each of the equations (2.101), (2.103), (2.105), (2.128), (2.130) and (2.132) the left hand side is obviously an integer, and hence the right hand side must be an integer as well. Therefore if we have an upper limit for $U_1$ or $U_2$ we can sharpen it by lowering the estimate to the first value that makes the right hand sides of the appropriate equations integer. Similarly, if we have a lower estimate for $U_1$ or $U_2$ we can sharpen it by increasing the estimate to the first value that makes the right hand sides of the appropriate equations integer.

The function performs this “rounding” for a pair of estimates (lower and upper) for either $U_1$ or $U_2$ at a time. The function takes five input variables: the first one, low, is either u1low or u2low, the lower estimate for one of the two $U$ values. The second one, high, is either u1high or u2high (with the same index as the first variable), the upper estimate for one the two $U$ values. The third variable, $r$, holds the value of $\rho$. The fourth variable, r1, is a list of two numbers, the two coefficients on the right hand side of either (2.101) or (2.128). The fifth variable, rsq, is a list of two numbers, the two coefficients on the right hand side of either (2.105) or (2.132). The function first pulls up the value of low to the next closest number that makes the right hand side of (2.101) or (2.128) integer. Then similarly it decreases the value of high to the next closest number that makes the right hand side of (2.101) or (2.128) integer. After this it increases the value of low by $\rho - 1$ repeatedly, until the right hand side of (2.105) or (2.132) becomes an integer. We have to step by multiples of $\rho - 1$ to ensure that we keep the right hand side of (2.101) or (2.128) integer. Then similarly,
we decrease the value of high by $\rho - 1$ repeatedly, until the right hand side of (2.105) or (2.132) becomes an integer. The function returns a list of two numbers, the new, improved values for low and high. Let us note that ensuring that the right hand side of (2.101) is an integer automatically guarantees that the right hand side of (2.103) is an integer as well. Similarly for (2.128) and (2.130).

For the listing see A.3.2.

4.3.3 SubRecipSum

In this function we work with equations (3.55)-(3.57). In some cases we can determine that these three conditions can not be satisfied simultaneously, even without knowing exactly what $k_m - \lambda$ values are taken. We can observe that if we form a reciprocal sum with a given number of terms from a fixed list, such that the sum of these numbers is a given value, we get the smallest reciprocal sum if the numbers are as even as possible. This function uses the above observation in the following way. It finds out what is the reciprocal sum if we choose the terms as evenly as possible from the given list, and if this reciprocal sum is larger than the prescribed reciprocal sum, we know that it is impossible to achieve that sum. The function takes four input values. The first, list, is a kml-list that includes the possible $k_m - \lambda$ values. The second, no, is the number of terms. The third, sum, is the sum of the numbers. The fourth, limit, is the reciprocal sum. The function first checks if we have a degenerate case, where some of the input numbers are 0 or negative. If all the inputs are positive, it proceeds to find the most even distribution adding up to sum. Then it computes the reciprocal
sum and checks if it is larger than limit. The function returns a Boolean variable indicating if the reciprocal sum is larger than limit.

For the listing see A.3.3.

4.3.4 SubChangeMax

This function tries to limit the set of values associated with each \( k_m - \lambda \) value, representing the possible number of blocks with this size covering a fixed point. The function takes seven input variables. The first one, list, is a km-l-list, containing the \( k_m - \lambda \) values. The second, no, is the number of terms in the sum. The third, sum, is the sum of the values. The fourth, limit, is the sum of the reciprocals. The fifth, a, is a record representing the entire table. The sixth, isu1out, is a Boolean variable showing if we are dealing with class-1 points outside the block. The seventh, isu2in, is a Boolean variable showing if we are dealing with class-2 points inside the block.

The function first tries to reduce the maximum number by using SubRecipSum. For each \( k_m - \lambda \) value it performs a binary search to determine the largest value that will not result in contradiction with the conditions of SubRecipSum. If this number is lower than the previous maximum for this \( k_m - \lambda \) value, then the maximum is lowered accordingly. Next the function calls FindUnique to determine if the \( k_m - \lambda \) values contain any unique prime powers. If a unique prime power is found in some number, then we know that the number must be a multiple of the extra prime power. If isu1out is true, then the function calls FindNecess to find those \( k_m - \lambda \) values that contain some prime power that is unique to them and has to be in the sum to
produce the required result. If such a unique power is found and there is exactly one possible multiple, then the conditions of Lemma 3.3.24 are checked.

If the \texttt{isu2in} is set to \texttt{true}, then several more steps are performed. For those numbers where some unique prime power is found, we know that there must be more then one of such blocks covering each point, and hence we can check whether we find a contradiction with Lemma 3.3.21. The function calls \texttt{FindNecess} to find those \( k_m - \lambda \) values that contain some prime power that is unique to them and has to be in the sum to produce the required result. Next we can use Lemma 3.3.22 to find a lower estimate for the number. Finally, if we can determine the exact number of blocks covering each pair, then we can use Lemma 3.3.23 to find an upper bound for the number.

At the end of the function, it checks all the \( k_m - \lambda \) values, and if the set containing the possible multiplicities is empty or contains only 0, then this \( k_m - \lambda \) value is removed. The function does not return anything.

For the listing see A.3.4.

4.3.5 \textbf{ChangeMaxRed}

This function is the parent function of \texttt{SubChangeMax}. It takes one input variable, representing the table. It cycles through all the \( kml \)-lists and calls \texttt{SubChangeMax} with the appropriate parameters. This function does not return anything.

For the listing see A.3.5.
4.3.6 IsPairOK

If there are only two $k_m - \lambda$ values left in the km-list, this function is called by IsListOK. From Theorem 2.3.6 we know that there are point pairs that are covered with a known number of blocks, so the total number of $k_m - \lambda$ values used is given by the Theorem. On the other hand from Lemma 2.3.3 we know that the reciprocal sum with the same coefficients must be a prescribed value. The function takes three inputs. The first one, pair, is a list of two numbers, the two $k_m - \lambda$ values, $x$ and $y$. The second input, rsum, is the reciprocal sum. The third input, sum, is the sum of the terms. In other words we are know that the

\[ a + b = \text{sum} \]  
(4.1)

\[ \frac{a}{x} + \frac{b}{y} = rsum \]  
(4.2)

system of equations has to have a non-negative integer solution for $a$ and $b$. The function computes $a$ and $b$ from these equations and checks if the solutions are non-negative and integers. It returns a Boolean value, showing if the conditions are satisfied.

For the listing see A.3.6.

4.3.7 IsShortOk

This function solves the equations (3.55)–(3.57). If the number of variables is at most three, we can solve these equations directly. If the number of variables is larger, we can find the solutions using a recursive process. As the number of variables grows
this gets increasingly time consuming. The global variable short determines what is
the largest number of variables where we still try to solve the system. The function
first handles the small cases. If the number of variables is at most three, it solves the
equations directly. The function returns the solution found, if there is one otherwise
it returns an empty list. If the number of variables is larger, then first all the \(k_m - \lambda\)
values are checked to find if any of them have a unique prime factor in them. If a
unique power is found it means that the multiplicity of the given \(k_m - \lambda\) value is a
multiple of the extra prime power. Using this information we compute how many
values are possible for the different variables. We chose the one that has a shortest
list of possible values. We take these possible values one by one and call IsShort0k
recursively with one fewer variables. The function returns the list of solutions found,
or an empty list if none is found.

For the listing see A.3.7.

4.3.8 IsList0K

This function is called for each km1-list whenever the table has been changed by any
test. The function takes ten input variables. The first, list, is a km1-list containing
the numbers. The second, au, is the associated \(U\) value. The third, rsum, is the sum
of the reciprocals. The fourth, sum, is the sum of the terms. The next three variables,
u, v, w, are the three variables passed down to IsShort0k. The eight variable, a, is
the table. The ninth variable, isu2in, is a Boolean value, showing if the list refers
to class-2 points outside the block. The tenth variable, force, shows if we want to
perform the most expensive tests.
First the function checks if the Diophantic equations have been solved already. If they were, and there are some solutions, then the function returns true without any further computations. If the list of numbers is empty, the function returns false. If the flag showing if the maximums have been checked is false, SubChangeMax is called. If the list is longer than the variable small, i.e. the list is too long to be checked, the function returns true. If the length of the list is 2, the function IsPairOK is called. Next, to check the list, the function IsShortOK is called. If the flag isu2in is true then the seven Solu functions are called to reduce the set of solutions found. After these reductions we check if all the $k_m - \lambda$ values appear in some solution. If a number is not used in any solution, that number can be removed. If such a number is found, than all the tests are run again. The function returns a Boolean variable, showing if any solutions were found.

For the listing see A.3.8.

4.3.9 Weeding

This function is called by CleanUp after some change has been made to the table. It takes two input variables. The first one is a record, representing the table; the second one is a Boolean variable, showing if we want to perform the most expensive tests. It cycles through the kml-lists and calls IsListOK on each one, with the appropriate parameters. If false is returned for any kml-list, then it marks the list for removal. After it checked all the lists, the marked ones are removed. Next, it resets the lower and upper estimates for $U$ and finds the critical value for $U$, i.e. the lowest $U$ value
such that, if we can raise the lower estimate to this level, we can eliminate the entire table. This function does not return anything.

For the listing see A.3.9.

4.3.10 CleanUp

This function is called after some change has been made to the table. It takes three input variables. The first one is a record, containing the table. The next two are integers, containing the number of cheap and expensive tests. First it calls Weeding to make sure all the lists are correct and to remove those that do not satisfy the conditions anymore. It checks if anything is left in the $U_1$ and $U_2$ lists. If they are not empty, then it checks if the new estimates for $U$ result in a contradiction with Lemma 2.3.13. If a contradiction is found or one of the lists is empty, then it marks the entire table done. Otherwise it resets the counters for the cheap and expensive tests, indicating that since the table has been changed, all the tests have to be redone. This function does not return any value.

For the listing see A.3.10.

4.3.11 Reduction

This function is the main loop that calls one test after the other, until the table has been eliminated or we get into a dead end. Since there is significant difference between the time needed to perform different tests, the tests are grouped into two categories, cheap and expensive tests. For efficiency this function first cycles through the cheap tests, until the table is eliminated or all the cheap tests are performed consecutively
with no change in either one. If we get through all the cheap ones than it starts with
the expensive tests, again going around until the table is eliminated or, all the tests
are performed with no change. The cycles are continued until the table is eliminated
or all the tests are performed with no change.

After each test it checks if there were any changes in the test. If there was some
change, CleanUp is called, that resets the counters, besides other things. If there
was no change, the appropriate counter ch_count or ex_count is increased by one,
showing that one more test has been performed with no change. So far Weeding has
been called with the second argument set to false, meaning that we don't want to do
the most expensive tests. If all the tests are completed with no change, then Weeding
is called one more time, but now the second parameter set to true, i.e. forcing every
test to be performed, even the most expensive ones. If this results in no change, we
are done, Reduction exits. If this last Weeding produces a change, Reduction calls
itself recursively and starts the test over again. This function returns no value.

For the listing see A.3.11.

4.3.12 PrintDesign

This function prints out the table in a formatted way. In the first line it outputs \( \lambda, d, \rho \) and \( v \). In the second line it prints the lower and upper limits for \( U \). Next it prints
the possible \( U \) values, with the corresponding possible \( k_m - \lambda \) values, the maximum
multiplicities for each \( k_m - \lambda \) value, and the solutions of the Diophantic equations,
if they were computed. After each \( U \) value, it prints also the corresponding pos,
eg, \( m21, m22, sq2 \) values. Finally at the end of the table it prints out two numbers,
showing how far the lower estimates and the upper estimates for $U$ are from resulting in a contradiction with Lemma 2.3.13. This function returns no value.

For the listing see A.3.12.

4.3.13 Test

This is the main function of the program. It takes a list of $\lambda$ values as input. First it calls PreProcess to get the $(\lambda, d, \rho)$ triples of parameters that satisfy the most easily checked conditions. It takes one such triple at a time and performs the rest of the tests on this table. It first computes $c_1, c_2, r_2, r_1,$ and $v$ using equations (2.38), (2.39), (2.36), (2.35) and (2.29) respectively. Next we compute the quantities $r_{11}, r_{12}, r_{1sq}, r_{21}, r_{22},$ and $r_{2sq}$. Each one is represented as a pair, the two numbers being the two coefficients on the right hand side of the equations (2.101), (2.103), (2.105), (2.128), (2.130), and (2.132), if they are viewed as polynomials in $U$. Then the function computes $j_{011}, j_{012}, j_{021},$ and $j_{022}$ according to (3.29), (3.30), (3.31), and (3.32). Next we compute $m_{11}, m_{12}, sq_{1}, m_{21}, m_{22},$ and $sq_{2}$ according to (3.33), (3.34), (3.35), (3.36), (3.37), and (3.38).

Next $A, t_{1}, t_{2}, x_1,$ and $x_2$ are computed according to (2.155), (2.156), (2.157), (2.97) and (2.98). We compute lower and upper estimates for $U$: $u_{1low}, u_{1high}, u_{2low}$ and $u_{2high}$ using (3.5), (3.7), (3.9), (3.11), (3.13), (3.15), (3.17), (3.19), (3.23) and (3.26). Next Rounding is called to improve on these estimates. Then we compute $q_{11}, q_{12}, q_{1sq}, q_{21}, q_{22},$ and $q_{2sq}$. Next lower estimates on $V v_{1low}$ and $v_{2low}$ are obtained using (3.27) and (3.28), and then Rounding is called to improve these estimates. Using $v_{1low}$ and $v_{2low}$ and the identities (2.164) and (2.165) we improve
on the upper estimates of $U$, and then Rounding is called again. At this point we check if the estimates are consistent with Lemma 2.3.13. If a contradiction is found, this design can be dropped. If not, we build a record named $a$ that contains all the parameters associated with the design.

We take a cycle that steps through the possible values of $U$ between the lower and the upper estimate, incrementing the value by $x - y$ according to Lemma 3.3.9 and its corollary.

For each $U_1$ value we compute $m1e v$, $m2e v$, and $sq 1 e v$, the numbers representing the values of (2.101), (2.103), and (2.105) with the particular $U$ value plugged in. Using these variables we compute pos and neg, the number of positive and negative terms in the sum (2.106). We create a list of triples, named array1, where each triple contains the $U$ value and the associated pos and neg values. Then we compute the analogous values for $U_2$.

Next we build the km1-lists starting from $\lambda$ and increasing the value by $x - y$. The upper and lower limits for the $k_m - \lambda$ values are computed by (3.43), (3.46), (3.49), and (3.52).

We make sure that we list only values that satisfy the obvious $k_m' \leq c_1$ and $k_m^* \leq c_2$ relations, or in case of the out-lists, the appropriate relation holds true with strong inequality.

We also use Lemma 3.3.25, that states that if $\lambda = -d$ then there is only one block of size $2\lambda$. After building the structure $a$, ChangeMaxRed, then Reduction is called. If the design survives all the tests, then the structure returned by Reduction is printed
by PrintDesign and included in the output. This function returns a list containing the tables that were not eliminated by the tests.

For the listing see A.3.13.

4.3.14 ReTest

This function takes as an input a list of tables. For each one it resets all the flags, showing that no tests has been done yet, and calls Reduction to perform the tests again. Finally it outputs the remaining tables the same way as Test does. The purpose of this function is to make it possible to rerun the tests on a previously tested table, with possibly changing some of the global variables, or changing the code for some of the tests, or possibly modifying the data by hand, using some ad-hoc methods, and see if the tests can further reduce this form. The function returns a list of those tables that survived all the tests.

For the listing see A.3.14.

4.4 Tests Removing one $k_m - \lambda$ value

These tests reject one $k_m - \lambda$ value at a time.

4.4.1 PrimeRed

The function takes one input variable, representing the table. This function exploits Lemma 3.3.20. It calls Trimming for each $U_1$ out-list with the fraction $\frac{1}{\rho} - \frac{1}{\lambda}$ to check if the smallest multiple mentioned in the Lemma results in a fraction less than or equal to $\frac{1}{\rho} - \frac{1}{\lambda}$. If the fraction is larger, then the given $k_m - \lambda$ value can not be part
of any sum, so it can be removed. Next it calls **Trimming** for each $U_2$ in-list with the fraction $\frac{1}{\rho}$. It returns a Boolean value showing if any numbers have been eliminated in this process.

For the listing see A.4.1.

### 4.4.2 Trimming

This function takes three inputs. The first, `list`, is a kml-list, containing the numbers used in the reciprocal sum. The second, `frac`, is a number, the result of the reciprocal sum. The third variable, `a`, represents the table. The function loops around and calls **SubTrimming** until there is no more change reported by **SubTrimming**. The function returns a Boolean value showing if any numbers have been eliminated.

For the listing see A.4.2.

### 4.4.3 SubTrimming

This function takes the same three inputs as **Trimming**. It first handles the trivial case, when the list is empty. Otherwise it calls **FindUnique** to find the unique prime powers in the $k_m - \lambda$ values. If the smallest multiple mentioned in Lemma 3.3.20, multiplied by the reciprocal of the $k_m - \lambda$ value, is larger than `frac`, then this $k_m - \lambda$ value can not be part of any reciprocal sum, and thus it can be removed. The function returns a Boolean value showing if any numbers have been eliminated.

For the listing see A.4.3.
4.4.4 CommonRed

This function takes one input variable, representing the table. This function is based on Lemma 3.4.1. It first collects the $k_m - \lambda$ values appearing in the four kinds of lists into four sets. Next it checks if every value appears in all four kinds of lists (with the exception of the three special values). If a value is found that does not appear in all four kinds of lists, then this value is removed from all the lists. The function returns a Boolean value showing if any numbers have been removed from the lists.

For the listing see A.4.4.

4.4.5 RLConstRed

This function takes one input variable, representing the table. The function is based on the fact that if for some block $k_m' \leq e_1 - 2$, then there are at least two class-1 points outside the block, and according to Lemma 2.3.3, the reciprocal sum of blocks containing these two points is $\frac{1}{p} - \frac{1}{\lambda}$. In this function we check if including particular $k_m' - \lambda$ values we can produce this number as a reciprocal sum. We work on one $U_1$ out-list at a time. For each number last in the list and for each prime factor of the number we collect all the numbers of the list that contain the prime with at least the same exponent, in a list called same. We also collect all the numbers for which $k_m' \leq e_1 - 2$ in a list called other. We compute $\frac{1}{p} - \frac{1}{\lambda} - \frac{1}{\text{last}}$ and store it in a variable called remains. The sum of the terms from same can be written as a fraction, whose denominator is the least common multiple of the numbers in same and the numerator is such that, after subtracting this fraction from remains, the denominator of the
fraction is not divisible by the prime power. We find all such fractions. For each fraction we form a list called left, that contains those numbers from other whose reciprocal is small enough to be possibly part of the sum. To end up with the right sum, the least common multiple of the possible numbers must be a multiple of the denominator of remains. We check this condition for every fraction and if we find a contradiction in each case, then the given \( k_m - \lambda \) number can not be part of a \( \frac{1}{\rho} - \frac{1}{\lambda} \) sum, and hence we can remove it from the list. The function returns a Boolean variable, showing if any numbers were removed.

For the listing see A.4.5.

4.4.6 RLSumRed

This function takes one input, representing the table. According to Lemma 2.3.3 we have to be able to produce \( \frac{1}{\rho} - \frac{1}{\lambda} \) as a reciprocal sum from the \( k_m - \lambda \) values in the \( U_1 \) out-list. This function cycles through the lines of \( U_1 \), and first checks if this test has been already performed for that line, and if not, it calls IsSumPossible with each element of the list, to check if we can produce a reciprocal sum including that number. If IsSumPossible returns false then that number can not be part of the sum, and hence we can remove it from the list. The function returns a Boolean variable showing if any numbers have been eliminated.

For the listing see A.4.6.
4.4.7 MaxRed

This function takes one input variable, representing the table. It checks if a particular block size appears in the table, what estimates can be made on the $U$ value, and if these estimates result in any contradiction with Lemma 2.3.13. The function first collects all possible $k_m - \lambda$ values in a block called int. Next it cycles through these values. For each block size it computes in1, out1, in2 and out2, the number of class–1 and class–2 points inside and outside the given block. Next it goes through all lines and checks what are the smallest and largest $U_1$ and $U_2$ values if this block size is used. Finally these values are used to check the conditions of Lemma 2.3.13. If a contradiction is found, the given $k_m - \lambda$ value is removed from all lists. The function returns a Boolean value showing if any numbers were removed.

For the listing see A.4.7.

4.4.8 LastSumRed

This function takes one input variable representing the table. According to Lemma 2.3.3 we should be able to produce a reciprocal sum from the numbers in $U_2$ in–list that add up to $\frac{1}{\rho}$. The function checks what is the minimal number of terms needed for this if a particular number is included. If the number of terms is larger than $jo22$, then we know that for a fixed $i$ for all other class–2 js $r_ij > jo22$. On the other hand according to Theorem 2.3.6, there is $j$ such that $r_ij = jo22$, and hence the number can be removed. The function returns a Boolean value, showing if any numbers have been eliminated.
For the listing see A.4.8.

### 4.4.9 SubCardRed

This function takes six input variables. The first, olst, is a kml-list. The second, a, represents the table. The next three, u, v, w, are the three values passed to IsShortOK. The last one, isu2in, is a Boolean value, showing if the kml-list represents a $U_2$ in-list. For each number in the list we form a set called sublist. This contains those $k_m - \lambda$ values that possibly can appear in the same reciprocal sum as our number. The reason to exclude some numbers is if for both $k_m$ values $k'_m \leq \lambda - 2$, hence they can have two common class-2 points and therefore appear in a $\frac{j}{p}$ sum according to Lemma 2.3.3. IsSumPossible is called to check if such a sum exists. If the answer is positive or if $k'_m > \lambda - 2$ for either one, than the number is included in sublist. If the resulting list is not longer than the global variable small.size, then IsShortOK is called. If that function finds a contradiction, the number is removed from the list. The function returns a Boolean value, showing if any numbers have been eliminated.

For the listing see A.4.9.

### 4.4.10 CardRed

This function takes one variable representing the table. It cycles through the $U_1$ out-lists and $U_2$ in-lists and calls SubCardRed with the appropriate values. The function returns a Boolean value showing if any numbers have been eliminated.

For the listing see A.4.10.
4.4.11 U2EitherRed

This function takes one input variable representing the table. The function is based on the observation that if a block size never appears in the in–list and the out–list of the same $U_2$ value, than only one block of this size is possible. (See Lemma 3.4.2) The function finds these block sizes and calls SubEither for these numbers. It returns a Boolean variable, showing if any changes were made.

For the listing see A.4.11.

4.4.12 SubEither

This function is called by U2EitherRed. It takes two input variables; num is a $k_m - \lambda$ value and a is the table. The function first checks if there is any unique prime power in num that appears only in this number. According to Lemma 3.3.20 this guarantees that more than one of this block must appear in the $\frac{1}{p}$ sum from Lemma 2.3.3. If such a unique prime power is found, than we have a contradiction on hand, since according to Lemma 3.3.20 we must have more than one such block, and on the other hand by Lemma 3.4.2 we can have at most one. In this case the number is removed from the kml–list. If no such contradiction is found, then we proceed by checking the list of possible multiplicities for this value in each list where it appears. If 1 is in the list, then max is set to 1 and set is modified accordingly. If 1 in not among the possible values, then the number can be removed from the list. The function returns a Boolean variable showing if any changes were made.

For the listing see A.4.12.
4.4.13  U2PairRed

This function takes one input representing the table. The function first finds pairs of \( k_m - \lambda \) values, such that there is some prime power that is found in these two numbers only and at least one of the two numbers can not appear alone, because the number of the reciprocals needed by Lemma 3.3.20 would be larger than \( \frac{1}{\rho} \), and hence the number can appear in a sum only if the other number is also present. For each such pair, the function checks the condition of Lemma 3.4.3. If a contradiction is found, than we know that the two values can not appear together, and hence we can remove the number that can not appear in a list without the other. The function returns a Boolean value showing if any numbers have been removed.

For the listing see A.4.13.

4.5  Tests Removing a Line

The following tests remove one line of the table at a time.

4.5.1  RijU2inRed

This function takes one input, representing the table. This function works with the in–lists for \( U_2 \). First it checks if the test has been performed already and if we want to perform the slower tests. According to Lemma 2.3.3 we know that we should be able to form a reciprocal sum from the numbers in this list that add up to \( \frac{1}{\rho} \). Next it makes the trivial estimates with the smallest and largest numbers of the list, to figure out how many terms it will take to get \( \frac{1}{\rho} \) as a reciprocal sum. Next it calls \texttt{SubSumTest} repeatedly, to find the smallest number of terms needed to form a reciprocal sum.
adding up to $\frac{1}{\rho}$. This number is called $num$ and this is a lower estimate for $r_{ij}$. We check if $num \geq \left\lfloor \frac{2n}{n+1} \right\rfloor$.

Next we check if this estimate for $r_{ij}$ leads to a contradiction with equation (2.130). If a contradiction is found, then the line is removed. At the end the remaining lines are marked as checked and the function returns a Boolean value showing if any lines were eliminated.

For the listing see A.5.1.

4.5.2 RijU1outRed

This function takes one input, representing the table. This function works with the out–lists for $U_1$. First it checks if the test has been performed already, and if we want to perform the slower tests. According to Lemma 2.3.3 we know that we should be able to form a reciprocal sum from the numbers in this list, that add up to $\frac{1}{\rho} - \frac{1}{\lambda}$.

First we select those $k_m - \lambda$ values where $k'_m \leq c_1 - 2$, i.e. there are at least two class-1 points outside the block. Next it makes the trivial estimates with the smallest and largest numbers of the list, to figure out how many terms it will take to get $\frac{1}{\rho} - \frac{1}{\lambda}$ as a reciprocal sum. Next it calls SubSumTest repeatedly, to find the smallest number of terms needed to form a reciprocal sum adding up to $\frac{1}{\rho} - \frac{1}{\lambda}$. This number is called $num$. We compute $rijmin = num - v + 2r_1$ which is a lower estimate for $r_{ij}$.

Next we check if this estimate for $r_{ij}$ leads to a contradiction with equation (2.103). If a contradiction is found, then the line is removed. At the end the remaining lines are marked as checked and the function returns a Boolean value showing if any lines were eliminated.
For the listing see A.5.2.

4.5.3  ConstRed

This function takes one input representing the table. According to Lemma 2.3.3 we have to be able to produce $\frac{1}{p} - \frac{1}{\lambda}$ as a reciprocal sum from the $k_m - \lambda$ values in the $U_1$ out-list and $\frac{1}{p}$ from the $k_m - \lambda$ values in the $U_2$ in-list. Clearly, there is no chance of doing this if the denominator of the fraction does not divide the least common multiple of the numbers in the list. This function checks this simple condition. It calls SubConstr to do the actual testing. If a contradiction is found, the entire line is removed. The function returns a Boolean variable showing if any lines were removed.

For the listing see A.5.3.

4.5.4  SubConstr

This function takes two inputs; the first, lst, is a list of numbers, the second, frac, is a fraction. The function first handles the trivial case when the list is empty. Next the function tests if the least common multiple of the numbers in lst is divisible by the denominator of frac. It returns a Boolean variable showing if a contradiction is found.

For the listing see A.5.4.

4.5.5  SinRecipSum

This function takes five input variables. The first, list, is a kml-list containing the $k_m - \lambda$ values. The second, no, is the number of terms in the sum. The third, sum, is
the sum of the terms. The fourth, limit, is the sum of the reciprocal sum. The fifth, a, represents the table. The function first finds the unique prime powers in the $k_m - \lambda$ values. This is done similarly to FindUnique. If a unique power is found in some $k_m - \lambda$ value, then the observation of Lemma 3.3.20 is used to determine the smallest number of terms that can be used of this particular value. Next, SubRecipSum is called to find if including this many terms leads to a contradiction. If a contradiction is found, the $k_m - \lambda$ value is removed. The function returns a Boolean value showing if any numbers have been removed.

For the listing see A.5.5.

4.5.6 RecipRed

This function takes one input variable, representing the table. The function calls SubRecipSum for all four kinds of km-l-lists to see if it leads to a contradiction. If it is, the entire line is removed. If no contradiction is found, then SinRecipSum is called, to see if at least some individual $k_m - \lambda$ values can be eliminated. The function returns a Boolean variable showing if any change has been done to the table.

For the listing see A.5.6.

4.6 Global Tests

The following tests check some conditions that allow us to reject the entire table if the conditions are not met.
4.6.1 SolveGauss

This is a auxiliary function called by SquareRed. It takes a 0–1 matrix representing a system of equations modulo 2. It uses standard Gauss–elimination modulo 2, bringing the matrix to an upper triangular form. The function returns a Boolean variable showing if the system has a solution.

For the listing see A.6.1.

4.6.2 SquareRed

This function takes a variable representing the table. It forms a matrix, eqs, one line a time. The next line is built in a list called list. First we build a set called kmls that consists all the $k_m - \lambda$ values. Each line represents an equation modulo 2, where the $i^{th}$ variable is the number of blocks of size $k_i - \lambda$. The first equation represents the fact that the number of all blocks equals, $v$. The first line consists all 1's and on the last number, representing the right hand side, is $v$ modulo 2. Next we find the highest power of 2 appearing in any $k_i - \lambda$ value. The following equations are based on Lemma 3.6.1. This Lemma says that $\frac{M(\rho+1)^2}{\rho} \prod_{m=1}^{v} (k_m - \lambda)$ is a complete square. Therefore for each prime the exponent in $\prod_{m=1}^{v} (k_m - \lambda)$ has the same parity as in $\frac{\lambda}{\rho}$. The second equation has a 1 in each position where the corresponding $k_m - \lambda$ value contains the highest power of 2, 0 otherwise. The last digit is 1 if $\frac{\lambda}{\rho}$ consists this prime power, and is 0 otherwise. The following equations use the lemma directly. Each line represent one prime. It has a 1 in the corresponding position, if the given prime appears in $k_i - \lambda$ with an odd power, 0 otherwise. The last digit shows the
same for \( \frac{1}{\rho} \). Finally the function calls \texttt{SolveGauss} to solve the system of equations. Depending on the answer of this function, \texttt{SquareRed} marks the table for deletion or leaves it unchanged. The function returns a Boolean variable, showing if the system of equations has a solution.

For the listing see A.6.2.

4.6.3 \texttt{GlobalSoluRed}

This function takes one input variable, representing the table. It first collects all the possible \( k_m - \lambda \) values for all four kinds of kml-lists in a variable called \texttt{allnum} and makes an upper estimate for the multiplicity of each \( k_m - \lambda \) value, similar to \texttt{RLGlobalRed}. Using these upper estimates we solve the Diophantic equations stating that the total number of blocks is \( v \), the sum of the \( k_m - \lambda \) values is \( e_1 r_1 + e_2 r_2 - v \lambda \), and the reciprocal sum is \( \rho + 2 + \frac{1}{\rho} - \frac{1}{\lambda} \). Each solution is checked against the conditions of Lemma 3.6.1, and those contradicting the lemma are removed. If no solutions remain at this point, the entire table can be eliminated. If some solutions remain, we find the highest multiplicity for each \( k_m - \lambda \) value, and list these in \texttt{maxes}. These are upper estimates for the multiplicities of blocks covering a certain point, and hence \texttt{ListMax} is called to adjust the maximum values in each kml-list. This function returns a Boolean value indicating if any changes were made.

For the listing see A.6.3
4.6.4 UniquePrime1Red

This function takes a single input, representing the table. First it checks if there is a prime power num in the denominator of \( \rho \) such that it appears only in one \( k_m - \lambda \) value. If such a \( k_m - \lambda \) value spec is found, next it checks if there is a unique number mult found, such that \( \frac{1}{\rho} - \frac{1}{\lambda} - \frac{\text{mult}}{\text{spec}} \) is non-negative and the denominator does not contain the prime power num. If there is a unique such mult value, then we can check the conditions of Lemma 3.6.2. We compute \( x \) as described in the lemma, and check if \( x \) is an integer and if the denominator of \( \frac{x}{\text{spec}} - \frac{1}{\rho} \) is not divisible by num. If either condition is not satisfied, then the table is marked for deletion and the function returns true. Otherwise the function leaves the table unchanged and returns false.

For the listing see A.6.4.

4.6.5 UniquePrime2Red

This function takes a single input, representing the table. First it checks if there is a prime power num in the denominator of \( \rho \) such that it appears only in one \( k_m - \lambda \) value. If such such a \( k_m - \lambda \) value spec is found, next it checks if there is a unique number mult found, such that \( \frac{1}{\rho} - \frac{\text{mult}}{\text{spec}} \) is non-negative and the denominator does not contain the prime power num. If there is a unique such mult value, then we can check the conditions of Lemma 3.6.3. We compute \( x \) as described in the lemma, and check if \( x \) is an integer. If the condition is not satisfied, then the table is marked for deletion and the function returns true. Otherwise the function leaves the table unchanged and returns false.
For the listing see A.6.5.

4.6.6 RLGlobalRed

This function takes one input variable, representing the table. First we find upper estimates for the overall number of blocks of a particular size. For this we observe that there must be some values from the first few lines, since otherwise we could push up the lower estimates for $U$ such that it would contradict Lemma 2.3.13. Hence we first find what are the first $U_1$ and $U_2$ values that would lead to contradiction with 2.3.13. Next we collect all $k_m - \lambda$ values in a set named allnum. Then for each $k_m - \lambda$ value we find the the sum of maximums in the in-list and out-list, form the maximum over those $U$ values that do not contradict 2.3.13, and take the minimum over $U_1$ and $U_2$. We store these values in a list called coef. The elements are the estimates for the number of blocks with a particular $k_m - \lambda$ value. Next we cycle over every prime factor of the denominator of $\frac{1}{\chi}$ which does not appear in the denominator of $\frac{1}{\rho}$. If there is no $k_m - \lambda$ value containing this prime factor, than we can not form a reciprocal sum of $\frac{1}{\rho} - \frac{1}{\chi}$, and hence we can eliminate the entire table. If there are such $k_m - \lambda$ values, we form their least common multiple, lcm. We find the smallest fraction that is larger than $\frac{1}{\chi}$ and whose denominator is lcm and the numerator is a multiple of the prime power. Next we check if the condition of Lemma 3.6.4 is satisfied, i.e. if

$$
\left( e_1 \right) \left( \frac{t}{lcm} - \frac{1}{\chi} \right) \leq \sum_{p^t \mid k_m - \lambda} \frac{coef[m]}{k_m - \lambda} \left( \frac{e_1 - k_m^t}{2} \right)
$$

(4.3)
If the condition is not met, we can remove the entire table and \texttt{true} is returned.

Otherwise the table is left unchanged and the function returns \texttt{false}.

For the listing see A.6.6.

\subsection*{4.6.7 ListMax}

This function takes four inputs. The first, \texttt{list}, is a kml-list. The second, \texttt{allnum}, is the list of all $k_m - \lambda$ values in the table. The third, \texttt{maxes}, is a list containing the number of blocks of the given size. The fourth represents the table. The function cycles through the $k_m - \lambda$ values and checks if the maximum can be improved by the new estimate. If it can, the \texttt{max} and \texttt{set} fields are adjusted accordingly. If no values are left in a \texttt{set}, then the $k_m - \lambda$ value is removed from the list. The function returns a Boolean variable, showing if any changes were made.

For the listing see A.6.7

\section*{4.7 Tests Eliminating Some Solutions of the Equations}

These tests work with the solutions found for the Diophantic equations in the case of the $U_2$ in–lists. Each one takes three inputs, a kml–list, the list of solutions and the table.

\subsection*{4.7.1 Solu1}

This test first finds for each solution those $k_m - \lambda$ values that appear with a positive multiplicity and checks if for some prime power in the denominator of $\frac{1}{p}$ there is only one $k_m - \lambda$ value containing this prime power. If a unique prime power is found, then
we compute in count the least number of such blocks that have to be included in the reciprocal sum, then we check the condition of Lemma 3.7.1. If a contradiction is found, the solution is removed. Next we check if the conditions of Lemmas 3.3.22 and 3.3.23 are met. If not, the solution can be removed.

For the listing see A.7.1.

4.7.2 Solu2

This function checks if any of the numbers that appear with a positive multiplicity in a solution have a unique prime power. If such a prime power is found, check if multiplying the reciprocal by the extra prime power produces a number over $\frac{1}{p}$. If this is the case, the solution can be removed.

For the listing see A.7.2.

4.7.3 Solu3

In this function we first find those prime powers that do not appear in $\frac{1}{p}$ and list those $k_m - \lambda$ values that have the highest power of this prime in a list called subnums. For each solution and each number that has a positive multiplicity in the solution we compute an upper estimate of the intersection with a fixed block in $C_2$ by $\lambda - k'_m - k'_s + e_1 - 1$. The multiplicity of the $k_m - \lambda$ value in a solution is an upper estimate of the number of such blocks that can cover a pair of points. We compute $\frac{1}{t} - \frac{1}{k_s - \lambda}$ discussed in (3.71) in a variable called frac and we check the condition of (3.71). If a contradiction is found, the solution is removed.

For the listing see A.7.3.
4.7.4 Solu4

In this function we consider those solutions that contain some block size with multiplicity 1. If this block size contains some prime power that is not found in the denominator of $\frac{1}{\rho}$, then we check what other $k_m - \lambda$ values have this prime power. If we find that there is exactly one other $k_m - \lambda$ value that contains a higher power of the prime such that multiplied by the extra prime power we get more than $\frac{1}{\rho}$ and this $k_m - \lambda$ value is not $\lambda$, then according to Lemma 3.7.2, we can remove the solution.

For the listing see A.7.4.

4.7.5 Solu5

We check if $\rho$ is an integer. If it is, we subtract reciprocals of the two smallest block sizes from $\frac{1}{\rho}$ and check if the rest can be produced as a reciprocal sum, by calling IsSumPossible. If the answer is no, we know that the two blocks have no two common class-2 points. Therefore we can check the conditions of Lemma 3.7.3. If a contradiction is found, the solution is removed.

For the listing see A.7.5.

4.7.6 Solu6

We take every pair of (not necessarily different) $k_m - \lambda$ values. If at least one of the values is different from both $\lambda$ and $\lambda + \rho - 1$ then the corresponding blocks are guaranteed to have at least two common points. Therefore we subtract the reciprocals of these two numbers from $\frac{1}{\rho}$ and check if the remaining fraction can be produced as a reciprocal sum. If not, then the solution can be removed.
For the listing see A.7.6.
CHAPTER V

Closing Remarks

While reading the code, an observant reader probably got the impression at certain points that the same result could have been achieved faster or more efficiently by slightly different programming techniques. This is probably true. Since this program is not designed for repeated, every-day use, rather a one-time run only, our main objective was to make sure that the code was correct, and indeed implemented exactly the intended algorithm. Also we wanted to make sure that the code is easily readable and its correctness can be verified by a reader with the least amount of effort. In order to achieve this, at some points we avoided the temptation of using some more efficient ways of achieving the same end, if we felt that it would unduly complicate the code.

Although the GAP language proved to be a pleasant and practical programming tool, there is one omission in the language that we missed a lot. Namely, that there is no way to break out from a loop. Some equivalent of the continue and break C commands would have greatly improved the readability of the code at many points.
APPENDIX A

Program Listing

A.1 Global Variables

small\_size := 3;
fast := true;

A.2 Utility Functions

A.2.1 Ceil

Ceil := function(x)

if IsInt(x) then
    return x;
else
    return Int(x+1);
fi;
end;

A.2.2 Floor

Floor := function(x)
if IsInt(x) then
    return x;
else
    return Int(x);
fi;
end;

A.2.3 RecursTestSum

RecursTestSum := function(list, sum, k, result, terms)
local i, bound;

if Length(result[2]) < Length(list) then
    if sum <= k/list[Length(result[2])].aval then
        Add(result[2], 0);
        if sum = k/list[Length(result[2])].aval and k + terms[Length(result[2])] <= list[Length(result[2])].max then
            result[1] := true;
            result[2][Length(result[2])] := k;
            for i in [Length(result[2])+1 .. Length(list)] do
                result[2][i] := 0;
            od;
    fi;
fi;

end;
else

    bound := Minimum(k-1, sum*list[Length(result[2])].aval,
                    list[Length(result[2])].max-terms[Length(result[2])]);

    while result[2][Length(result[2])] <= bound and
      not result[1] do

      result := RecursTestSum(list,sum-result[2][Length (result[2])]/list[Length(result[2])].aval,k-
                              result[2][Length(result[2])],result,terms);

      if not result[1] then

      result[2][Length(result[2])] :=
        result[2][Length(result[2])] + 1;

      fi;

    od;

    if not result[1] then

      Unbind(result[2][Length(result[2])]);

      fi;

  fi;

fi;

fi;

fi;

fi;

return result;

end;
A.2.4 SubSumTest

SubSumTest := function(lst, n_terms, frac)

local leng, factset, powers, except, num, temp, rfactset, rpowers,
max1, max2, unique, remains, terms, term_no, rpower, c, tpl, n, k,
i, sum, last, result;

leng := Length(lst);
factset := [];
powers := [];
except := [];
terms := [];
for num in lst do
    Add(factset,num.fact);
    Add(powers,num.pow);
    Add(terms,0);
od;
temp := FactorsInt(Denominator(frac));
rfactset := Set(temp);
rpowers := Collected(temp);
for num in rfactset do
    max1 := 0;
    max2 := 0;
unique := 0;

rpower := rpowers[PositionSorted(rfactset,num)][2];

for c in [1..leng] do
    if (num in factset[c]) then
        temp := powers[c][PositionSorted(factset[c],num)][2];
        if (temp > max2) and (temp < max1) then
            max2 := temp;
        fi;
        if (temp = max1) then
            max2 := max1;
            unique := 0;
        fi;
        if (temp > max1) then
            max2 := max1;
            max1 := temp;
            unique := c;
        fi;
    fi;
    fi;
od;
if (unique > 0) and (max1 >= rpower) and (max2 < rpower) then
    Add(except,[1st[unique].aval, num, max1,max2,unique]);
fi;
remains := frac;
term_no := 0;
for temp in except do
    while (RemInt(Denominator(remains),temp[2]^ (temp[4]+1)) = 0 and
        remains >= 0 and term_no <= n_terms) do
        remains := remains - 1/temp[1];
        terms[temp[5]] := terms[temp[5]] + 1;
        term_no := term_no + 1;
    od;
    od;
if remains = 0 then
    if term_no = n_terms then
        return true;
    else
        return false;
    fi;
fi;
if (remains > 0 and term_no = n_terms) then
    return false;
fi;
if (remains < 0) then

return false;
fi;
n := Length(lst);
k := n_terms - term_no;
result := [ false, [] ];
result := RecursTestSum(lst,remains,k,result,terms);
return result[1];
end;

A.2.5 MakeList

MakeList := function(nums,card)
local list, temp, next, num;

list := [];
for num in nums do
    temp := FactorsInt(num);
    next := rec(
        aval := num,
        fact := Set(temp),
        pow := Collected(temp),
        max := Int(card*num),
        per := 1,
        set := [0..Int(card*num)]
    )
end;
A.2.6 MakeFakeList

MakeFakeList := function(nums)
  local list, temp, next, num;
list := [];  
for num in nums do  
    Add(list, num.aval);  
od;

return rec(  
    leng := Length(nums),
    lst := Set(nums),
    trimmed := false,
    short := false,
    nums := Set(list),
    lastsum := false,
    u2in := false,
    u1out := false,
    r1sum := false,
    maxcheck := false,
    solu := []
)
);

end;

A.2.7 TouchList

TouchList := function(list,a)

list.trimmed := false;
list.short := false;
list.lastsum := false;
list.u2in := false;
list.uiout := false;
list.rlsum := false;
list.maxcheck := false;
list.solu := [];
list.cover := [];
a.ulist[1] := Set(a.ulist[1]);
end;

A.2.8 ListRemove

ListRemove := function(list,num,a)
local elm, listc;

listc := Copy(list.lst);
for elm in listc do
if elm.aval = num then
    RemoveSet(list.lst,elm);
fi;
od;
RemoveSet(list.nums,num);
list.leng := Length(list.nums);
TouchList(list,a);
end;

A.2.9 SumPossibleNums

SumPossibleNums := function(lst, frac)
local min, max, num, lstc, retnums;

if lst = [] then return [] fi;
if frac <= 0 then return [] fi;
min := frac*lst[1].aval;
if not IsInt(min) then min := Int(min) + 1 fi;
max := Int(frac*lst[Length(lst)].aval);
retnums := [];
for num in [min..max] do
  lstc := Copy(lst);
  if SubSumTest(lstc,num,frac) then
    Add(retnums,num);
  fi;
od;
return retnums;
end;
A.2.10 IsSumPossible

IsSumPossible := function(lst, frac)
local min, max, num, lstc;

if lst = [] then return false; fi;
if frac <= 0 then return false; fi;
min := frac*lst[1].aval;
if not IsInt(min) then min := Int(min) + 1; fi;
max := Int(frac*lst[Length(lst)].aval);
for num in [min..max] do
    lstc := Copy(lst);
    if SubSumTest(lstc, num, frac) then return true; fi;
od;
return false;
end;

A.2.11 IsSumPossibleLim

IsSumPossibleLim := function(lst, lim, frac)
local min, max, num, lstc;

if lst = [] then return false; fi;
if frac <= 0 then return false; fi;
min := frac*lst[1].aval;
if not IsInt(min) then min := Int(min) + 1; fi;
for num in [min..lim] do
    lstc := Copy(lst);
    if SubSumTest(lstc,num,frac) then return true; fi;
od;
return false;
end;

\textbf{A.2.12 FindUnique}

FindUnique := function(list,frac)
local factset, except, num, allprimes, temp, rfactset, rpowers,
max1, max2, unique, c;

factset := [];
except := [];
for num in list.lst do
    Append(factset,num факт);
    Add(except,1);
od;
allprimes := Set(factset);
temp := FactorsInt(Denominator(frac));
rfactset := Set(temp);
rpowers := Collected(temp);
for num in allprimes do
    max1 := 0;
    max2 := 0;
    unique := 0;
    if (num in rfactset) then
        maxi := rpowers[PositionSorted(rfactset, num)][2];
        max2 := maxi;
        fi;
    for c in [1..list.leng] do
        if (num in list.1st[c].fact) then
            temp := list.1st[c].pow
                [PositionSorted(list.1st[c].fact, num)][2];
            if (temp > max2) and (temp < maxi) then
                max2 := temp;
            fi;
            if (temp = maxi) then
                max2 := maxi;
                unique := 0;
            fi;
            if (temp > maxi) then
                max2 := maxi;
            fi;
        fi;
    fi;
end;

max1 := temp;
unique := c;
fi;
fi;
od;
if (unique > 0) then
    except[unique] := except[unique] * num^(max1 - max2);
fi;
od;
return except;
end;

A.2.13 FindNecess

FindNecess := function(list,frac)
local temp, rfactset, rpowers, ex, ret, c, prime, num, hits,
    spec, fr, count, ok, div;

temp := FactorsInt(Denominator(frac));
rfactset := Set(temp);
rpowers := Collected(temp);
ex := [];
ret := [];
for c in [1..list.leng] do
  Add(ex,[]);
  Add(ret,[]);
od;
for prime in rfactset do
  num := prime^powers[PositionSorted(rfactset,prime)][2];
  hits := 0;
  for c in [1..list.leng] do
    if RemInt(list.nums[c], num) = 0 then
      hits := hits+1;
      spec := c;
    fi;
  od;
  if hits = 1 then
    Add(ex[spec],num);
  fi;
od;
for c in [1..list.leng] do
  if ex[c] <> [] then
    fr := frac;
    count := 0;
    repeat
count := count + 1;
fr := fr - 1/list.nums[c];
ok := true;
for div in ex[c] do
    if RemInt(Denominator(fr),div) = 0 then
        ok := false;
    fi;
    od;
if ok then
    Add(ret[c],count);
fi;
until fr < 0 or list.lst[c].max <= count;
fi;
od;
return ret;
end;

A.3 Top Level Functions

A.3.1 PreProcess

PreProcess := function(bbb)
local i, l, j, r, prod, d, e1, e2, r1, r2, v, t1, t2, x1, x2, prelist;
prelist := [];

for l in bbb do

    if not IsPrime(l) and not(IsInt(l/2) and IsPrime(l/2)) then

        for i in [3.. l-2] do

            for j in [1.. l-1] do

                if Gcd(i,j) = 1 and Gcd(l,i) > 2 then

                    r := (i+j)/j;

                    prod:=i/Gcd(l,i);

                    if r < l-1 then

                        for d in [Int(-1*r/2 - 1/2 +1*(r-1)/(4*r -2)) .. -2] do

                            e1 := l + ( 1 + d ) / r;

                            if IsInt( e1 ) then

                                e2 := l *r - d* r - r + l;

                                if IsInt( e2 ) then

                                    r2 := e1 -d;

                                    r1 := 1 + r*( r2 - 1 );

                                    if IsInt( r1 ) then

                                        v := e1 + e2;

                                        if not (IsInt( r1/(r+1)) or

                                            IsInt(r2/(r+1))) then

                                            t1:= Int(r1/(r+1)) + 1 -r1/(r+1);


t2 := \text{Int}( r2/(r+1) ) + 1 - r2/(r+1); 

\text{if } t1 + t2 = (r-1)/(r+1) \text{ and } t1 < 
\text{(r-1)/(r+1) + d*(r-1)/((r+1)*(l*r+1-r)) } \text{ and } 
\text{t2 < (r-1)/(r+1)+}
\text{(d+1)*(r-1)/((r+1)*(l*r+1-1)) then}

x1 := d*(l*r-1)/(l*r+1- r); 

x2 := (d+1)*(l*r-1)/(l*r+1-1); 

\text{if not(IsInt((l-x1)/r) or IsInt(x2)) then}
\text{if IsInt(Int(r2*r/(r+1)-1)/prod ) then}
\text{Add(prelist, [l, d, r]);}

\text{fi;}

\text{fi;}

\text{fi;}

\text{fi;}

\text{fi;}

\text{fi;}

\text{fi;}

\text{od;}

\text{fi;}

\text{fi;}

\text{od;}

\text{od;}
fi;

od;

return prelist;

end;

A.3.2 Rounding

Rounding := function(low, high, r, r1, rsq)

if not IsInt(r1[1] + r1[2]*low) then
    if (r1[2] > 0) then
        low := (Int(r1[1] + r1[2]*low) + 1 - r1[1])/r1[2];
    else
    fi;
fi;

if not IsInt(r1[1] + r1[2]*high) then
    if (r1[2] > 0) then
        high := (Int(r1[1] + r1[2]*high) - r1[1])/r1[2];
    else
        high := (r1[1] - Int(r1[1] + r1[2]*high) - 1)/(-r1[2]);
    fi;
fi;
while not IsInt(rsq[1]+rsq[2]*low) do
    low := low + r - 1;
od;
while not IsInt(rsq[1]+rsq[2]*high) do
    high := high - (r - 1);
od;
return [low, high];
end;

A.3.3 SubRecipSum

SubRecipSum := function(list,no,sum,limit)
local aver, c, smaller, larger, extra, no_larger;

if no < 0 then return true;fi;
if no = 0 then
    return not(sum = 0 and limit <= 0);fi;
if sum < 0 then return true; fi;
if sum = 0 then
    return not(no = 0 and limit <= 0);fi;
if limit < 0 then return true; fi;
if limit = 0 then
return not(no = 0 and sum = 0);
fi;

aver := sum/no;
if list.nums = [] then return true; fi;
if list.nums[1] <= aver and aver <= list.nums[list.leng] then
  for c in [1..list.leng -1] do
    if list.nums[c] <= aver and aver <= list.nums[c+1] then
      smaller := list.nums[c];
      larger := list.nums[c+1];
      fi;
    od;
  extra := no * (aver-smaller);
  no_larger := extra/(larger-smaller);
  if (no-no_larger)*(l/smaller)+no_larger*(l/larger) > limit then
    return true;
  fi;
else
  return true;
fi;
return false;
end;
A.3.4 SubChangeMax

SubChangeMax := function(list,no,sum,limit,a,isulout,isu2in)

local c, top, bottom, mid, ret, sum_terms, sum_nums, num, unique,
kmprime, kmstar, min, max, listc, nec;

for c in [1..list.leng] do
  bottom := 0;
  top := list.lst[c].max + 1;
  while (top - bottom) > 1 do
    mid := Int((top+bottom)/2);
    sum_terms := mid/list.nums[c];
    sum_nums := mid*list.nums[c];
    ret := SubRecipSum(list,no-mid,sum-sum_nums,limit-sum_terms);
    if ret then
      top := mid;
    else
      bottom := mid;
    fi;
  od;
  if bottom < list.lst[c].max then
    list.lst[c].max := bottom;
    list.lst[c].set := Filtered(list.lst[c].set,n -> n <= bottom);
  fi;
end:

TouchList(list,a);
fi;
 od;
unique := FindUnique(list,limit);
for c in [1..list.leng] do
 if unique[c] > 1 then
   list.lst[c].per := unique[c];
   list.lst[c].set := Filtered(list.lst[c].set,
     n -> RemInt(n,unique[c])=0);
   if list.lst[c].set <> [] then
     list.lst[c].max := Maximum(list.lst[c].set);
   fi;
 fi;
 od;
if isulout then
 nec := FindNecess(list,1/a.r-1/a.l);
for c in [1..list.leng] do
 if Length(nec[c]) >= 1 then
   kmprime := a.l - (list.nums[c]-a.l)/(a.r-1);
   max := nec[c][1]*(a.e1-kmprime-1)/(a.e1-1);
   if Length(nec[c]) = 1 then
     list.lst[c].set :=
Filtered(list.lst[c].set,n->n=max);

else

list.lst[c].set :=

Filtered(list.lst[c].set,n->n>=max);

fi;

if list.lst[c].set = [] then

list.nums := [];
list.lst := [];
list.leng := 0;
list.solu := [];
return true;
else

list.lst[c].max := Maximum(list.lst[c].set);

fi;
fi;

if unique[c] > 1 then

if 2*unique[c]/list.nums[c] > 1/a.r - 1/a.l then

kmprime := a.l - (list.nums[c]-a.l)/(a.r-1);
max := unique[c]*(a.e1-kmprime-1)/(a.e1-1);
list.lst[c].set :=

Filtered(list.lst[c].set,n->n=max or n = 0);

if list.lst[c].set <> [] then
list.lst[c].max := Maximum(list.lst[c].set);

fi;
fi;
fi;

od;
fi;

if isu2in then

nec := FindNecess(list,1/a.r);

for c in [1..list.leng] do

    kmprime := a.l - (list.nums[c]-a.l)/(a.r-1);
    kmstar := a.l + (list.nums[c]-a.l)*a.r/(a.r-1);
    if a.l-(2*kmprime-a.e1+1) < 0 then
        Filtered(list.lst[c].set,n->n <= 1);
        if list.lst[c].set <> [] then
            list.lst[c].max := Maximum(list.lst[c].set);
        fi;
    fi;

if nec[c] <> [] then

    min := (nec[c][1]-1)*(kmstar-1)/
            (a.l-(2*kmprime-a.e1+1))+1;

    list.lst[c].set :=
        Filtered(list.lst[c].set,n->n=0 or n >= min);

end if;

end if;

end if;

end if;

end if;
max := nec[c][1]*(a.e2-1)/(kmstar-1);

if Length(nec[c]) = 1 then
    list.lst[c].set :=
        Filtered(list.lst[c].set,n->n=max);
else
    list.lst[c].set :=
        Filtered(list.lst[c].set,n->n>=max);
fi;

if list.lst[c].set = [] then
    list.nums := [];
    list.lst := [];
    list.leng := 0;
    list.solu := [];
    return true;
else
    list.lst[c].max := Maximum(list.lst[c].set);
    fi;
fi;

if unique[c] > 0 then
    if 2*unique[c]/list.nums[c] > 1/a.r then
        max := unique[c]*(a.e2-1)/(kmstar-1);
        list.lst[c].set :=
Filtered(list.lst[c].set, n->n=max or n=0);

if list.lst[c].set <> [] then
    list.lst[c].max := Maximum(list.lst[c].set);
    fi;
fi;
fi;
fi;
od;
fi;
listc := Copy(list);
for num in listc.lst do
    if num.set = [] or num.set = [0] then
        ListRemove(list, num.aval, a);
        fi;
    od;
list.maxcheck := true;
return;
end;

A.3.5 ChangeMaxRed

ChangeMaxRed := function(a)
local total, line;

    total := a.e1*a.r1+a.e2*a.r2-(a.e1+a.e2)*a.l;
for line in a.ulist[1] do
  SubChangeMax(line.in1,a.r1,line.au + a.r1^2/(a.r+1),a.r+1,a,
              false,false);
  SubChangeMax(line.out,a.r2-1,total-line.au-a.r1^2/(a.r+1),
              1+1/a.r-1/a.l,a,true,false);
  od;
for line in a.ulist[2] do
  SubChangeMax(line.in1,a.r2,line.au + a.r2^2*a.r/(a.r+1),
              1/a.r+1,a,false,true);
  SubChangeMax(line.out,a.r1-1,total-line.au-a.r2^2*a.r/(a.r+1),
              a.r+1-1/a.l,a,false,false);
  od;
return;
end;

A.3.6 IsPairOK

IsPairOK := function(pair,rsum, sum)
  local x,y,a,b;

  x := pair[1];
  y := pair[2];
  a := (x*y*rsum-x*sum)/(y-x);
  b := sum - a;
return IsInt(a) and IsInt(b) and a >= 0 and b >= 0;
end;

A.3.7  IsShortOK

IsShortOK := function(list,uo,vo,wo,a)
local limit, num, u, v, w, q, list, lim, i, factset, powers,
except, temp, allprimes, rfactset, npowers, max1, max2, unique,
c, maxexcept, maxnum, start, step, leng, x, y, z, aa, b, ret,
listc, solu, set, base, pair, frac, done;

leng := list.leng;
if leng = 0 then
  if uo = 0 and vo = 0 and wo = 0 then
    return [[0]];
  else
    return [];
  fi;
fi;
if uo < 0 or vo < 0 or wo < 0 then
  return [];
fi;
if uo = 0 or vo = 0 or wo = 0 then
  if uo = 0 and vo = 0 and wo = 0 then
ret := [];
for c in [1..list.leng] do
    Add(ret,0);
od;
return [ret];
else
    return [];
fi;
fi;
if leng = 1 then
    if uo/list.nums[1] = vo and uo*list.nums[1] = wo and
       uo in list.lst[1].set then
        return [[uo]];
    else
        return [];
    fi;
fi;
if leng = 2 then
    x := list.nums[1];
y := list.nums[2];
aa := (x*y*vo-x*uo)/(y-x);
b := uo - aa;
if IsInt(aa) and IsInt(b) and aa*x+b*y=wo and aa >= 0 and
  b >= 0 and aa in list.lst[1].set and
  b in list.lst[2].set then
  return [[aa,b]];
else
  return [];
fi;
fi;

if leng = 3 then
  x := list.nums[1];
  y := list.nums[2];
  z := list.nums[3];
  c := z*(wo-uo*x-uo*y+vo*x*y)/((-x+z)*(-y+z));
  b := y*(-wo+uo*x+uo*z-vo*x*z)/((-x+y)*(-y+z));
  aa := x*(wo-uo*y-uo*z+vo*y*z)/((-x+y)*(-x+z));
  if IsInt(aa) and IsInt(b) and IsInt(c) and aa >= 0 and
      b >= 0 and c >= 0 and aa in list.lst[1].set and
      b in list.lst[2].set and c in list.lst[3].set then
      return [[aa,b,c]];
  else
      return [];
  fi;
fi;

limit := [];

for num in list.1st do
    Add(limit,Int(Minimum(uo,vo*num.aval,wo/num.aval,num.max)));
od;

factset := [];
powers := [];
except := [];
base := [];

for num in list.1st do
    Add(factset,num.fact);
    Add(powers,num.pow);
    Add(except,1);
    Add(base,[]);
od;

allprimes := Set(Flat(factset));

for num in allprimes do
    max1 := 0;
    max2 := 0;
    unique := 0;
    for c in [1..leng] do
        if (num in factset[c]) then
temp := powers[c][PositionSorted(factset[c],num)][2];

if (temp > max2) and (temp < max1) then
    max2 := temp;
fi;

if (temp = max1) then
    max2 := max1;
    unique := 0;
fi;

if (temp > max1) then
    max2 := max1;
    max1 := temp;
    unique := c;
fi;
fi;

od;

if (unique > 0) then
    except[unique] := except[unique] * num^(max1 - max2);
    Add(base[unique],[num,max2]);
fi;

od;

maxexcept := list.1st[1].max;

maxnum := 1;
for c in [1..leng] do
    if limit[c]/except[c] < maxexcept then
        maxexcept := limit[c]/except[c];
        maxnum := c;
    fi;
od;
step := except[maxnum];
q := list.lst[maxnum].aval;
lim := limit[maxnum];
listc := Copy(list);
ListRemove(listc,q,a);
i := -1;
repeat
    i := i+1;
    done := true;
    frac := vo - i/q;
    for pair in base[maxnum] do
        if RemInt(Denominator(frac),pair[1]^2) = 0 then
            done := false;
        fi;
    od;
until done or frac < 0;
if frac < 0 then
    return [];
fi;
solu := [];
while i <= lim do
    u := uo - i;
    v := vo - i/q;
    w := wo - iq;
    ret := IsShortOK(listc, u, v, w, a);
    if ret <> [] then
        for set in ret do
            temp := [];
            for c in [1..maxnum-1] do
                Add(temp, set[c]);
            od;
            Add(temp, i);
            for c in [maxnum..Length(set)] do
                Add(temp, set[c]);
            od;
            Add(solu, temp);
        od;
    fi;
end;
\[ i := i + \text{step}; \]

od;

return solu;
end;

A.3.8 IsListOK

IsListOK := function(list, au, rsum, sum, u, v, w, a, isu1out, isu2in, force)
local ret, done, sol, keep, i, temp;

if list.solu <> [] then
    return true;
fi;

repeat
    done := true;
    if list.nums = [] then
        return false;
    fi;
    if not list.maxcheck then
        SubChangeMax(list, u, w, v, a, isu1out, isu2in);
    fi;
    if list.leng > a.small and not force then
        return true;
    fi;

end;
if list.leng = 2 then
    if not IsPairOK(list.nums,rsum,sum) then
        return false;
    fi;
fi;
ret := IsShortOK(list,u,v,w,a);
ret := Set(ret);
if isu2in then
    if ret <> [] then
        Solu1(list,ret,a);
    fi;
    if ret <> [] then
        Solu2(list,ret,a);
    fi;
    if ret <> [] then
        Solu3(list,ret,a);
    fi;
    if ret <> [] then
        Solu4(list,ret,a);
    fi;
    if ret <> [] then
        Solu5(list,ret,a);
fi;
if ret <> [] then
    Solu6(list,ret,a);
fi;
fi;
if ret <> [] then
    keep := [];
    for i in [1..list.leng] do
        Add(keep,[[]]);
    od;
    for sol in ret do
        for i in [1..list.leng] do
            AddSet(keep[i],sol[i]);
        od;
    od;
    for i in [1..list.leng] do
        IntersectSet(list.lst[i].set,keep[i]);
        if Length(list.lst[i].set) > 0 then
            list.lst[i].max := Maximum(list.lst[i].set);
        fi;
    od;
    temp := Copy(list);
for i in [1..temp.leng] do
    if temp.lst[i].set = [] or temp.lst[i].set = [0] then
        done := false;
        ListRemove(list,temp.nums[i],a);
    fi;
    od;
fi;
until done;
list.solu := Copy(ret);
return ret <> [];
end;

A.3.9 Weeding

Weeding := function(a,force)
local line, ulst, remlist;

remlist := [];
for line in a.ulist[1] do
    if not(IsListOK(line.inl,line.au,1,a.jo12,a.r1,a.r+1,
        line.au+a.r1^2/(a.r+1),a,false,false,false)
    and IsListOK(line.out,line.au,1/a.r,a.r1-a.jo12,a.r2-1,
        1+1/a.r-1/a.1,a.e1*a.r1+a.e2*a.r2-a.v*a.1-line.au-
        a.r1^2/(a.r+1),a,true,false,false)) then
Add(remlist,line);

fi;

od;

for line in remlist do

RemoveSet(a.ulist[1],line);

od;

if a.ulist[1] <> [] then

a.u1low := a.ulist[1][1].au;
a.u1high := a.ulist[1][Length(a.ulist[1])].au;

fi;

remlist := [];

for line in a.ulist[2] do

if not(IsListOK(line.ini, line.au, 1/a.r, a.jo22, a.r2, 1+1/a.
line.au+a.r2^2*a.r/(a.r+1), a, false, true, force and
line.au <= a.u2crit) and
IsListOK(line.out, line.au, 1, a.r2-a.jo22, a.r1-1,
a.r+1-1/a.l, a.e1*a.r1+a.e2*a.r2-a.v*a.1-line.au-
a.r2^2*a.r/(a.r+1), a, false, false, false)) then

Add(remlist,line);

fi;

od;

for line in remlist do
RemoveSet(a.ulist[2],line);

od;

if a.ulist[2] <> [] then
    a.u2low := a.ulist[2][1].au;
    a.u2high := a.ulist[2][Length(a.ulist[2])].au;
fi;

    for line in a.ulist[2] do
        if a.e1*a.r*a.uilow/(a.r-1)+a.e2*line.au/(a.r-l)-a.e1*a.e2*(a.r-l)/(a.r+l)<=0 then
            a.u2crit := line.au;
        fi;
        od;
    fi;
end;

A.3.10 CleanUp

CleanUp := function(a,ch_no,ex_no)

Weeding(a,false);

if (a.ulist[1] = []) or (a.ulist[2] = []) or
    a.e1*a.r*a.uilow/(a.r-1)+a.e2*a.u2low/(a.r-1)-a.e1*a.e2*(a.r-1)/(a.r+l) > 0 or
-a.e1*a.e2*(a.r-1)/(a.r+1)+a.e1*a.r*a.u1high/(a.r-1)+
a.e2*a.u2high/(a.r-1) < 0 then

a.ulist := [[] , []];
a.ch_count := ch_no;
a.ex_count := ex_no;
else
    a.ch_count := 0;
a.ex_count := 0;
fi;
end;

A.3.11 Reduction

Reduction := function(a)
local ch_no, ex_no, aold, i, change;

ch_no := 12;
ex_no := 6;
CleanUp(a, ch_no, ex_no);
repeat
    repeat
        if (a.ch_count < ch_no) then
            if CommonRed(a) then
                CleanUp(a, ch_no, ex_no);
            end;
        end;
    end;
end;
else
    a.ch_count := a.ch_count + 1;
fi;
fi;

if (a.ch_count < ch_no) then
    if PrimeRed(a) then
        CleanUp(a,ch_no,ex_no);
    else
        a.ch_count := a.ch_count + 1;
    fi;
fi;

if (a.ch_count < ch_no) then
    if MaxRed(a) then
        CleanUp(a,ch_no,ex_no);
    else
        a.ch_count := a.ch_count + 1;
    fi;
fi;

if (a.ch_count < ch_no) then
    if ConstRed(a) then
        CleanUp(a,ch_no,ex_no);
    else
a.ch_count := a.ch_count + 1;
fi;
fi;
if (a.ch_count < ch_no) then
  if RLConstRed(a) then
    CleanUp(a,ch_no,ex_no);
  else
    a.ch_count := a.ch_count + 1;
  fi;
fi;
if (a.ch_count < ch_no) then
  if RecipRed(a) then
    CleanUp(a,ch_no,ex_no);
  else
    a.ch_count := a.ch_count + 1;
  fi;
fi;
if (a.ch_count < ch_no) then
  if U2PairRed(a) then
    CleanUp(a,ch_no,ex_no);
  else
    a.ch_count := a.ch_count + 1;
  fi;
fi;
fi;

if (a.ch_count < ch_no) then
  if UniquePrime1Red(a) then
    CleanUp(a,ch_no,ex_no);
  else
    a.ch_count := a.ch_count + 1;
  fi;
fi;

if (a.ch_count < ch_no) then
  if UniquePrime2Red(a) then
    CleanUp(a,ch_no,ex_no);
  else
    a.ch_count := a.ch_count + 1;
  fi;
fi;

if (a.ch_count < ch_no) then
  if SquareRed(a) then
    CleanUp(a,ch_no,ex_no);
  else
    a.ch_count := a.ch_count + 1;
  fi;
fi;
fi;
if (a.ch_count < ch_no) then
  if U2EitherRed(a) then
    CleanUp(a, ch_no, ex_no);
  else
    a.ch_count := a.ch_count + 1;
  fi;
fi;
if (a.ch_count < ch_no) then
  if RLGlobalRed(a) then
    CleanUp(a, ch_no, ex_no);
  else
    a.ch_count := a.ch_count + 1;
  fi;
fi;
until (a.ch_count >= ch_no);
if (a.ex_count < ex_no) then
  if LastSumRed(a) then
    CleanUp(a, ch_no, ex_no);
  else
    a.ex_count := a.ex_count + 1;
  fi;
fi;

fi;

if (a.ex_count < ex_no) then
    if RijU2inRed(a) then
        CleanUp(a, ch_no, ex_no);
    else
        a.ex_count := a.ex_count + 1;
    fi;
fi;

fi;

if (a.ex_count < ex_no) then
    if RijU1outRed(a) then
        CleanUp(a, ch_no, ex_no);
    else
        a.ex_count := a.ex_count + 1;
    fi;
fi;

fi;

if (a.ex_count < ex_no) then
    if RLSumRed(a) then
        CleanUp(a, ch_no, ex_no);
    else
        a.ex_count := a.ex_count + 1;
    fi;
fi;
if (a.ex_count < ex_no) then
  if CardRed(a) then
    CleanUp(a,ch_no,ex_no);
  else
    a.ex_count := a.ex_count + 1;
  fi;
fi;

if (a.ex_count < ex_no) then
  if GlobalSoluRed(a) then
    CleanUp(a,ch_no,ex_no);
  else
    a.ex_count := a.ex_count + 1;
  fi;
fi;

until (a.ch_count >= ch_no and a.ex_count >= ex_no);
aold := Copy(a);
Weeding(a,true);
change := false;
for i in [1..Length(a.ulist[2])] do
  if aold.ulist[2][i].au <> a.ulist[2][i].au or
     aold.ulist[2][i].ini.nums <> a.ulist[2][i].ini.nums then
change := true;

fi;

od;

if change then
  Reduction(a);
fi;

end;

A.3.12 PrintDesign

PrintDesign := function(a)
local line, c, block, r22, r21, r2sq, jo21, jo22, sq2, m21, m22;

r22 := [a.r2*(a.e2-1)/(a.r+1), a.r/(a.r-1)];
r21 := [a.r2*a.e1*a.r/(a.r+1), -1/(a.r-1)];
r2sq := [a.r2^2 *(a.l*a.r+a.l-1)/(a.r+1), 1];
jo21 := a.r1-Int(a.r1*a.r/(a.r+1));
jo22 := a.r2 - jo21;
sq2 := r2sq + [a.e1*jo21^2+(a.e2-1)*jo22^2,0] -
  2*jo21*r21 - 2*jo22*r22;
m21 := r21 - [a.e1*jo21,0];
m22 := r22 - [(a.e2-1)*jo22,0];

Print(a.uilow,"<=U1="a.u1high,"",a.u2low,
"<=U2="a.u2high,"\n");

for block in [1..2] do
  for line in a.ulist[block] do
    Print("U",block,"="a.au,"\n");
    Print(" kmlin="a.inl.nums,"\n");
    Print(" max="[ "");
    for c in [1..line.inl.leng-1] do
      Print(line.inl.lst[c].max," ");
    od;
    Print(line.inl.lst[line.inl.leng].max," ]\n");
    if line.inl.solu <> [] then
      Print(" solu="a.inl.solu,"\n");
    fi;
    if line.inl.cover <> [] then
      Print(" cover="a.inl.cover,"\n");
    fi;
    Print(" kmlout="a.out.nums,"\n");
    Print(" max="[ "");
    for c in [1..line.out.leng-1] do
      Print(line.out.lst[c].max," ");
    od;
A.3.13 Test

Test := function(bbb)
local t1, t2, rnum, rden, e1, e2, r2, r1, v, u2, u1, k,
a, a2, l, d, r, out, s1, s2, r11, r12,
r21, r22, risq, r2sq, u1low, u2low, u1high,
u2high, zz, sq1, sq2, sq1ev, sq2ev, m11, m12, m21, m22, m11ev,
m12ev, m21ev, m22ev, jo11, jo12, jo21, jo22, array1, array2,
pos, neg, prod, primes, x1, x2, v1low, v2low, v1high, v2high,
pr1, pr2, q11, q12, q21, q22, q1sq, q2sq, trip, prelist, pair,
min, max, amax, c, kmlout, kmlin, ulist, u2list, ulist,
num, line, A, C;

out:=[ ];
prelist := PreProcess(bbb);
for trip in prelist do
  l := trip[1];
  d := trip[2];
  r := trip[3];
  rnum := Numerator(r);
  rden := Denominator(r);
  zz := rnum - rden;
  e1 := 1 + ( 1 + d ) / r;
  e2 := 1 * r - d* r - r + 1;
  r2 := e1 -d;
  r1 := 1 + r*( r2 - 1 );
\[ v := e_1 + e_2; \]

\[ r_{22} := [r_2 \cdot (e_2 - 1) / (r + 1), r / (r - 1)]; \]

\[ r_{21} := [r_2 \cdot e_1 \cdot r / (r + 1), -1 / (r - 1)]; \]

\[ r_{2s} := [r_2 \cdot 2 \cdot (1 + r + 1 - 1) / (r + 1), 1]; \]

\[ r_{12} := [r_1 \cdot e_2 / (r + 1), r / (r - 1)]; \]

\[ r_{11} := [r_1 \cdot (e_1 - 1) \cdot r / (r + 1), -1 / (r - 1)]; \]

\[ r_{1s} := [r_1 \cdot 2 \cdot (1 + r + 1 - r) / (r + 1), 1]; \]

\[ j_{011} := \text{Int}(r_1 \cdot r / (r + 1)); \]

\[ j_{012} := r_1 - j_{011}; \]

\[ j_{021} := j_{012}; \]

\[ j_{022} := r_2 - j_{021}; \]

\[ s_{01} := r_{1s} + [e_2 \cdot j_{012}^2 + (e_1 - 1) \cdot j_{011}^2, 0] - \\
    2 \cdot j_{011} \cdot r_{11} - 2 \cdot j_{012} \cdot r_{12}; \]

\[ m_{11} := r_{11} - [(e_1 - 1) \cdot j_{011}, 0]; \]

\[ m_{12} := r_{12} - [e_2 \cdot j_{012}, 0]; \]

\[ s_{02} := r_{2s} + [e_1 \cdot j_{021}^2 + (e_2 - 1) \cdot j_{022}^2, 0] - \\
    2 \cdot j_{021} \cdot r_{21} - 2 \cdot j_{022} \cdot r_{22}; \]

\[ m_{21} := r_{21} - [e_1 \cdot j_{021}, 0]; \]
\[ m_{22} := r_{22} - [(e_2 - 1) \times j_0_{22}, 0]; \]

\[ A := \frac{zz}{\text{Gcd}(1, zz)}; \]
\[ t_1 := \text{Ceil} \left( \frac{r_1}{r+1} \right) - \frac{r_1}{r+1}; \]
\[ t_2 := \text{Ceil} \left( \frac{r_2}{r+1} \right) - \frac{r_2}{r+1}; \]
\[ x_1 := d \times (1 \times r - 1) / (1 \times r + 1 - r); \]
\[ x_2 := (d + 1) \times (1 \times r - 1) / (1 \times r + 1 - 1); \]

\[ C := \frac{zz \times \text{Floor} \left( \frac{r_1}{r+1} - 1 \right)}{zz}; \]

\[ u_{\text{low}} := \frac{((r-1) \times (A \times t_1 \times (e_2 - e_1) - (v-l) \times t_1^2))}{((r-1) \times (1+A) - 2 \times t_1 \times (r+1))}; \]

\[ u_{\text{low}} := \text{Maximum} \left( u_{\text{low}}, \frac{((r-1) \times (A \times t_1 \times (e_2 + e_1 - 1) - (v-l) \times t_1^2))}{((r-1) \times (A-2 \times t_1) \times (r+1))} \right); \]

\[ u_{\text{low}} := \text{Maximum} \left( u_{\text{low}}, \frac{(r_1 / (r+1) - 1-C) \times (1+C+zz-r_1/(r+1))}{1} \right); \]

\[ u_{\text{high}} := \frac{((r-1) \times ((e_1+e_2-1) \times t_1 \times A + (v-1) \times t_1^2))}{((r+1) \times (A+2 \times t_1) - (r-1))}; \]

\[ u_{\text{high}} := \text{Minimum} \left( u_{\text{high}}, \frac{((r-1) \times ((v-1) \times t_1^2 - t_1 \times A \times (e_1-e_2-1)))}{((r-1) \times (A-1) + 2 \times t_1 \times (r+1))} \right); \]

\[ C := \frac{zz \times \text{Floor} \left( \frac{r_2}{r+1} - 1 \right)}{zz}; \]

\[ u_{\text{2low}} := \frac{((r-1) \times (A \times t_2 \times (e_2 - e_1 - 1) - (v-1) \times t_2^2))}{1}; \]
[((r-1)*(1+A)-2*t2*(r+1)));

u21ow := Maximum(u21ow,((r-1)*(A*t2*(e2+e1-1)-(v-1)*t2^-2))/
((r-1)+(A-2*t2)*(r+1)));

u21ow := Maximum(u21ow,(1/r+1)*(r*r2/(r+1)-1-C)*
(1+C+zz-r*r2/(r+1)));

u2high := ((r-1)*((e1+e2-1)*t2*A+(v-1)*t2^-2))/
((r+1)*(A+2*t2)-(r-1));

u2high := Minimum(u2high,((r-1)*((v-1)*t2^-2-t2*A*
(e1-e2+1)))/((r-1)*(A-1)+2*t2*(r+1)));

pair := Rounding(u1low, u1high, r, r11, r1sq);

u1low := pair[1];

u1high := pair[2];

pair := Rounding(u2low, u2high, r, r21, r2sq);

u2low := pair[1];

u2high := pair[2];

q11 := [(e1-1)*(r1-l+x1),1/(r-1)];

q12 := [e2*(r2-(l-x1)/r), -r/(r-1)];

q1sq := [r1^-2*(l*r+1-r)/(r+1)+(r-1)^2*(e1-1)*e2/
\[(r+1)*(l+r+1-r),-1];\]

\[q21 := [e1*(r1-1*r+r*x2), 1/(r-1)];\]
\[q22 := [(e2-1)*(r2-1+x2), -r/(r-1)];\]
\[q2sq := [r2^2*(l*r+l-1)/(r+1)+(r-1)^2*e1*(e2-1)/
((r+1)*(l+r-1-l)), -1];\]

\[C := zz*Floor(((r2-1)/(1+1/r-1/l-1)-l)/zz);\]
\[v1high := 0;\]
\[v1low := (r2-1)*(x1+2*C+1+zz)-(1+1/r-1/l)*(l+C)*(l+C+zz);\]

\[C := zz*Floor(((r1-1)/(1+r-1/l)-1)/zz);\]
\[v2high := 0;\]
\[v2low := (r1-1)*(x2+2*C+1+zz)-(1+r-1/l)*(l+C)*(l+C+zz);\]

\[pair := Rounding(v1low, v1high, r, q11, q1sq);\]
\[v1low := pair[1];\]
\[v1high := pair[2];\]

\[pair := Rounding(v2low, v2high, r, q21, q2sq);\]
\[v2low := pair[1];\]
\[v2high := pair[2];\]
pr1 := \((r-1)^2(e1-1)*e2/((r+1)*(l*r +l-r))\);
pr2 := \((r-1)^2(e2-1)*e1/((r+1)*(l*r +1-1))\);

ulhigh := \text{Minimum}(ulhigh, pr1 - v1low);

u2high := \text{Minimum}(u2high, pr2 - v2low);

pair := \text{Rounding}(u1low, u1high, r, r11, r1sq);

u1low := pair[1];

u1high := pair[2];

pair := \text{Rounding}(u21ow, u2high, r, r21, r2sq);

u2low := pair[1];

u2high := pair[2];

if \((e1*r*u1low/(r-1) + e2*u2low/(r-1)) \leq \) \\
\(e1*e2*(r-1)/(r+1)) \) \text{ and } \((e1*r*u1high/(r-1) + e2*u2high/(r-1)) \geq e1*e2*(r-1)/(r+1)) \) then

a := \text{rec}(
  l := l,
  d := d,
)
r := r,
v := v,
r1 := r1,
r2 := r2,
e1 := e1,
e2 := e2,
jo11 := jo11,
jo12 := jo12,
jo21 := jo21,
jo22 := jo22,
ulow := ulow,
ulow := ulow,
ulow := ulow,
ulow := ulow,
ulow := ulow,
e1 := e1;
e2 := e2;
repeat
\begin{verbatim}

pos := 0; neg := 0;
if m12ev > 0 then
    pos := pos + m12ev;
else
    neg := neg - m12ev;
fi;
if m11ev > 0 then
    pos := pos + m11ev;
else
    neg := neg - m11ev;
fi;
pos := pos + (sqlev - AbsInt(m11ev) - AbsInt(m12ev))/2;
neg := neg + (sqlev - AbsInt(m11ev) - AbsInt(m12ev))/2;
Add(array1, [u1, pos, neg]);

u1 := u1 + rden*(r-1);
until u1 > u1high;

array2 := [];
u2 := u2low;
\end{verbatim}
repeat
    m21ev := m21[1] + m21[2] * u2;
    pos := 0; neg := 0;
    if m21ev > 0 then
        pos := pos + m21ev;
    else
        neg := neg - m21ev;
    fi;
    if m22ev > 0 then
        pos := pos + m22ev;
    else
        neg := neg - m22ev;
    fi;
    pos := pos + (sq2ev - AbsInt(m21ev) - AbsInt(m22ev))/2;
    neg := neg + (sq2ev - AbsInt(m21ev) - AbsInt(m22ev))/2;
    Add(array2, [u2, pos, neg]);
    u2 := u2 + rden*(r-1);
until u2 > u2high;

ullist := [];

for trip in array1 do
    max := trip[2];
    min := trip[3];
    amax := Int(Maximum(-max/(rnum*jo12-rden*jo11),
                         min/(rnum*jo12-rden*jo11)));
    kmlout := [];
    for c in [0..amax] do
        num := 1+c*zz;
        if (num > (l-e1)*(r-l)+l) then
            Add(kmlout, num);
        fi;
    od;
    amax := Int(Maximum((l-rl+jo11-max)/(rnum*(jo12-1)-
                          rden*(jo11-1)),(l-rl+jo11+min)/
                        (rnum*(jo12-1)-rden*(jo11-1))));
    kmlin := [];
    for c in [0..amax] do
        num := 1+c*zz;
        if (num >= (l-e1)*(r-1)+l) then
            Add(kmlin, num);
        fi;
    od;
line := rec(
    au := trip[1],
    inl := MakeList(kmlin,1+r),
    out := MakeList(kmlout,1+1/r-1/l),
    pos := trip[2],
    neg := trip[3]);
AddSet(a.ulist[1], line);
)

u2list := [];
for trip in array2 do
    max := trip[2];
    min := trip[3];
    amax:=Int(Maximum(-max/(rnum*jo22-rden*jo21),
        min/(rnum*jo22-rden*jo21)));
    kmlout := [];
    for c in [0..amax] do
        num := l+c*zz;
        if (num >= (l-e1)*(r-1)+1) and (num <
            (e2-l)*(r-1)/r+1) then
            Add(kmlout, num);
        fi;
    od;
)
od;

amax := Int(Maximum((1-r2+jo22-max)/(rnum*(jo22-1)-
rden*(jo21-1)),(1-r2+jo22+min)/
(rnum*(jo22-1)-rden*(jo21-1))));

kmlin := [];
for c in [0..amax] do
  num := l+c*zz;
  if (num >= (l-e1)*(r-l)+l) then
    Add(kmlin,num);
  fi;
od;

line := rec(
  au := tripC l],
inl := MakeList(kmlin,1+1/r),
out := MakeList(kmlout,1+r-1/l),
pos := trip[2],
neg := trip[3]);
if 1 = -d then
  if line.inl.lst[1].aval = 1 then
    line.inl.lst[1].max := 1;
    line.inl.lst[1].set := [0,1];
  fi;
AddSet(a.ulist[2], line);

od;

ChangeMaxRed(a);

Reduction(a);

if a.ulist <> [[], []] then
    PrintDesign(a);
    Add(out, a);
    fi;

fi;

od;

return out;

end;

A.3.14 ReTest

ReTest := function(inlist)

local out, line, a, block;

out := [];

for a in inlist do
    a.small := small_size;

    for block in [1..2] do
        for line in a.ulist[block] do

TouchList(line.in1,a);
TouchList(line.out,a);

od;

od;
Reduction(a);

if a.ulist <> [[],[]] then

PrintDesign(a);
Add(out,a);

fi;

od;

return out;
end;

A.4 Tests Removing a $k_m - \lambda$ Value

A.4.1 PrimeRed

PrimeRed := function(a)

local line, change;

change := false;

for line in a.ulist[1] do
change := Trimming(line.out,1/a.r-1/a.1,a) or change;

od;

for line in a.ulist[2] do
    change := Trimming(line.inl,1/a.r,a) or change;

od;

return change;

end;

A.4.2 Trimming

Trimming := function(list,frac,a)
local diff, change;

change := false;
repeat
    diff := SubTrimming(list,frac,a);
    change := change or diff;
until not diff;
return change;
end;

A.4.3 SubTrimming

SubTrimming := function(list,frac,a)
local except, remove, c, num;
remove := [];
if list.leng = 0 then
    return false;
fi;
except := FindUnique(list,frac);
for c in [1..list.leng] do
    if (frac < except[c]*(1/list.nums[c])) then
        Add(remove,list.nums[c]);
    fi;
od;
for num in remove do
    ListRemove(list,num,a);
od;
return remove <> [];
end;

A.4.4 CommonRed

CommonRed := function(a)
local change, block, line, 11, 12, 13, 14, int, line, num, ln;

change := false;
11 := [];  
12 := [];  
13 := [];  
14 := [];  
for line in a.ulist[1] do  
    Append(11,line.in1.nums);  
    Append(12,line.out.nums);  
od;  
for line in a.ulist[2] do  
    Append(13,line.in1.nums);  
    Append(14,line.out.nums);  
od;  
int := Intersection(11, 12, 13, 14);  
AddSet(int,a.l*a.r);  
AddSet(int,(a.e2-a.l)*(a.r-1)/a.r+a.l);  
AddSet(int,(a.l-a.el)*(a.r-1)+a.l);  
for block in [1..2] do  
    for line in a.ulist[block] do  
        ln := Copy(line);  
        for num in ln.in1.nums do  
            if not num in int then  
                change := true;
ListRemove(line.inl,num,a);

fi;

od;

for num in ln.out.nums do
    if not num in int then
        change := true;
        ListRemove(line.out,num,a);
        fi;
    od;

od;

od;

od;

return change;

end;

A.4.5 RLConstRed

RLConstRed := function(a)

local line, min, lst, leng, last, factset, powers, num, temp, ok, remains, exp, same, other, c, denom, left, prime, i, change, done;

change := false;

min := a.l - (a.r-1)*(a.e1-2-a.l);

for line in a.ulist[1] do
1st := Copy(line.out.nums);
leng := Length(1st);
factset := [];
powers := [];
for num in 1st do
  temp := FactorsInt(num);
  Add(factset, Set(temp));
  Add(powers, Collected(temp));
od;
for i in [1..leng] do
  ok := true;
  last := lst[i];
  for prime in factset[i] do
    if ok then
      remains := 1/a.r - 1/a.l - 1/last;
      exp := powers[i][PositionSorted(factset[i],prime)][2];
      same := [];
      other := [];
      for c in [1..leng] do
        if prime in factset[c] then
          if exp <= powers[c][PositionSorted(factset[c],prime)][2] then

    if prime in factset[c] then
      if exp <= powers[c][PositionSorted(factset[c],prime)][2] then
Add(same, lst[c]);

fi;

fi;

if lst[c] >= min then
    Add(other, lst[c]);

fi;

od;

denom := Lcm(same);

done := false;

while not done and (remains > 0) do
    while (remains > 0) and
        (RemInt(Denominator(remains), prime^exp) = 0) do
        remains := remains - 1/denom;
    od;

    if remains = 0 then
        done := true;
    fi;

    if remains > 0 then
        left := [];
        for num in other do
            if remains >= 1/num then
                Add(left, num);
            fi;
        od;
    fi;

od;

if done = true then
    return left;
fi;

for num in other do
    if num >= 1 then
        Add(left, num);
    fi;
od;

return left;
fi;

od;

if Length(left) > 0 then
    if RemInt(Lcm(left), Denominator(remains)) = 0 then
        done := true;
    fi;
    fi;

remains := remains - 1/denom;

od;

if (remains < 0) and not done then
    ok := false;
    fi;

fi;

fi;

od;

if not ok then
    change := true;
    ListRemove(line.out, last, a);
    fi;

od;

od;

return change;
end;

A.4.6 RLSumRed

RLSumRed := function(a)

local line, lst, num, sum, change;

if fast then
    return false;
fi;

change := false;

for line in a.u[1] do
    if not line.out.rlsum then
        lst := Copy(line.out.lst);
        for num in lst do
            sum := 1/a.r -1/a.l -1/num.aval;
            if not IsSumPossible(lst,sum) then
                ListRemove(line.out,num.aval,a);
                change := true;
            fi;
        od;
        line.out.rlsum := true;
    fi;
od;
return change;
end;

A.4.7 MaxRed

MaxRed := function(a)
local block, line, int, max, in1, in2, out1, out2, u1inmax, u1inmin,
    u1outmin, u1outmax, u2inmin, u2inmax, u2outmin, u2outmax, change,
temp;

change := false;
int := [];
for block in [1..2] do
    for line in a.ulist[block] do
        Append(int, line.ini.nums);
        Append(int, line.out.nums);
od;
int := Set(int);
for max in int do
    in1 := a.l -(max-a.l)/(a.r-1);
    out1 := a.e1 - in1;
    in2 := a.l + (max-a.l)*a.r/(a.r-1);
out2 := a.e2 - in2;
ulinmax := -a.v^3;
ulinmin := a.v^3;
uloutmin := a.v^3;
uloutmax := -a.v^3;

for line in a.u list[1] do
    if max in line.ini.nums then
        ulinmax := Maximum(ulinmax,line.au);
        ulinmin := Minimum(ulinmin,line.au);
    fi;
    if max in line.out.nums then
        uloutmax := Maximum(uloutmax,line.au);
        uloutmin := Minimum(uloutmin,line.au);
    fi;
od;

for line in a.u list[2] do
    if max in line.ini.nums then
        u2inmax := Maximum(u2inmax,line.au);
    fi;
od;
\[
\begin{align*}
\text{u2inmin} & := \text{Minimum}(\text{u2inmin}, \text{line.au}); \\
\text{fi}; \\
\text{if max in line.out.nums then} & \\
\text{u2outmax} & := \text{Maximum}(\text{u2outmax}, \text{line.au}); \\
\text{u2outmin} & := \text{Minimum}(\text{u2outmin}, \text{line.au}); \\
\text{fi;}
\end{align*}
\]

\text{od;}

\text{if } \left( \left( \frac{a.r}{(a.r-1)} \right) \times (u1inmin \times i1 + u1outmin \times o1) \right.

\left. + \left( \frac{1}{(a.r-1)} \right) \times (u2inmin \times i2 + u2outmin \times o2) \right)

\left( \frac{a.e1 \times a.e2 \times (a.r-1)}{(a.r+1)} \right) \text{ or}

\left( \left( \frac{a.r}{(a.r-1)} \right) \times (u1inmax \times i1 + u1outmax \times o1) \right.

\left. + \left( \frac{1}{(a.r-1)} \right) \times (u2inmax \times i2 + u2outmax \times o2) \right)

< \frac{a.e1 \times a.e2 \times (a.r-1)}{(a.r+1)} \text{ then}

\text{change := true;}

\text{for block in [1..2] do}

\text{for line in a.ulist[block] do}

\text{if max in line.inl.nums then}

\text{ListRemove(line.inl, max, a);}

\text{fi;}

\text{if max in line.out.nums then}

\text{ListRemove(line.out, max, a);}

\text{fi;}

\end{document}
A.4.8 LastSumRed

LastSumRed := function(a)
local line, lst, curr, max, pos, trip, change;

change := false;
max := Ceil(a.r2/(a.r+1))-1;
for line in a.ulist[2] do
  if not line.ini.lastsum and
      (not fast or line.au <= a.u2crit) then
    lst := Copy(line.ini.lst);
    for curr in lst do
      if not IsSumPossibleLim(lst,max,1/a.r-1/curr.aval) then
        change := true;
        ListRemove(line.ini,c curr.aval,a);
      fi;
    od;
  fi;
od;
return change;
end;
line.inl.lastsum := true;
fi;

od;
return change;
end;

A.4.9 SubCardRed

SubCardRed := function(olst,a,u,v,w,isu2in)
local lst, lim, sublist, first, second, sum, ok, change,
 lstc, firstval, secondval;

if olst.leng < 2 then
    return false;
fi;

change := false;
lst := Copy(olst);
lim := 2*(a.r-1) + a.l;
for first in [1..lst.leng] do
    firstval := lst.nums[first];
    sublist := [lst.lst[first]]; 
    for second in [1..lst.leng] do
        if first <> second then
            secondval := lst.nums[second];
if firstval < lim and secondval < lim then
    Add(sublist,lst[second]);
else
    sum := 1/a.r - 1/firstval - 1/secondval;
    lstc := Copy(lst[1st]);
    lstc[first].max := lstc[first].max - 1;
    lstc[first].set := Filtered(lstc[first].set,n ->
        n <= lstc[first].max);
    lstc[second].max := lstc[second].max - 1;
    lstc[second].set := Filtered(lstc[second].set,n ->
        n <= lstc[second].max);
    if IsSumPossible(lstc,sum) then
        Add(sublist,lst[second]);
        fi;
        fi;
        fi;
od;
if Length(sublist) > a.small and not isu2in then
    ok := true;
else
    ok := IsShortOK(MakeFakeList(sublist),u,v,w,a) <> [];
fi;
if not ok then
    ListRemove(olst, firstval, a);
    change := true;
fi;
od;
return change;
end;

A.4.10 CardRed

CardRed := function(a)
local line, lst, change;

change := false;
if not fast then
    for line in a.ulist[1] do
        change := SubCardRed(line.out, a, a.r2-1, 1+1/a.r-1/a.1,
                              a.e1*a.r1+a.e2*a.r2-a.v*a.1-line.au-a.r1^2/(a.r+1),false)
or change;
od;
fi;
for line in a.ulist[2] do
    if not fast or line.au <= a.u2crit then
        change := SubCardRed(line.inl, a, a.r2, 1+1/a.r,
line.au + a.r^2 - 2*a.r/(a.r+1), line.au <= a.u2crit) or change;

fi;

od;

return change;

end;

A.4.11 U2EitherRed

U2EitherRed := function(a)

local change, allnum, line, delete, num, del;

change := false;

allnum := [];

for line in a.ulist[2] do

Append(allnum, line.ini.nums);

Append(allnum, line.out.nums);

od;

allnum := Set(allnum);

delete := [];

for num in allnum do

del := true;

for line in a.ulist[2] do

if num in line.ini.nums and num in line.out.nums then

del := false;
if del then
  AddSet(delete, num);
fi;
od;  

for num in delete do
  change := SubEither(num, a) or change;
od;

return change;
end;

A.4.12 SubEither

SubEither := function(num, a)

local change, remove, temp, rpowers, rfactset, prime, power,
max1, max2, unique, c, n, lpowers, lfactset, line, block, allnum,
leng, factset, powers;

change := false;
allnum := [ ];
for block in [1..2] do
  for line in a.ulist[block] do
    Append(allnum, line.inl.nums);
Append(allnum, line.out.nums);

od;

od;

allnum := Set(allnum);
leng := Length(allnum);
factset := [];
powers := [];
for n in allnum do
    temp := FactorsInt(n);
    Add(factset,Set(temp));
    Add(powers,Collected(temp));
od;

remove := [];
    temp := FactorsInt(num);
lfactset := Set(temp);
lpowers := Collected(temp);
temp := FactorsInt(Denominator(1/a.r));
rfactset := Set(temp);
rpowers := Collected(temp);
for prime in lfactset do
    power := prime^lpowers[PositionSorted(lfactset,prime)][2];
    if RemInt(Denominator(1/a.r),power) <> 0 then
max1 := 0;
max2 := 0;
unique := 0;
if (prime in rfactset) then
    max1 := rpowers[PositionSorted(rfactset,prime)][2];
    max2 := max1;
fi;
for c in [1..leng] do
    if allnum[c] <> num and prime in factset[c] then
        temp := powers[c][PositionSorted(factset[c],prime)][2];
        if (temp > max2) and (temp < max1) then
            max2 := temp;
        fi;
        if (temp = max1) then
            max2 := max1;
            unique := 0;
        fi;
        if (temp > max1) then
            max1 := temp;
            unique := c;
        fi;
    fi;
end do;
fi;

od;

if (unique > 0) then
    if (prime^(max1-max2))/allnum[unique] > 1/a.r and allnum[unique] <> a.l then
        Add(remove,allnum[unique]);
        Add(remove,num);
    fi;
fi;
fi;
fi;

od;
remove := Set(remove);
if remove <> [] then
    for n in remove do
        for block in [1..2] do
            for line in a.ulist[block] do
                ListRemove(line.in1,n,a);
                ListRemove(line.out1,n,a);
            od;
        od;
    od;
return true;
for block in [1..2] do
    for line in a.ulist[block] do
        if num in line.inl.nums then
            if line.inl.lst[PositionSorted(line.inl.nums,num)].max <> 1 then
                if 1 in line.inl.lst[PositionSorted(line.inl.nums,num)].set then
                    line.inl.lst[PositionSorted(line.inl.nums,num)].max := 1;
                    line.inl.lst[PositionSorted(line.inl.nums,num)].set := Filtered(line.inl.lst[PositionSorted(line.inl.nums,num)].set,n->n<=1);
                    TouchList(line.inl,a);
                else
                    ListRemove(line.inl,num,a);
                fi;
            fi;
        change := true;
        fi;
    fi;
    if num in line.out.nums then
        if line.out.lst[PositionSorted(line.out.nums,num)].max
<> 1 then
if 1 in line.out.lst[PositionSorted
(line.out.nums,num)].set then
line.out.lst[PositionSorted
(line.out.nums,num)].max := 1;
line.out.lst[PositionSorted(line.out.nums,num)].set
:= Filtered(line.out.lst[PositionSorted
(line.out.nums,num)].set,n->n<=1);
TouchList(line.out,a);
else
ListRemove(line.out, num, a);
fi;
change := true;
fi;
fi;
o;
o;
return change;
end;

A.4.13  U2PairRed

U2PairRed := function(a)
local allnum, line, delete, leng, factset, powers, num,
allprimes, temp, rfactset, rpowers, max1, max2, top1, top2, c, num1,
num2, found1, found2, kms1, kms2, kmp1, kmp2, both, large1,
large2, block;

allnum := [];
for block in [1..2] do
    for line in a.ulist[block] do
        Append(allnum, line.ini.nums);
        Append(allnum, line.out.nums);
    od;
od;
allnum := Set(allnum);
delete := [];
leng := Length(allnum);
factset := [];
powers := [];
for num in allnum do
    temp := FactorsInt(num);
    Add(factset, Set(temp));
    Add(powers, Collected(temp));
od;
allprimes := Set(Flat(factset));
temp := FactorsInt(Denominator(1/a.r));
rfactset := Set(temp);
rpowers := Collected(temp);
for num in allprimes do
    maxi := 0;
    max2 := 0;
    top1 := 0;
    top2 := 0;
    if (num in rfactset) then
        maxi := rpowers[PositionSorted(rfactset,num)][2];
        max2 := maxi;
    fi;
    for c in [1..leng] do
        if (num in factset[c]) then
            temp := powers[c][PositionSorted(factset[c],num)][2];
            if (temp > max2) and (temp < maxi) then
                max2 := temp;
            fi;
            if (temp = maxi) then
                if max2 < maxi then
                    if top2 <> 0 then
max2 := max1;

top1 := 0;

top2 := 0;

else

top2 := c;

fi;

fi;

fi;

if (temp > max1) then

max2 := max1;

max1 := temp;

top1 := c;

top2 := 0;

fi;

fi;

od;

if top1 > 0 and top2 > 0 then

num1 := allnum[top1];

num2 := allnum[top2];

large1 := false;

large2 := false;

if num^((max1-max2)/num1 > 1/a.r then large1 := true; fi;
if num^((max1-max2)/num2 > 1/a.r then large2 := true; fi;

kms1 := a.l + (num1-a.l)*a.r/(a.r-1);

kms2 := a.l + (num2-a.l)*a.r/(a.r-1);

kmp1 := a.l - (num1-a.l)/(a.r-1);

kmp2 := a.l - (num2-a.l)/(a.r-1);

if large1 or large2 then

found1 := false;

found2 := false;

for line in a.ulist[2] do

if (num1 in line.ini.nums and num2 in line.out.nums) then

found1 := true;

fi;

if (num2 in line.ini.nums and num1 in line.out.nums) then

found2 := true;

fi;

od;

if (kms1 > a.l and kms2 > a.l) or

(kms1 = a.l and kmp1 + a.l > a.e1) or

(kms2 = a.l and kmp2 + a.l > a.e1) then

both := true;

else

both := false;
if (both and not (found1 and found2)) or 
   (not both and not (found1 or found2)) then 
   if large1 then Add(delete, num1); fi;
   if large2 then Add(delete, num2); fi;
   fi;
fi; 
fi; 
fi; 
fi; 
fi; 
for num in delete do 
   for block in [1..2] do 
      for line in a.ulist[block] do 
         ListRemove(line.inl, num,a);
         ListRemove(line.out, num,a);
      od;
   od;
od;
return delete <> [];
end;
A.5 Tests Removing a Line

A.5.1 RijU2inRed

RijU2inRed := function(a)
local line, lst, num, done, orig, lim, change, max;

change := false;
for line in a.ulist[2] do
  if not line.inl.u2in and (not fast or
    line.au <= a.u2crit) then
    lst := line.inl.nums;
    num := Ceil(lst[1]/a.r);
    max := Floor(1/a.r*line.inl.lst[line.inl.leng].aval);
    done := false;
    while not done and num <= max do
      orig := Copy(line.inl.lst);
      if SubSumTest(orig,num,1/a.r) then
        done := true;
      fi;
      num := num + 1;
    od;
  num := num - 1;
lst := Ceil(a.r2/(a.r+1));
if num >= lim then
if num*(a.e2-1) >
(a.e2-1)*a.r2/(a.r+1)+line.au*a.r/(a.r-l) then
change := true;
line.inl.nums := [];
line.inl.1st := [];
line.inl.leng := 0;
line.inl.solu := [];
fi;
fi;
line.inl.u2in := true;
fi;
od;
return change;
end;

A.5.2 RijU1outRed

RijU1outRed := function(a)
local line, lst, sublst, num, done, orig, rijmin, change, max;
change := false;
for line in a.ulist[1] do
if not line.out.ulout and
(not fast or line.au <= a.u2crit) then

lst := line.out.lst;
sublst := [];
for num in lst do
    if a.l-(num.aval-a.l)/(a.r-l) <= a.el-2 then
        Add(sublst, num);
    fi;
od;
num := Ceil(sublst[1].aval*(1/a.r-1/a.l));
max := Floor((1/a.r-1/a.l)*line.out.lst[line.out.leng].aval);
done := false;
while not done and num <= max do
    orig := Copy(sublst);
    if SubSumTest(orig, num, 1/a.r-1/a.l) then
        done := true;
    fi;
    num := num + 1;
od;
num := num - 1;
rijmin := num - a.v + 2*a.r1;
if rijmin * (a.el-1) >
    (a.el-1)*a.r1*a.r/(a.r+1)-line.au/(a.r-1) then
line.out.nums := [];  
line.out.lst := [];  
line.out.leng := 0;  
line.out.solu := [];  
change := true;  
fi;  
line.out.ulout := true;  
fi;  
od;  
return change;  
end;  

A.5.3 ConstRed

ConstRed := function(a)  
local line, change, ulst;  
change := false;  
ulst := Copy(a.ulist);  
for line in ulst[1] do  
if SubConstr(line.out.nums,1/a.r-1/a.l) then  
change := true;  
RemoveSet(a.ulist[1],line);  
fi;  
if
od;
for line in ulst[2] do
    if SubConstr(line.inl.nums,1/a.r) then
        change := true;
        RemoveSet(a.ulist[2],line);
    fi;
od;
return change;
end;

A.5.4 SubConstr

SubConstr := function(lst, frac)

    if lst = [] then
        return true;
    fi;

    return RemInt(Lcm(lst),Denominator(frac)) <> 0;
end;

A.5.5 SinRecipSum

SinRecipSum := function(list,no,sum,limit,a)
local leng, factset, powers, num, origlst, allprimes, temp,
rfactset, rpowers, max1, max2, unique, c, no_terms, sum_terms,
change, sum_nums;

change := false;

leng := list.leng;

factset := [];

powers := [];

for num in list.nums do
    temp := FactorsInt(num);
    Add(factset, Set(temp));
    Add(powers, Collected(temp));
od;

origlst := Copy(list);

allprimes := Set(Flat(factset));

temp := FactorsInt(Denominator(sum));

rfactset := List(Set(temp));
rpowers := Collected(temp);

for num in allprimes do
    max1 := 0;
    max2 := 0;
    unique := 0;
    if (num in rfactset) then
        max1 := rpowers[PositionSorted(rfactset, num)][2];
max2 := max1;

fi;

for c in [1..leng] do

if (num in factset[c]) then

    temp := powers[c][PositionSorted(factset[c],num)][2];

    if (temp > max2) and (temp < max1) then

        max2 := temp;

    fi;

    if (temp = max1) then

        max2 := max1;

        unique := 0;

    fi;

    if (temp > max1) then

        max2 := max1;

        max1 := temp;

        unique := c;

    fi;

fi;

od;

if (unique > 0) then

    no_terms := num^(max1 - max2);

    sum_terms := no_terms/origlst.nums[unique];
sum_nums := no_terms*origlst.nums[unique];
if SubRecipSum(origlst,no-no_terms,sum-sum_nums,
    limit-sum_terms) then
    change := true;
    ListRemove(list,origlst.nums[unique],a);
fi;
fi;
od;
return change;
end;

A.5.6 RecipRed

RecipRed := function(a)
    local line, total, change;

    change := false;
    total := a.e1*a.r1+a.e2*a.r2-(a.e1+a.e2)*a.l;
    for line in a.ulist[1] do
        if SubRecipSum(line.inl,a.r1,line.au + a.r1^2/(a.r1+1),a.r1+1) then
            change := true;
            line.inl.nums := [];
            line.inl.1st := [];
            line.inl.leng := 0;
        fi;
    od;
end;
line.inl.solu := []; 
else 
    change := SinRecipSum(line.inl,a.r1,line.au + 
        a.r1^2/(a.r+1),a.r+1,a) or change; 
fi; 
if SubRecipSum(line.out,a.r2-1,total-line.au-a.r1^2/(a.r+1), 
    1+1/a.r-1/a.l) then 
    change := true; 
    line.out.nums := []; 
    line.out.1st := []; 
    line.out.leng := 0; 
    line.out.solu := []; 
else 
    change := SinRecipSum(line.out,a.r2-1, 
        total-line.au-a.r1^2/(a.r+1),1+1/a.r-1/a.l,a) or change; 
fi; 
od; 
for line in a.ulist[2] do 
    if SubRecipSum(line.inl,a.r2,line.au + 
        a.r2^2*a.r/(a.r+1),1/a.r+1) then 
        change := true; 
        line.inl.nums := [];
line.inl.1st := [];  
line.inl.leng := 0;  
line.inl.solu := [];  
else  
change := SinRecipSum(line.inl,a.r2,line.au +  
a.r2^2*a.r/(a.r+1),1/a.r+1,a)  
or change;  
fi;  
if SubRecipSum(line.out,a.r1-1,total-line.au-  
a.r2^2*a.r/(a.r+1),a.r+1-1/a.1) then  
change := true;  
line.out.nums := [];  
line.out.1st := [];  
line.out.leng := 0;  
line.out.solu := [];  
else  
change := SinRecipSum(line.out,a.r1-1,  
total-line.au-a.r2^2*a.r/(a.r+1),a.r+1-1/a.1,a) or change;  
fi;  
od;  
return change;  
end;
A.6 Global Tests

A.6.1 SolveGauss

SolveGauss := function(mat)
local m, n, i, k, j, row, w;

m := Length(mat);
n := Length(mat[1]);
i := 0;
for k in [1 .. n] do
  j := i + 1;
  while j <= m and mat[j][k] = 0 do
    j := j + 1;
  od;
  if j <= m then
    i := i + 1;
    row := mat[j];
    mat[j] := mat[i];
    mat[i] := row;
    for j in [1 .. m] do
      if i <> j and mat[j][k] <> 0 then
        for w in [1..n] do
          mat[j][w] := RemInt(mat[j][w] + mat[i][w], 2);
        od;
      od;
  od;
end:

for i in [l..m] do
  if Sum(mat[i]) = 1 and mat[i][n] = 1 then
    return false;
  fi;
od;

for i in [1..m] do
  if Sum(mat[i]) = 1 and mat[i][n] = 1 then
    return false;
  fi;
od;
return true;
end;

A.6.2 SquareRed

SquareRed := function(a)
local kmls, block, line, leng, factset, powers, num, temp,
  Ifact, lpowers, rfact, rpowers, allprimes, eqs, prime, c, solved,
solutions, k, ok, prime_no, lst, pow2, max2;

kmls := [];
for block in [1..2] do
  for line in a.ulist[block] do
    Append(kmls,line.inl.nums);
  od;
fi;
for i in [l..m] do
  if Sum(mat[i]) = 1 and mat[i][n] = 1 then
    return false;
  fi;
od;
return true;
end;
Append(kmls,line.out.nums);

od;

od;

kmls := Set(kmls);
leng := Length(kmls);

factset := [];
powers := [];

for num in kmls do
    temp := FactorsInt(num);
    Add(factset,Set(temp));
    Add(powers,Collected(temp));
od;

temp := FactorsInt(Numerator(a.1/a.r));

lfact := Set(temp);
lpowers := Collected(temp);

temp := FactorsInt(Denominator(a.1/a.r));

rfact := Set(temp);
rpowers := Collected(temp);

allprimes := Set(Flat(factset));

prime_no := Length(allprimes);

eqs := [];

line := [];

for c in [1..leng] do
    Add(line,1);
od;
Add(line,RemInt(a.v,2));
Add(eqs,line);
max2 := 0;
for c in [1..leng] do
    if 2 in factset[c] then
        pow2 := powers[c][PositionSorted(factset[c],2)][2];
        if pow2 > max2 then max2 := pow2; fi;
    fi;
od;
line := [];
for c in [1..leng] do
    if 2 in factset[c] then
        pow2 := powers[c][PositionSorted(factset[c],2)][2];
        if pow2 = max2 then
            Add(line,1);
        else
            Add(line,0);
        fi;
    else
        Add(line,0);
    fi;
ext
Add(line,0);

fi;

od;

if RemInt(Denominator(1/a.r - 1/a.l + a.r),2^max2) = 0 then
    Add(line,1);
else
    Add(line,0);
fi;

Add(eqs,line);

for prime in allprimes do
    line := [];
    for c in [1..leng] do
        if prime in factset[c] then
            Add(line,RemInt(powers[c]
            [PositionSorted(factset[c].prime)][2],2));
        else
            Add(line,0);
        fi;
    od;
    num := 0;
    if prime in lfact then
        num := num + lpowers[PositionSorted(lfact,prime)][2];
fi;
if prime in rfact then
    num := num + rpowers[PositionSorted(rfact,prime)][2];
fi;
Add(line,RemInt(num,2));
Add(eqs,line);
od;
if not SolveGauss(eqs) then
    a.ulist := [[],[]];
    return true;
else
    return false;
fi;
end;

A.6.3 GlobalSoluRed

GlobalSoluRed := function(a)
local u1num, u2num, line, allnum, block, coef, i, coef1, coef2,
    num, sum, list, length, ret, cret, sol, change, maxes;

u1num := 0;
for line in a.ulist[1] do
if $a.e1*a.r*\text{line}.au/(a.r-1)+a.e2*a.u2\text{low}/(a.r-1)-$
\hspace{1em}a.e1*a.e2*(a.r-1)/(a.r+1) \leq 0$ then
\hspace{1em}$\text{u1num} := \text{u1num} + 1;$
fi;
\hspace{1em}od;
\hspace{1em}$\text{u2num} := 0;$
\hspace{1em}for $\text{line}$ in $\text{a.ulist}[2]$ do
\hspace{2em}if $a.e1*a.r*\text{line}.au/(a.r-1)+a.e2*\text{line}.au/(a.r-1)-$
\hspace{3em}a.e1*a.e2*(a.r-1)/(a.r+1) \leq 0$ then
\hspace{4em}$\text{u2num} := \text{u2num} + 1;$
\hspace{3em}fi;
\hspace{2em}od;
\hspace{1em}$\text{allnum} := [];$
\hspace{1em}for $\text{block}$ in $[1..2]$ do
\hspace{2em}for $\text{line}$ in $\text{a.ulist}[\text{block}]$ do
\hspace{3em}$\text{Append}($allnum, $\text{line}.\text{in1}.\text{nums});$
\hspace{3em}$\text{Append}($allnum, $\text{line}.\text{out}.\text{nums});$
\hspace{2em}od;
\hspace{1em}od;
\hspace{1em}$\text{allnum} := \text{Set}($allnum$);$ 
\hspace{1em}$\text{coef} := [];$
\hspace{1em}for $\text{num}$ in $\text{allnum}$ do
coef1 := 0;
coef2 := 0;
for i in [1..u1num] do
    line := a.ulist[1][i];
    sum := 0;
    if num in line.inl.nums then
        sum := line.inl.lst[PositionSorted(line.inl.nums,num)].max;
    fi;
    if num in line.out.nums then
        sum := sum + line.out.lst[PositionSorted(line.out.nums,num)].max;
    fi;
    coef1 := Maximum(coef1,sum);
od;
for i in [1..u2num] do
    line := a.ulist[2][i];
    sum := 0;
    if num in line.inl.nums then
        sum := line.inl.lst[PositionSorted(line.inl.nums,num)].max;
    fi;
if num in line.out.nums then
    sum := sum + line.out.lst[PositionSorted(line.out.nums,num)].max;
fi;
coefficient2 := Maximum(coefficient2,sum);
end;
Add(coef,Minimum(coef1,coefficient2));
end;
list := MakeList(allnum,a.v);
length := Length(allnum);
for i in [1..length] do
    list.lst[i].max := coef[i];
end;
result := IsShortOK(list,a.v,a.e1*a.r1+a.e2*a.r2-a.v*a.l,
a.r+2+1/a.r-1/a.l,a);
result := Copy(result);
for solution in result do
    num := a.l*(a.r+1)^2/a.r;
    for i in [1..length] do
        if RemInt(sol[i],2) = 1 then
            num := num * allnum[i];
        fi;
    end;
    for j in [1..length] do
        if RemInt(sol[j],2) = 1 then
            num := num * allnum[j];
        fi;
    end;
end;
if not (RootInt(Numerator(num))^2=Numerator(num) and
    RootInt(Denominator(num))^2=Denominator(num))
then
    RemoveSet(ret,sol);
fi;
fi;
maxes := [];
for i in [1..length] do
    Add(maxes,0);
od;
for sol in ret do
    for i in [1..length] do
        maxes[i] := Maximum(maxes[i],sol[i]);
    od;
    od;
change := false;
for block in [1..2] do
for line in a.ulist[block] do
    change := ListMax(line.inl.allnum, maxes, a) or change;
    change := ListMax(line.out.allnum, maxes, a) or change;
    od;
od;
return change;
end;

A.6.4 UniquePrime1Red

UniquePrime1Red := function(a)
local kms, line, temp, rfactset, rpowers, prime, num, hits, km, spec, count, frac, mult, x, kmp, smult;

dms := [];
for line in a.ulist[1] do
    Append(kms, line.inl.nums);
    Append(kms, line.out.nums);
    od;

kms := Set(kms);
temp := FactorsInt(Denominator(1/a.r));
rfactset := Set(temp);
rpowers := Collected(temp);
for prime in rfactset do
num := prime^rpowers[PositionSorted(rfactset.prime)][2];

if RemInt(Denominator(1/a.l),num) <> 0 then
    hits := 0;
    for km in kms do
        if RemInt(km, num) = 0 then
            hits := hits+1;
            spec := km;
        fi;
    od;
    if hits = 1 then
        count := 0;
        frac := 1/a.r - 1/a.l;
        mult := 0;
        while frac > 0 and count < 2 do
            frac := frac - 1/spec;
            mult := mult + 1;
            if RemInt(Denominator(frac),num) <> 0 then
                count := count + 1;
                smult := mult;
            fi;
        od;
        if count = 1 then
kmp := a.l - (spec - a.l)/(a.r-1);
x := smult * a.e1*(a.e1-1)/((a.e1-kmp)*(a.e1-kmp-1));
if (not IsInt(x)) or
   (RemInt(Denominator(x/spec -1/a.r),num) = 0) then
   a.ulist := [ [], [] ];
   return true;
   fi;
   fi;
   fi;
   fi;
od;
return false;
end;

A.6.5 UniquePrime2Red

UniquePrime2Red := function(a)
   local kms, line, temp, rfactset, rpowers, prime, num, hits, km,
   spec, count, frac, mult, x, kms, smult;

   kms := [];
   for line in a.ulist[1] do
      Append(kms,line.inl.nums);
      Append(kms,line.out.nums);
od;
kms := Set(kms);

temp := FactorsInt(Denominator(1/a.r));

rfactset := Set(temp);

rpowers := Collected(temp);

for prime in rfactset do
    num := prime^rpowers[PositionSorted(rfactset,prime)][2];
    hits := 0;
    for km in kms do
        if RemInt(km, num) = 0 then
            hits := hits+1;
            spec := km;
            fi;
        od;
    if hits = 1 then
        count := 0;
        frac := 1/a.r;
        mult := 0;
        while frac > 0 and count < 2 do
            frac := frac - 1/spec;
            mult := mult + 1;
            if RemInt(Denominator(frac),num) <> 0 then
count := count + 1;
smult := mult;
fi;
od;
if count = 1 then
  kms := a.l + (spec - a.l)*a.r/(a.r-1);
x := smult * a.e2*(a.e2-1)/(kms*(kms-1));
if not IsInt(x) then
  a.ulist := [[];[]];
  return true;
  fi;
  fi;
  fi;
od;
return false;
end;

A.6.6 RLGlobaRed

RLGlobalRed := function(a)
local change, uinum, u2num, line, allnum, block, coef, coef1, coef2, i, sum, prime, power, frac, subset, lcm, kmprime, lhs, rhs, allnum, lfactset, lpowers, temp, num;
change := false;
ulnum := 0;
for line in a.ulist[1] do
    if a.e1*a.r*line.au/(a.r-1)+a.e2*a.u2low/(a.r-1)-
    a.e1*a.e2*(a.r-l)/(a.r+l) <= 0 then
        ulnum := ulnum + 1;
    fi;
od;
u2num := 0;
for line in a.ulist[2] do
    if a.e1*a.r*a.uilow/(a.r-1)+a.e2*line.au/(a.r-1)-
    a.e1*a.e2*(a.r-l)/(a.r+l) <= 0 then
        u2num := u2num + 1;
    fi;
od;
allnum := [];
for block in [1..2] do
    for line in a.ulist[block] do
        Append(allnum, line.inl.nums);
        Append(allnum, line.out.nums);
od;
od;
allnum := Set(allnum);
coef := [];
for num in allnum do
  coef1 := 0;
  coef2 := 0;
  for i in [1..u1num] do
    line := a.ulist[1][i];
    sum := 0;
    if num in line.inl.nums then
      sum := line.inl.lst[PositionSorted(line.inl.nums,num)].max;
    fi;
    if num in line.out.nums then
      sum := sum + line.out.lst[PositionSorted(line.out.nums,num)].max;
    fi;
    coef1 := Maximum(coef1,sum);
  od;
  for i in [1..u2num] do
    line := a.ulist[2][i];
    sum := 0;
    if num in line.inl.nums then
sum := line.inl.lst[PositionSorted(line.inl.nums, num)].max;
fi;
if num in line.out.nums then
  sum := sum + line.out.lst[PositionSorted(line.out.nums, num)].max;
fi;
coef2 := Maximum(coef2, sum);
od;
Add(coef, Minimum(coef1, coef2));

temp := FactorsInt(Denominator(1/a.1));
lfactset := Set(temp);
lpowers := Collected(temp);
for prime in lfactset do
  power := prime^lpowers[PositionSorted(lfactset, prime)][2];
  if RemInt(Denominator(1/a.r), power) <> 0 then
    subset := Filtered(allnum, n->RemInt(n, power)=0);
    if subset = [] then
      a.ulist := [[], []];
      return true;
    fi;
  fi;
"
frac := -1/a.l;
lcm := Lcm(subset);
repeat
  frac := frac + 1/lcm;
until (frac > a.v/\text{subset}[1]) or
  (frac > 0 \text{ and } \text{RemInt(Denominator(frac+1/a.l),power)} <> 0);
rhs := 0;
for \text{num in subset do}
  kmprime := a.l - (\text{num-}a.l)/(a.r-1);
  rhs := rhs + (a.e1-kmprime)*(a.e1-kmprime-1)/2*
    \text{coef[PositionSorted(allnum,num)]/num};
end;
lhs := a.e1*(a.e1-1)*frac/2;
if lhs > rhs then
  change := true;
  a.ulist := [[];[[]];
fi;
fi;
end;
return change;
A.6.7 ListMax

ListMax := function(list, allnums, maxes, a)

local num, change, clist, i, max;

change := false;
clist := Copy(list);
for i in [1..clist.leng] do
    num := clist.nums[i];
    max := maxes[PositionSorted(allnums, num)];
    if list.lst[i].max > max then
        change := true;
        list.lst[i].set := Filtered(list.lst[i].set, n -> n <= max);
        if list.lst[i].set <> [] then
            list.lst[i].max := Maximum(list.lst[i].set);
        fi;
        TouchList(list, a);
    fi;
od;
clist := Copy(list);
for num in clist.lst do
    if num.set = [] or num.set = [0] then
        ListRemove(list, num.aval, a);
A.7 Tests Eliminating Some Solutions of the Equations

A.7.1 Solu1

Solu1 := function(list, ret, a)
local temp, cret, rfactset, rpowers, prime, power, hits,
special, c, sol, count, scount, num, kmp, kms;

cret := Copy(ret);
temp := FactorsInt(Denominator(1/a.r));
rfactset := Set(temp);
rpowers := Collected(temp);
for sol in cret do
  for prime in rfactset do
    power := prime^rpowers[PositionSorted(rfactset, prime)][2];
    hits := 0;
    special := 0;
    for c in [1..list.leng] do
      if sol[c] > 0 and RemInt(list.nums[c], power) = 0 then

hits := hits + 1;
special := c;
fi;
od;
if hits = 1 then
  count := 0;
  num := 0;
repeat
  count := count + 1;
  if RemInt(Denominator(1/a.r-count/list.nums[special]),
             power) <> 0 then
    scount := count;
    num := num + 1;
  fi;
until num > 1 or 1/a.r-count/list.nums[special] < 0;
kms := a.l+(list.nums[special]-a.l)*a.r/(a.r-1);
kmr := a.l-(list.nums[special]-a.l)/(a.r-1);
if sol[special]*(kms-l) < scount*(a.e2 - 1) then
  RemoveSet(ret,sol);
fi;
if sol[special] < (scount-l)*(kms-l)/
(a.l-(2*kmr-a.e1+l)) + 1 then
RemoveSet(ret, sol);
fi;
if num = 1 then
  if sol[special] *(kms-1) <> scount*(a.e2 - 1) then
    RemoveSet(ret, sol);
  fi;
fi;
fi;
fi;
od;
od;
return;
end;

A.7.2 Solu2

Solu2 := function(list, ret, a)
local temp, cret, sol, factset, except, num, allprimes, rfactset,
   rpowers, max1, max2, unique, c;

cret := Copy(ret);
for sol in cret do
  if 0 in sol then
    factset := [];
    except := [];

for num in list.1st do
    Append(factset,num.fact);
    Add(except,1);
od;
allprimes := Set(factset);
temp := FactorsInt(Denominator(1/a.r));
ractset := Set(temp);
rpowers := Collected(temp);
for num in allprimes do
    maxi := 0;
    max2 := 0;
    unique := 0;
    if (num in ractset) then
        maxi := rpowers[PositionSorted(ractset,num)][2];
        max2 := maxi;
    fi;
    for c in [1..list.leng] do
        if (num in list.1st[c].fact and sol[c] > 0) then
            temp := list.1st[c].pow
                [PositionSorted(list.1st[c].fact,num)][2];
            if (temp > max2) and (temp < maxi) then
                max2 := temp;
            fi;
        fi;
    od;
end;

if (temp = max1) then
  max2 := max1;
  unique := 0;
fi;
if (temp > max1) then
  max2 := max1;
  max1 := temp;
  unique := c;
fi;
fi;
fi;
od;
if (unique > 0) then
  if (unique > 0) then
    except[unique] := except[unique] * num^((max1 - max2);
  fi;
od;
for c in [1..list.leng] do
  if except[c]/list.nums[c] > 1/a.r and sol[c] > 0 then
    RemoveSet(ret,sol);
  fi;
od;
fi;
A.7.3 Solu3

Solu3 := function(list, ret, a)
local temp, cret, rfactset, rpowers, prime, power, sol, factset,
powers, leng, allprimes, max, subnums, kmprimes, kmstars, i, j,
lcm, subleng, frac, rhs, lhs, num, position, max2, subsub;

cret := Copy(ret);
temp := FactorsInt(Denominator(1/a.r));
rfactset := Set(temp);
rpowers := Collected(temp);
factset := [];
powers := [];
for num in list.lst do
  Add(factset,num.fact);
  Add(powers,num.pow);
end;
leng := list.leng;
allprimes := Set(Flat(factset));
for prime in allprimes do
max := 0;
for i in [l..leng] do
    if prime in factset[i] then
        max := Maximum(max, powers[i]
        [PositionSorted(factset[i], prime)][2]);
    fi;
od;
power := prime^max;
if RemInt(Denominator(1/a.r), power) <> 0 then
    subnums := [];
    kmprimes := [];
    kmstars := [];
    max2 := 0;
    for i in [l..leng] do
        if prime in factset[i] then
            if powers[i][PositionSorted(factset[i],
                prime)][2] = max then
                Add(subnums, list.nums[i]);
                Add(kmprimes, a.l-(list.nums[i]-a.l)/(a.r-1));
                Add(kmstars, a.l+((list.nums[i]-a.l)*a.r)/(a.r-1));
            else
                max2 := Maximum(powers[i][PositionSorted(factset[i],
                    prime)][2], max2);
            fi;
        fi;
    od;
end;
prime)][2], max2);
fi;
fi;
od;
subleng := Length(subnums);
for j in [1..subleng] do
  for sol in cret do
    if sol[PositionSorted(list.nums, subnums[j])] > 0 then
      subsub := Filtered(subnums, n->sol[PositionSorted(list.nums, n)] > 0);
lcm := Lcm(subsub);
frac := -1/subnums[j];
repeat
  frac := frac + prime^(max-max2)/lcm;
until frac >= 0 and IsSumPossible(Sublist(list.lst, Filtered([1..leng], n->sol[n] > 0)), frac);
 rhs := (kmstars[j]-1+a.e1-kmprimess[j])*frac;
lhs := 0;
for i in [1..leng] do
  if sol[i] > 0 then
    if list.nums[i] in subnums then
      position := PositionSorted(subnums, list.nums[i]);
if list.nums[i] = subnums[j] then
    lhs := lhs + 1/subnums[position] * 
    (a.l-kmprimes[position]-kmprimes[j]+a.e1-1) 
    * (sol[i]-1);
else
    lhs := lhs + 1/subnums[position] * 
    (a.l-kmprimes[position]-kmprimes[j]+a.e1-1) 
    * sol[i];
fi;
fi;
fi;
fi;
if lhs < rhs then
    RemoveSet(ret,sol);
fi;
fi;
fi;
od;
if lhs < rhs then
    RemoveSet(ret,sol);
fi;
fi;
od;
of;
return;
end;
A.7.4 Solu4

Solu4 := function(list, ret, a)
local temp, rfactset, rpowers, num, c, ret, leng, solu, lfactset,
lpowers, prime, power, max1, max2, unique, i, c, factset, powers;

temp := FactorsInt(Denominator(1/a.r));
rfactset := Set(temp);
rpowers := Collected(temp);
factset := [];
powers := [];
for num in list.1st do
  Add(factset,num.fact);
  Add(powers,num.pow);
od;
cret := Copy(ret);
leng := list.leng;
for solu in cret do
  for c in [1..leng] do
    if solu[c] = 1 then
      lfactset := list.lst[c].fact;
      lpowers := list.lst[c].pow;
      for prime in lfactset do

power := prime

lpowers[PositionSorted(lfactset, prime)][2];

if RemInt(Denominator(1/a.r), power) <> 0 then

max1 := 0;
max2 := 0;
unique := 0;

if (prime in rfactset) then

max1 := rpowers[PositionSorted(rfactset, prime)][2];
max2 := max1;
fi;

for i in [1..leng] do

if i <> c and prime in factset[i] and solu[i] > 0 then

temp := powers[i]

[PositionSorted(factset[i], prime)][2];

if (temp > max2) and (temp < max1) then

max2 := temp;
fi;
if (temp = max1) then

max2 := max1;
unique := 0;
fi;
if (temp > max1) then
max2 := max1;
max1 := temp;
unique := i;
fi;
fi;
od;
if (unique > 0) then
    if (prime^(max1-max2))/list.nums[unique] > 1/a.r and list.nums[unique] <> a.l then
        RemoveSet(ret,solu);
    fi;
    fi;
fi;
fi;
fi;
od;
fi;
od;
od;
return;
end;

A.7.5 Solu5

Solu5 := function(list, ret, a)
local cret, solu, listc, c;

if not IsInt(a.r) then
    return;
fi;
if list.leng < 2 then
    return;
fi;
if list.nums[1] <> a.l or list.nums[2] <> a.l+a.r-1 then
    return;
fi;
cret := Copy(ret);
for solu in cret do
    if solu[1] > 0 and solu[2] > 0 then
        listc := Copy(list.1st);
        for c in [1 .. list.leng] do
            listc[c].max := solu[c];
            listc[c].set := Filtered(listc[c].set,n ->
                n <= solu[c]);
        od;
        listc[1].max := listc[1].max-1;
        listc[1].max := Filtered(listc[1].set,n ->

}
n <= listc[1].max);
listc[2].max := listc[2].max-1;
listc[2].max := Filtered(listc[2].set,n ->
    n <= listc[2].max);
if not IsSumPossible(listc, 1/a.r-1/
    list.nums[1]-1/list.nums[2]) then
    if (a.1-1)*solu[1] > a.e2-1-list.nums[2] or
        solu[1] > a.e1-a.1+1 then
        RemoveSet(ret,solu);
    fi;
fi;
fi;
fi;
od;
return;
end;

A.7.6  Solu6

Solu6 := function(list, ret, a)
local cret, solu, first, second, skip, listc, c;

if list.leng < 2 then
    return;
fi;
cret := Copy(ret);

for solu in cret do
  for first in [1..list.leng] do
    if solu[first] > 0 then
      for second in [first..list.leng] do
        if solu[second] > 0 then
          skip := false;
          if list.nums[first] = a.l and
            list.nums[second] = a.l then
            skip := true;
          fi;
          if IsInt(a.r) and list.nums[second] = a.l+a.r-1 then
            skip := true;
          fi;
        fi;
      if not skip then
        listc := Copy(list.lst);
        for c in [1..list.leng] do
          listc[c].max := solu[c];
          listc[c].set := Filtered(listc[c].set,n ->
            n <= solu[c]);
        od;
        listc[first].max := listc[first].max - 1;
      fi;
    fi;
  od;
listc[first].set := Filtered(listc[first].set, n -> n <= listc[first].max);
listc[second].max := listc[second].max - 1;
listc[second].set := Filtered(listc[second].set, n -> n <= listc[second].max);
if not IsSumPossible(listc, 1/a.r-1/list.nums[first]-1/list.nums[second]) then
    RemoveSet(ret,solu);
fi;
fi;
fi;
od;
fi;
od;
od;
return;
end;
APPENDIX B

Tables

Table 1: The number of possible $(\lambda, d, \rho)$ triples

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Number of Triples</th>
<th>$\lambda$</th>
<th>Number of Triplets</th>
<th>$\lambda$</th>
<th>Number of Triplets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>21</td>
<td>13524</td>
<td>41</td>
<td>125761</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>22</td>
<td>13855</td>
<td>42</td>
<td>136023</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>23</td>
<td>18442</td>
<td>43</td>
<td>146842</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>24</td>
<td>21312</td>
<td>44</td>
<td>158226</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>25</td>
<td>24469</td>
<td>45</td>
<td>170219</td>
</tr>
<tr>
<td>6</td>
<td>89</td>
<td>26</td>
<td>27910</td>
<td>46</td>
<td>182757</td>
</tr>
<tr>
<td>7</td>
<td>195</td>
<td>27</td>
<td>31690</td>
<td>47</td>
<td>195946</td>
</tr>
<tr>
<td>8</td>
<td>355</td>
<td>28</td>
<td>35775</td>
<td>48</td>
<td>209749</td>
</tr>
<tr>
<td>9</td>
<td>589</td>
<td>29</td>
<td>40228</td>
<td>49</td>
<td>224238</td>
</tr>
<tr>
<td>10</td>
<td>900</td>
<td>30</td>
<td>45028</td>
<td>50</td>
<td>239298</td>
</tr>
<tr>
<td>11</td>
<td>1307</td>
<td>31</td>
<td>50217</td>
<td>51</td>
<td>255147</td>
</tr>
<tr>
<td>12</td>
<td>1826</td>
<td>32</td>
<td>55753</td>
<td>52</td>
<td>271623</td>
</tr>
<tr>
<td>13</td>
<td>2462</td>
<td>33</td>
<td>61730</td>
<td>53</td>
<td>288757</td>
</tr>
<tr>
<td>14</td>
<td>3232</td>
<td>34</td>
<td>68117</td>
<td>54</td>
<td>306722</td>
</tr>
<tr>
<td>15</td>
<td>4154</td>
<td>35</td>
<td>74933</td>
<td>55</td>
<td>325389</td>
</tr>
<tr>
<td>16</td>
<td>5230</td>
<td>36</td>
<td>82202</td>
<td>56</td>
<td>344742</td>
</tr>
<tr>
<td>17</td>
<td>6482</td>
<td>37</td>
<td>89940</td>
<td>57</td>
<td>364947</td>
</tr>
<tr>
<td>18</td>
<td>7919</td>
<td>38</td>
<td>98113</td>
<td>58</td>
<td>385851</td>
</tr>
<tr>
<td>19</td>
<td>9574</td>
<td>39</td>
<td>106828</td>
<td>59</td>
<td>407004</td>
</tr>
<tr>
<td>20</td>
<td>11424</td>
<td>40</td>
<td>116038</td>
<td>60</td>
<td>430177</td>
</tr>
</tbody>
</table>
Table 2: The number of \((\lambda, d, \rho)\) triples remaining after the first set of tests

<table>
<thead>
<tr>
<th>(\lambda)</th>
<th>number of triples</th>
<th>(\lambda)</th>
<th>number of triplets</th>
<th>(\lambda)</th>
<th>number of triplets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>21</td>
<td>14</td>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>42</td>
<td>89</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>24</td>
<td>38</td>
<td>44</td>
<td>65</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>25</td>
<td>15</td>
<td>45</td>
<td>93</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>27</td>
<td>24</td>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>28</td>
<td>36</td>
<td>48</td>
<td>124</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>29</td>
<td>0</td>
<td>49</td>
<td>37</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>30</td>
<td>50</td>
<td>50</td>
<td>63</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>31</td>
<td>0</td>
<td>51</td>
<td>61</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>32</td>
<td>32</td>
<td>52</td>
<td>83</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>33</td>
<td>32</td>
<td>53</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>34</td>
<td>0</td>
<td>54</td>
<td>102</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>35</td>
<td>39</td>
<td>55</td>
<td>73</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>36</td>
<td>71</td>
<td>56</td>
<td>128</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>37</td>
<td>0</td>
<td>57</td>
<td>66</td>
</tr>
<tr>
<td>18</td>
<td>16</td>
<td>38</td>
<td>0</td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>39</td>
<td>45</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>40</td>
<td>79</td>
<td>60</td>
<td>219</td>
</tr>
</tbody>
</table>
Table 3: The number of $(\lambda, d, \rho)$ triples remaining after the second set of tests

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Number of Triples</th>
<th>$\lambda$</th>
<th>Number of Triples</th>
<th>$\lambda$</th>
<th>Number of Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>21</td>
<td>4</td>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>42</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>24</td>
<td>7</td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>25</td>
<td>2</td>
<td>45</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>26</td>
<td>0</td>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>27</td>
<td>2</td>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>28</td>
<td>4</td>
<td>48</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>49</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>30</td>
<td>10</td>
<td>50</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>31</td>
<td>0</td>
<td>51</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>32</td>
<td>4</td>
<td>52</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>33</td>
<td>4</td>
<td>53</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>34</td>
<td>0</td>
<td>54</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>35</td>
<td>7</td>
<td>55</td>
<td>14</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>36</td>
<td>13</td>
<td>56</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>37</td>
<td>0</td>
<td>57</td>
<td>6</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>38</td>
<td>0</td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>39</td>
<td>5</td>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>40</td>
<td>11</td>
<td>60</td>
<td>33</td>
</tr>
</tbody>
</table>
BIBLIOGRAPHY


[BK] W. G. Bridges and E. S. Kramer, The determination of all \( \lambda \)-designs with \( \lambda = 3 \), J. of Comb. Th. 8(1970), 343-349.


[Se89] Á. Seress, Some characterizations of type-1 \( \lambda \)-designs, J. of Comb. Th. (A) 52(1989), 288-300.


[Se95] Á. Seress, All \( \lambda \)-designs with \( \lambda = 2p \) are type-1, in preparation.
[SW] Á. Seress and I. Weisz, \textit{$\lambda$-designs with small $\lambda$ are type-1}, in preparation.


