DESIGNS WITH AN INTERSECTION PROPERTY INSPIRED BY THE ERDÖS-KO-RADO PROBLEM.

Mathieu Loiselle

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

December 2015

Abstract

Designs with an Intersection Property Inspired by the Erdös-Ko-Rado Problem.

Mathieu Loiselle

Katona's proof of the Erdős-Ko-Rado theorem (EKR) relies on the existence of a family of v k-subsets of a set of size v with no subfamily of $k+1$ sets that are pairwise 1-intersecting. Herein, we consider a generalization of these families: collections of size $\lambda\binom{v}{t}/\binom{k}{t}$ of k-subsets of a set of size v with no subcollection of $\lambda+1$ sets that are pairwise t-intersecting. Katona's original family corresponds to the case $t=1$ and $\lambda=k$. Replacing Katona's family with such a collection in his proof constitutes a Katona-style proof of EKR for the parameters t, v, and k.

Any Steiner system is such a collection and any such collection is proven to be a t-design. Moreover, the Erdős-Ko-Rado theorem itself can be seen as the statement that for any t and k, and for a sufficiently large value of v, the set of all k-subsets of a set of size v is such a collection with $\lambda=\binom{v-t}{k-t}$. Proofs and analogs of the Erdős-Ko-Rado theorem have been topics of interest since its publication and the t-designs we identify have aspects of both. Each design proves a particular instance of EKR and satisfies a condition analogous to the conclusion of EKR. In honor of Katona's proof, we have named these collections Katona sieves. The research questions addressed by this thesis are for what values of λ, given t, v, and k, does such a collection exist and which t-designs have this property.

We largely restricted our attention to 2-designs and developed programs for generating 2-(v,k,λ) designs, testing for the additional property that no subfamily of $\lambda+1$ subsets is 2-intersecting, that is, testing the condition for being a Katona sieve. For any choice of v, k, and λ, the number of blocks in the design, b, and the number of blocks containing a given element, r, are uniquely determined. Of the 142 case listed in the CRC Handbook for 2-designs with $b \leq 64$ and $r \leq 21$, existence or non-existence of a Katona sieve is established through theory for 92 cases. Of these the enumeration problem was also settled for by theory for 71 cases and settled by computation for 10 cases. Of the 50 cases for which the existence problem is not settled by theory, we resolve 17 more through computation, fully enumerating 14 of these cases. This leaves 33 cases with $b \leq 64$ and $r \leq 21$ for which existence is not determined and an additional 14 only partially enumerated.

Acknowledgments

I would like to extend a sincere thank you to my professor Clement Lam for his patient assistance throughout this process. He is also the person that turned my attention towards enumeration of combinatorial objects. I also wish to thank Vašek Chvátal whose class made me consider the problem in this work and who first pointed me to the relationship of this problem to t-designs. Moreover, Lam and Chvátal's insights were decisive in the final definition of the problem herein.

Finally, Mireille Sauvageau, your support has meant everything to me.

Contents

List of Figures

List of Tables

Chapter 0

Prologue

I would like to take the opportunity to summarize the early evolution of this problem in an informal manner. It began with reading the work of Erdős, Faigle, and Kern in "A Group-Theoretic Settting for Some Interesting Sperner Families" [EFK92]. They perform counting arguments on the action of a group on a monotone increasing sequence of subsets. After a good amount of head scratching, I decided to work through an example, taking as the monotone sequence a given t-subset $T \subseteq V$ and a k-subset K containing T. Working through their Theorem 2.1 with this selection leads to a greatly simplified question with much of the group theory discarded:

> Does there exist a collection, $\mathcal{B}$, of k-subsets of V with $|\mathcal{B}| = \binom{v}{t}$ containing no t-intersecting subcollection of size greater than $\binom{k}{t}$?

This is roughly the question that I asked Vašek Chvátal who responded with the proof that such a collection cannot be larger than $\binom{v}{t}$ and that if the bound is reached, then it must be a t-design. He also attached Östergård's paper on enumerating 2-$(9,4,6)$ designs noting that if such a collection of size $\binom{9}{2} = 36$ exists, then it must be one of the 270 million generated by Östergård [Ös01].

Finally, Clement Lam pointed out that I was limiting myself to designs of size $\binom{v}{t}$ and that the problem makes sense without this constraint. Indeed, considering an arbitrary t-(v,k,λ) design, the final condition that emerges is:

> Does there exist $\lambda > 0$ and a collection, $\mathcal{B}$, of k-subsets of V with $\mathcal{B} = \lambda\frac{\binom{v}{t}}{\binom{k}{t}}$ containing no t-intersecting subcollection of size greater than λ?

In Chapter 3, we will show the rather satisfying result:

> If yes, then $\mathcal{B}$ is a t-(v,k,λ) design that can be used in a Katona-style proof of EKR-t for those parameters.

Chapter 1

Introduction

In this thesis, we study the existence of t-(v, k, λ) designs whose blocks have the additional property that any selection of greater than λ blocks contains a pair of blocks that intersects in less than t varieties. While this is motivated by Katona's proof of the Erdős-Ko-Rado theorem on intersecting families of subsets, the aforementioned property that some t-(v, k, λ) designs possess is interesting in its own right.

1.1 Preliminaries

A t-(v, k, λ) design is a collection, $\mathcal{B}$, of sets such that:

1. Each $B \in \mathcal{B}$ contains k members from a set V of size v.
2. Each t-subset of elements of V is contained in exactly λ elements of $\mathcal{B}$.

When discussing t-designs, the elements of V will also be referred to as *varieties* and elements of $\mathcal{B}$ will also be referred to as *blocks*. Consider the following example with $t = 2$ and $V = \{0, 1, 2, 3\}$:

$$\mathcal{B} = \big\{\{0,1\}, \{0,2\}, \{0,3\}, \{1,2\}, \{1,3\}, \{2,3\}\big\}.$$

This forms a 2-$(4, 2, 1)$ design because each element of $\mathcal{B}$ contains $k = 2$ elements and each pair of elements of V occurs in exactly one member of $\mathcal{B}$.This is a special design because it consists precisely of every pair of elements of V.

Designs are usually depicted in the form of a matrix, known as an incidence matrix. For example, if $\mathcal{B}$ is the design in the preceeding paragraph, each column of the matrix in Figure 1 can be considered to be a member of $\mathcal{B}$ and each row can be considered to be an element of V. Note that the indexing of the matrix is zero-based so the first row corresponds to the element $0 \in V$. A 1 appears at the intersection of a row and column if the element of V corresponding to the row is an element of the member of $\mathcal{B}$ corresponding to the column.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Figure 1: The complete 2-(4,2,1) design

Note how the two conditions to form a 2-(v, k, λ) design translate into properties of this matrix. Each column contains k 1s, since the block it represents must contain k elements of V. Further to this, the dot product of any pair of rows equals λ, which in this case is 1.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure 2: A 2-(7,3,2) design

Figure 2 is an incidence matrix representing a 2-$(7, 3, 2)$ design. This design does not consist simply of all subsets of size 3 of a set of size 7. It also illustrates that the members of a design may be repeated; for instance, the first two columns represent the same subsets of V. There is another property of this design to note. The following lists the pairs of columns that intersect in at least 2 elements of V:

$$\{0, 1\}$$
$$\{2, 3\}$$
$$\{4, 5\}$$
$$\{6, 8\}, \{6, 9\}, \{6, 10\}$$
$$\{7, 8\}, \{7, 9\}, \{7, 11\}$$
$$\{8, 12\}$$
$$\{9, 13\}$$
$$\{10, 12\}, \{10, 13\}$$
$$\{11, 12\}, \{11, 13\}$$

Notice that if two pairs $\{a, b\}$ and $\{b, c\}$ appear in this list, then $\{a, c\}$ does *not* appear in this list.

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}$$

Figure 3: A 2-(11,5,2) design

In other words, there does not exist a triple $\{a, b, c\}$ such that any choice of two is 2-intersecting. Thus, the largest set of columns that is pairwise 2-intersecting is of size 2, the value of λ.

That the largest set of columns such that any pair has is 2-intersecting is of size λ is not necessarily true of all 2-(v, k, λ) designs. Consider the 2-$(11, 5, 2)$ design with incidence matrix illustrated in Figure 3. In this design, *any* choice of two columns has 1s in two common rows. So the largest subset of columns such that any pair contains a common pair of elements of V is the set of all columns!

1.2 Our Question

For any given t-(v, k, λ) design, a selection of λ blocks that pairwise t-intersect can always be made. In fact, taking any t-subset of varieties, the definition of a t-(v, k, λ) design provides such a selection: the λ blocks containing that t-subset. But, when the pairs of blocks in the selection are allowed to intersect in *different* t-subsets of varieties, we may be able to make a larger selection. The question addressed in this thesis is whether, for a given t-(v, k, λ) design, there is *no* larger selection of blocks with each pair of blocks intersecting in at least t varieties. For some parameter sets, the answer is “no” for all t-(v, k, λ) designs. That is, in any t-(v, k, λ) design, a selection of more than λ pairwise t-intersecting blocks always exists. For other parameter sets, it is “yes” for all t-(v, k, λ) designs: in any t-(v, k, λ) design, a selection of more than λ pairwise t-intersecting blocks never exists. Finally, for some parameter sets the answer is more complex: it depends on the design.

In the example illustrated in Figure 1, where $t = 2$, no pair of columns intersects in 2 rows. So, the largest pairwise t-intersecting subsets are the singletons of size 1, the value of λ. So for this design, the answer to our questions is “yes”. But, recall that this example is a special example. It is the complete design of 3-subsets of a set of size 6. The Erdős-Ko-Rado theorem

(EKR) can be seen as the statement that for any t and k, and for a sufficiently large value of v, the answer to our question for the complete design of k-subsets of a set of size v is "yes", there does not exist a selection of greater than λ pairwise t-intersecting blocks. A 1-(v,k,k) design where the answer to our question is "yes" also occurs prominently in Katona's celebrated proof of the Erdős-Ko-Rado theorem when $t = 1$. In this proof he uses this design as a sieve when counting images of members of a family of sets under the action of a permutation group. It is this proof that inspired the search for these t-designs and the name that we have selected for them, Katona sieves.

1.3 Organization of the Thesis.

The body of this thesis is in Chapters 2 to 5. In Chapter 2, we begin by covering terminology and definitions that will be used throughout. After this, we introduce concepts that not all readers may be familiar with. In the final section of Chapter 2, we precisely define the property of t-designs that we seek, as described in Section 1.2. This last section of Chapter 2 covers some results that would be interesting to experienced readers.

In Chapter 3, we cover the Erdős-Ko-Rado theorem and Katona's proof of this theorem for the class of 1-intersecting families of sets. It is to extend this proof that our question becomes pertinent. The existence of a design for which the answer to our question is "yes" proves EKR for a specific case.

Chapter 4 explains the programs that were used. A discussion of their performance is also included. Chapter 5 summarizes the results obtained computationally and theoretically. Finally in Chapter 6 we conclude with a summary of the entire work and potential avenues for future work.

1.4 Contributions of the Thesis.

This thesis presents a novel problem in the study of t-designs related to the historic result of Erdős, Ko, and Rado that spurred on much investigation into intersecting set systems. The property, distilled from Katona's proof of the Erdős, Ko, and Rado theorem, has never been defined previously for t-designs.

We prove that the existence of a solution is equivalent to the conclusion of the Erdős, Ko, and Rado theorem for the parameter set t, v, and k. Non-existence is proven for a very large number of cases. Tit's lower bound on the existence of Steiner systems[Tit64] is shown to extend to these designs. Further, symmetric designs are proven to not have this property if $\lambda \neq 1$. These theoretical results, combined with the results published in the CRC Handbook [CD06], settle the problem for 71 out of 142 cases with $b \leq 64$ and $r \leq 21$. Applying Property 2.6.4 to the known Steiner systems proves existence for 21 additional cases.

We present programs that can serve to introduce the reader to enumeration of combinatorial objects. Isomorphism testing through recorded objects is explained as well as a technique for indirect isomorphism testing by considering the isomorphism groups of smaller sections of completions. The performance comparisons of these programs reveals that such an approach is difficult to adapt to this problem as the condition is a global condition, not conducive to testing on the smaller sections.

An extensive computer search across many cases was performed. The results represent over 200 CPU days of computation. In some cases, after weeks of generation, no solutions were generated. But, in other cases, very many solutions were found. In fact, for some cases we generated more designs than published in the CRC Handbook. One case of particular interest, which started our research, is the case of 2-$(9,4,6)$ designs for which of the 270 million designs, there is a *unique* solution to the problem. The programs resolved existence of a Katona sieve for 17 additional cases with $b \leq 64$ and $r \leq 21$ for which existence is not resolved by theory. The Katona sieves for 24 cases were fully enumerated. For $b \leq 64$ and $r \leq 21$, there are 33 cases for which existence is not determined and an additional 14 that have only been partially enumerated.

Chapter 2

Definitions, Concepts, and Properties

This chapter is meant to provide the reader with all of the notions that are required to understand this thesis. We begin with the definitions and notation that will be used to describe set systems. This is followed by a discussion of permutation groups and their action on set systems and incidence matrices to introduce isomorphism testing. Finally, we provide the definitions of t-designs and Katona sieves.

2.1 Initial Definitions

Our first set of definitions establishes the general terminology and notation we will use to describe and elaborate on properties of set systems. In the entirety of this work, all sets are assumed to be finite.

Definition 2.1.1. Given a set V, 2^V will denote the set of all subsets of V.

Definition 2.1.2. A *permutation* is a bijection from a set to itself. Given a set V, $\mathcal{S}_V$ will denote the set of all permutations of V.

If a set V is, for example, enumerated $V = \{0, 1, 2, 3, \ldots, v-1\}$, we can represent permutations in *cyclic* notation. Cyclic notation denotes a permutation as a sequence of parenthesized disjoint lists of elements of V. Each element within a parenthesized list is mapped by the permutation to the following element in a left to right order except for the last element in the list. This element is mapped to the first element in the subset. Fixed points are usually omitted. For example if $V = \{0, 1, 2, 3, 4, 5\}$ the permutation in cyclic notation,

$$f = (0\ 3\ 1)(4\ 5),$$

denotes the bijection:

$$\begin{aligned}
f(0) &= 3,\\
f(1) &= 4,\\
f(2) &= 2,\\
f(3) &= 1,\\
f(4) &= 5, \text{ and}\\
f(5) &= 4.
\end{aligned}$$

Note how 0, 3, and 4 map to the following element within their respective pairs of parentheses. Being followed by closing parentheses, 1 and 5 map to the first element within their respective pairs of parentheses. And 2, not appearing in the notation, remains fixed. For the identity permutation, all elements are fixed and it is denoted (0), even though 0 is a fixed point.

Definition 2.1.3 (Indicator Function)**.** Given a set V and a subset $U \subseteq V$, the indicator function of U (with domain 2^V) is the mapping $\mathbb{1}_U : 2^V \to \{0,1\}$ defined as:

$$\mathbb{1}_U(W) = \begin{cases} 1 & \text{if } W \subseteq U. \\ 0 & \text{if } W \nsubseteq U. \end{cases}$$

We will be considering two kinds of set systems. *Families* of subsets of a given set and *Collections* of subsets of a given set. The two differ in that collections permit repeated subsets and families do not. Here, we will precisely define these terms. The "given" set will usually be denoted by V and will frequently be referred to as the *base set*.

Definition 2.1.4. Given a set V and an index set I, a *collection* $(\mathcal{B}, I, V)$ on V with index set I is a mapping, $\mathcal{B}$, from I to 2^V.

Definition 2.1.5. Given a set V and an index set I, a *family* $(\mathscr{F}, I, V)$ on V with index set I is an *injective* mapping, $\mathscr{F}$, from I to 2^V.

It is important to note that collections and families are defined as *mappings*. This has consequences on the way in which these objects are discussed. Given a collection $(\mathcal{B}, I, V)$ and an element of the index set, $i \in I$, $\mathcal{B}(i)$ is a *subset* of V. And, $\mathcal{B}(I) = \{\mathcal{B}(i); i \in I\}$ is a *subset* of 2^V. It is the set of subsets of V that occur in the collection, forgotting multiplicity.

Now, a mapping is formally defined as a subset of the direct product of the domain and codomain, we need a term for the pairs that constitute the mapping for a collection $\mathcal{B}$. When referring to the pairs $(i, \mathcal{B}(i)) \in I \times 2^V$ we will use the term *members*. The value of a member is its second coordinate. Given a pair of members $(i, \mathcal{B}(i))$ and $(i', \mathcal{B}(i'))$, their *intersection* will refer to the set $B(i) \cap B(i')$. Note that the intersection of two members of a collection is not necessarily a member of that collection. We will say that two members $(i, \mathcal{B}(i))$ and $(i', \mathcal{B}(i'))$ *repeat* each other if $\mathcal{B}(i) = \mathcal{B}(i')$. More generally, if in prose we perform a set operation on members of a collection, it is to be understood that the operation is on the second coordinates, their values. An example of this that will be seen frequently is summation over collections.

Although the symbol $\mathcal{B}$ by itself denotes only a mapping, in practice, we will use set notation even for collections, and the reader is left to understand that members of the collection should be considered *with multiplicity* if applicable. For instance, we may write $B \in \mathcal{B}$ as in:

$$\sum_{B \in \mathcal{B}} f(B) \text{ instead of } \sum_{i \in I} f(\mathcal{B}(i)).$$

Despite this, we try to include the base set, V, and the symbol used to represent the index set, I, each time a collection is "declared", as in "Let $(\mathcal{B}, I, V)$ be a collection."

Although we could have defined a family more simply as a subset of 2^V, or a set of subsets of V, this isn't the case for collections. For collections, the index set is necessary to maintain distinction between members that may have equal values. The definitions above are consistent with each other; a family is a particular type of collection.

Now we have other definitions related to collections:

Definition 2.1.6. Given a collection $(\mathcal{B}, I, V)$, a *subcollection*, $(\mathcal{B}\restriction_J, J, V)$ of $(\mathcal{B}, I, V)$ is a subset $J \subseteq I$ and a mapping $\mathcal{B}\restriction_J : J \to 2^V$ such that for any $i \in J$, $\mathcal{B}\restriction_J(i) = \mathcal{B}(i)$. A *subfamily* is a subcollection of a family.

Definition 2.1.6 is the natural definition since we defined collections as mappings. If $J \subseteq I$, then $(\mathcal{B}\restriction_J, J, V)$ is a subcollection of $(\mathcal{B}, I, V)$ if and only if $\mathcal{B}\restriction_J$ is the *restriction* of the mapping $\mathcal{B}$ to J in the usual sense of restricting a function to a subdomain of its domain. Note that we have also introducing notation for this restriction, $\mathcal{B}\restriction_J$, but will not always use it. Instead we may simply say declare a collection $(\mathcal{B}', J, V)$ as a subcollection of $(\mathcal{B}, I, V)$.

Definition 2.1.7. If $(\mathcal{B}, I, V)$ is a collection, the *size* of $(\mathcal{B}, I, V)$ is $|I|$. Also, the notation, $|\mathcal{B}|$ is to be understood as $|I|$.

One should contrast this notion of size with the size of the set $\mathcal{B}(I)$. If $\mathcal{B}(I)$ has repeated members, then $|\mathcal{B}(I)| < |I| = |\mathcal{B}|$. A collection is a family if and only if $|\mathcal{B}(I)| = |I| = |\mathcal{B}|$.

Definition 2.1.8. Given a set V, $\binom{V}{k}$ will denote the set of all k-subsets of V. That is, the set of subsets of V containing exactly k elements.

Later, we will be restricting our attention to *k-uniform* collections. A collection is k-uniform if the value of each of its members is a k-subset of V.

Definition 2.1.9. A collection $(\mathcal{B}, I, V)$ is *k-uniform* if $\mathcal{B}(I) \subseteq \binom{V}{k}$.

And, we will be discussing intersections between members of a collection a great deal.

Definition 2.1.10. A pair of subsets $U \subseteq V$ and $W \subseteq V$ is *t-intersecting* if $|U \cap W| \geq t$. A collection $(\mathcal{B}, I, V)$ is *t-intersecting* if for any pair $i, i' \in I$, $|\mathcal{B}(i) \cap \mathcal{B}(i')| \geq t$.

So a collection is t-intersecting if and only if the size of the intersection of any pair of its members is at least t. In set notation, $(\mathcal{B}, I, V)$ is *t-intersecting* if for any pair $B, B' \in \mathcal{B}$, $|B \cap B'| \geq t$.

At times it will be convenient to have a notation for a generic set of a given size;

Definition 2.1.11. If $v \in \mathbb{N}$, then $\bar{v} = \{0, 1, \ldots, v-1\}$.

We will use interval notation for intervals of *integers*,

Definition 2.1.12. If $a, b \in \mathbb{N}$, then define $[a, b] = \{n \in \mathbb{N}; a \leq n \leq b\}$.

Definition 2.1.13. If M is a $v \times b$ matrix and $0 \leq i < v$ and $0 \leq j < b$, M^i represents the i^{th} row of the matrix and M_j will represent the j^{th} column, and $M_{(i,j)}$ the entry in row i and column j.

To simplify the algorithms, the rows and columns of matrices will be indexed starting with 0. However, in prose we may still refer to the 0^{th} row as the first row and 0^{th} column as the first column. We will use calligraphy script to represent the set of all $v \times b$ matrices for integers v and b.

Definition 2.1.14. Let $\mathcal{M}_{v,b}(L)$ represent the set of matrices with v rows, b columns, and entries from the set L.

If the set L is omitted, it should be understood from the context. The reason that we include a set of possible entries even if we will really only be concerned with (0,1)-matrices where $L = \{0, 1\}$ is that we will also need to be able to discuss partially completed matrices. We will represent the undetermined entries with a "?", and, as such, we will sometimes need $L = \{0, 1, ?\}$.

We will be discussing canonical forms and will use the row-major maximum matrix as an example of a canonical form. The example depends on an ordering of $\mathcal{M}_{v,b}(L)$. We begin by explaining this ordering. Consider the small matrix:

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}.$$

We can consider this a binary integer by concatenating the rows together one after the other as in:

$$101101001100.$$

This is the row-major binary value of the matrix. However, if some entries of the matrix were undetermined, we couldn't do so directly:

$$\begin{bmatrix} ? & 0 & 1 & 1 \\ 0 & ? & 0 & 0 \\ 1 & 1 & 0 & ? \end{bmatrix}.$$

Instead we need to assign a value to "?". Suppose that we assign it the value 2 and glue then concatenate the rows together:

$$201102001102.$$

This does represent an integer base 3. So with this function,

$$f(\ell) = \begin{cases} 0 & \text{If } \ell = 0, \\ 1 & \text{If } \ell = 1, \text{ and} \\ 2 & \text{If } \ell = ?; \end{cases} \tag{1}$$

we were able to assign this matrix an integer value.

Definition 2.1.15. Given an injection $f : L \to \overline{|L|}$ the *row-major value* of $M \in \mathcal{M}_{v,b}(L)$ with respect to f is defined as:

$$\|M\|_f = \sum_{i=0}^{v-1} \sum_{j=0}^{b-1} f(M_{i,j}) |L|^{((v-1)-i)b+((b-1)-j)}.$$

The *row-major ordering* with respect to f is defined for $M_1, M_2 \in \mathcal{M}_{v,b}(L)$ as:

$$M_1 \leq M_2 \iff \|M_1\|_f \leq \|M_2\|_f.$$

This defines the row-major value of $M \in \mathcal{M}_{v,b}(L)$ with respect to f as the integer base $|L|$ formed from concatenating the values of the entries of each row together one row after another in order, so that the first row becomes the highest order "digits" in the integer. The row-major ordering of matrices is a total ordering since each matrix maps to a different integer. Column-major value and ordering is defined equivalently, and we use the same notation. But, it will always be clear from the context to which we are referring.

Definition 2.1.16. Given an injection $f : L \to \overline{|L|}$ the *column-major value* of $M \in \mathcal{M}_{v,b}(L)$ with respect to f is defined as:

$$\|M\|_f = \sum_{j=0}^{b-1} \sum_{i=0}^{v-1} f(M_{i,j}) |L|^{((b-1)-j)v+((v-1)-i)}.$$

The *column-major ordering* with respect to f is defined for $M_1, M_2 \in \mathcal{M}_{v,b}(L)$ as:

$$M_1 \leq M_2 \iff \|M_1\|_f \leq \|M_2\|_f.$$

Now, the value of a matrix is the integer formed by concatenating the columns together one after another. When dealing with partially completed $(0,1)$-matrices, $\|M\|$, with no index will denote the row major order with f as defined in Equation 1.

2.2 Incidence Matrix Representation

The most common representation of collections is the incidence matrix. This representation is also the basic representation when studying collections with computers. In an incidence matrix, relationships between elements of the base set and members of the collection are represented as a $(0,1)$-matrix. The rows of the matrix represent the elements of the base set and the columns represent the elements of the index set. In an incidence matrix of a collection $(\mathcal{B}, I, V)$, for each $i \in I$ and $v \in V$, a 1 appears in the intersection of the row representing v and the column representing i if $v \in \mathcal{B}(i)$ and a 0 if $v \notin \mathcal{B}(i)$.

In order to create an incidence matrix, the base set and index set must be enumerated. For example, consider the following collection $(\mathcal{B}, I, V)$, where $I = \{i_1, i_2, i_3, i_4\}$, $V = \{v_1, v_2, v_3, v_4, v_5\}$, and

$$\begin{aligned}
\mathcal{B}(i_1) &= \{v_1, v_3, v_5\}, \\
\mathcal{B}(i_2) &= \{v_2, v_3\}, \\
\mathcal{B}(i_3) &= \{v_1, v_4, v_5\}, \text{ and} \\
\mathcal{B}(i_4) &= \{v_1, v_4, v_5\}.
\end{aligned}$$

Then, an incidence matrix representing this collection is:

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}
\begin{array}{c} \begin{array}{cccc} i_1 & i_2 & i_3 & i_4 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{array}\right] \end{array}.$$

Note that in this description of incidence matrices we said that this is "an" incidence matrix representation. The astute reader will note that in selecting an enumeration of the base set V and the index set I, we have introduced a choice that is not intrinsic to the collection. There are $|V|!$ ways to number the elements of V and $|I|!$ ways to number the elements of I. A computer cannot generate collections directly, as these are abstract objects. Rather, incidence matrices are generated. But, this means that two different incidence matrices may be generated that are in fact representations of the same collection with different choices of indexing.

For Property 2.5.2 in Section 2.5 and Property 2.6.4 in Section 2.6, we will be showing that concatenating collections with certain special properties yields collections with these same special properties. The concatenation of two collections is the collection that can be represented by the incidence matrix formed by putting incidence matrices of each of the two side by side. This is possible as long as the two collections have the same base set. For example, concatenating $\mathcal{B}$ with itself yields a collection that can be represented by the incidence matrix:

$$\begin{array}{c|cccc:cccc|} & i_1 & i_2 & i_3 & i_4 & i'_1 & i'_2 & i'_3 & i'_4 \\ v_1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ v_2 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ v_3 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ v_4 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ v_5 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{array}.$$

2.3 Permutations Acting on Incidence Matrices

In Chapter 4, we will be performing isomorph rejection on incidence matrices to generate only nonisomorphic matrices. In this section, we explain how this is equivalent to the generation of nonisomorphic collections. As explained previously, to represent a collection by an incidence matrix, the base set and the index set of a collection need to be enumerated. This implies that a given collection can be represented by more than one incidence matrix. To remedy this, we first need to define the notion of equivalence of collections. Then, we need to define the notion of equivalence of incidence matrices. And finally, we need to analyze how the two notions correspond.

It will be necessary to deal with groups and their actions on objects. So, we remind the reader of the definition of a group and the action of a group. Many resources exist for a more general and in depth introduction to group theory such as Micheal Artin's *Algebra* [Art91].

Definition 2.3.1. A group is a set G endowed with a a composition, $\cdot : G \times G \to G$ taking (g_1, g_2) to $g_1g_2 \in G$, satisfying the following properties:

1. For any $g_1, g_2, g_3 \in G$ composition is *associative*, $g_1(g_2g_3) = (g_1g_2)g_3$.
2. There exists an *identity* $e \in G$ such that for any $g \in G$, $eg = ge = g$.
3. For any $g \in G$ there exists an *inverse* of g, $g^{-1} \in G$, such that $gg^{-1} = g^{-1}g = e$.

A related definition is that of a subgroup of a group:

Definition 2.3.2. A subgroup H of a group G is a subset $H \subseteq G$ such that H is itself a group under the same composition as G, or, equivalently, for any $h_1, h_2 \in H$, $h_1h_2 \in H$ and $h_1^{-1} \in H$. That H is a subgroup of G is denoted $H \leq G$.

Now, the definition of an action:

Definition 2.3.3. An *action* of a group G on a set S is a mapping $G \times S \to S$ taking (g, s) to $gs \in S$ satisfying the following conditions:

1. For any $g_1, g_2 \in G$ and $s \in S$, $(g_1 g_2)s = g_1(g_2 s)$.

2. If e is the identity of g, for any $s \in S$, $es = s$.

And, the definition of an associated concept, the orbit of an element, is:

Definition 2.3.4. Given a group G acting on a set S, the *orbit* of an element $s \in S$, is the set of all images of s through the action of G, that is $\{gs : g \in G\}$. It may be denoted as GS or $\mathrm{Orbit}_G(S)$.

The set of all permutations of a set V, $\mathcal{S}_V$ as in Definition 2.1.2 forms a group frequently called the *symmetric group* of V. We will principally be dealing with symmetric groups, the group of isomorphisms of a collection to itself, and the group of isomorphisms of a matrix to itself. We have not yet formally defined the concept of an isomorphism of one collection to another. We do so now.

Definition 2.3.5. A pair of bijective mappings,

$$\begin{aligned} &f : V \to V', \text{ and} \\ &g : I \to I', \end{aligned}$$

is an *isomorphism* from a collection $(\mathcal{B}, I, V)$ to another collection $(\mathcal{B}', I', V')$ if they satisfy

$$v \in \mathcal{B}(i) \iff f(v) \in \mathcal{B}'(g(i)).$$

The collections $(\mathcal{B}, I, V)$ and $(\mathcal{B}', I', V')$ are *isomorphic* if there exists such a pair (f, g). That two collections are isomorphic will be denoted:

$$(\mathcal{B}, I, V) \cong (\mathcal{B}', I', V').$$

Definition 2.3.6. An isomorphism from a collection to itself is an *automorphism* of that collection. The automorphisms of a collection form a group called the *automorphism group* of that collection.

Note that we are saying that the isomorphisms from a collection to itself forms a group. We need to show that the identity is an automorphism, that the composition of automorphisms yields automorphisms, and that inverting automorphisms yields automorphisms. It's clear that if f and g are the identity mapping of V and I respectively,

$$v \in \mathcal{B}(i) \iff f(v) \in \mathcal{B}'(g(i)).$$

Now, suppose that the pairs (f, g) and (f', g') are automorphisms of $(\mathcal{B}, I, V)$, then:

$$v \in \mathcal{B}(i) \iff f(v) \in \mathcal{B}(g(i)) \iff f'(f(v)) \in \mathcal{B}(g'(g(i))).$$

So, $(f' \circ f, g' \circ g)$ is also an automorphism of $(\mathcal{B}, I, V)$.

Finally, we need to show that if (f, g) is an automorphism so is (f^{-1}, g^{-1}). Since f is injective, for any $v \in V$, $v = f^{-1}(u)$ for some $u \in V$. Similarly, $i = g^{-1}(j)$ for some $j \in I$. For any such u and j,

$$v \in \mathcal{B}(i) \iff f(v) \in \mathcal{B}(g(i))$$

is equivalent to:

$$f^{-1}(u) \in \mathcal{B}(g^{-1}(j)) \iff f(f^{-1}(u)) \in \mathcal{B}(g(g^{-1}(j)) \iff u \in \mathcal{B}(j).$$

But, since f and g are surjective, for *any* $u \in V$ and $j \in J$ there exists corresponding $v = f^{-1}(u)$ in V and $i = g^{-1}(j)$ in I.

We now turn our attention to matrices and their isomorphisms, since they will be used to represent collections. Afterwards, we will establish how isomorphisms of collections are equivalent to isomorphisms of the matrices that are used to represent them.

Definition 2.3.7. Given a pair of permutations,

$$\begin{aligned} &\rho \in \mathcal{S}_v, \text{ and} \\ &\sigma \in \mathcal{S}_b, \end{aligned}$$

and a matrix M, let $\rho M \sigma$ be the matrix such that $(\rho M \sigma)_{(i,j)} = M_{(\rho^{-1} i, \sigma^{-1} j)}$.

Definition 2.3.8. A pair of permutations,

$$\begin{aligned} &\rho \in \mathcal{S}_v, \text{ and} \\ &\sigma \in \mathcal{S}_b, \end{aligned}$$

is an *isomorphism* from a $v \times b$ matrix M to a $v \times b$ matrix M' if

$$\rho M \sigma = M'.$$

The matrices M and M' are *isomorphic* if there exists such a pair $(\rho, \sigma) \in \mathcal{S}_v \times \mathcal{S}_b$. That two matrices are isomorphic will be denoted:

$$M \cong M'.$$

At times we may need to restrict the group acting on matrices from $\mathcal{S}_v \times \mathcal{S}_b$ to one of its subgroups. As such we define:

Definition 2.3.9. The matrices M and M' are *isomorphic with respect to a permutation group* $G \leq \mathcal{S}_v \times \mathcal{S}_b$ if there exists a pair $(\rho, \sigma) \in G$ such that $\rho M \sigma = M'$.

Finally, given a matrix, we have a group:

Definition 2.3.10. An isomorphism from a matrix to itself is an *automorphism* of that matrix. The automorphisms of a matrix form a group called the *automorphism group* of that matrix.

To see that this is a group, suppose that (ρ, σ) is an automorphism of a matrix. Then for any i and j,

$$(\rho M \sigma)_{(i,j)} = M_{(\rho^{-1} i, \sigma^{-1} j)} = M_{(i,j)}.$$

That is, applying the permutation to M keeps all of the values of the matrix the same. Applying the inverse permutation will also keep all the values the same:

$$(\rho^{-1} M \sigma^{-1})_{(i,j)} = M_{(\rho i, \sigma j)} = M_{(\rho^{-1}(\rho i), \sigma^{-1}(\sigma j))} = M_{((\rho^{-1}\rho) i, (\sigma^{-1}\sigma) j)} = M_{(i,j)}.$$

Composing another automorphism (ρ', σ') with (ρ, σ) also yields an automorphsim of M:

$$((\rho'\rho) M (\sigma'\sigma))_{(i,j)} = M_{((\rho'\rho)^{-1} i, (\sigma'\sigma)^{-1} j)} = M_{(\rho^{-1}(\rho'^{-1} i), \sigma^{-1}(\sigma'^{-1} j))} = M_{(\rho'^{-1} i, \sigma'^{-1} j)} = M_{(i,j)}.$$

Note that we chose to write the row permutations on the left and the column permutations on the right. This makes it clear that the row and column permutation can be applied in either order. Now, we need a precise definition of an incidence matrix representation if we wish to show how isomorphisms of incidence matrices are equivalent to isomorphisms of the collections that they represent.

Definition 2.3.11. Given a collection $(\mathcal{B}, I, V)$, an incidence matrix representation is a pair of bijections $\alpha_M : \{0, \dots, |V| - 1\} \to V$ and $\beta_M : \{0, \dots, |I| - 1\} \to I$ together with a matrix, M, such that:

$$M_{(a,b)} = \begin{cases} 0 & \text{if } \alpha_M(a) \notin \mathcal{B}(\beta_M(b)), \text{ and} \\ 1 & \text{if } \alpha_M(a) \in \mathcal{B}(\beta_M(b)). \end{cases}$$

So an incidence matrix representation is not only the matrix itself, but also the mapping of the rows to the varieties V and the mapping of the columns to the index set I. The next property says that if M and M' are the matrices of two incidence matrix representations, then any isomorphism from M to M' corresponds to an isomorphism of the collections that they represent, and any isomorphsim of these collections corresponds to an isomorphism of these matrices.

Property 2.3.1. If (α_M, β_M, M) is an incidence matrix representation of a collection $(\mathcal{B}, I, V)$ and $(\alpha_{M'}, \beta_{M'}, M')$ is an incidence matrix representation of $(\mathcal{B}', I', V')$, then:

$$(\rho, \sigma) \in \mathcal{S}_v \times \mathcal{S}_b \text{ is an isomorphism from } M \text{ to } M' \iff$$
$$(\alpha_{M'} \rho \alpha_M^{-1}, \beta_{M'} \sigma \beta_M^{-1}) \text{ is an isomorphism from } (\mathcal{B}, I, V) \text{ to } (\mathcal{B}', I', V').$$

And,

$$(f,g) \text{ is an isomorphism from } (\mathcal{B},I,V) \text{ to } (\mathcal{B}',I',V') \iff$$
$$(\alpha_{M'}^{-1} f \alpha_M, \beta_{M'}^{-1} g \beta_M) \text{ is an isomorphism from } M \text{ to } M'.$$

Proof. Suppose that $(\rho,\sigma) \in \mathcal{S}_v \times \mathcal{S}_b$, and

$$M = \rho M' \sigma.$$

Then,

$$\begin{aligned} v \in \mathcal{B}(i) &\iff M_{\alpha_M^{-1}(v),\beta_M^{-1}(i)} = 1 \\ &\iff M'_{\rho\alpha_M^{-1}(v),\sigma\beta_M^{-1}(i)} = 1 \\ &\iff \alpha_{M'}\rho\alpha_M^{-1}(v) \in \mathcal{B}'(\beta_{M'}\sigma\beta_M^{-1}(i)). \end{aligned}$$

So, $\alpha_{M'}\rho\alpha_M^{-1}$ and $\beta_{M'}\sigma\beta_M^{-1}$ form an isomorphism from $(\mathcal{B},I,V)$ to $(\mathcal{B}',I',V')$. Now suppose that there are bijections $f : V \to V'$ and $g : I \to I'$ such that

$$v \in \mathcal{B}(i) \iff f(v) \in \mathcal{B}'(g(i)).$$

Then,

$$\begin{aligned} M_{(a,b)} = 1 &\iff \alpha_M(a) \in \mathcal{B}(\beta_M(b)) \\ &\iff f\alpha_M(a) \in \mathcal{B}'(g\beta_M(b)) \\ &\iff M'_{(\alpha_{M'}^{-1} f\alpha_M(a),\beta_{M'}^{-1} g\beta_M(b))} = 1. \end{aligned}$$

Since all the involved mappings are bijective, $(\alpha_{M'}^{-1} f \alpha_M, \beta_{M'}^{-1} g \beta_M) \in \mathcal{S}_v \times \mathcal{S}_b$ and is an isomorphism from M to M'. □

This fully states the equivalence, but it is a consequence of this, a simpler property, that is more important to us:

Property 2.3.2. If (ρ,σ) is an isomorphism from M to M', then (α_M, β_M, M) is an incidence matrix representation of $(\mathcal{B},I,V)$ if and only if $(\alpha_M\rho^{-1}, \beta_M\sigma^{-1}, M')$ is an incidence matrix representation of $(\mathcal{B},I,V)$.

Proof. If (α_M, β_M, M) is an incidence matrix representation of $(\mathcal{B},I,V)$, then

$$\begin{aligned} M'_{a,b} = 1 &\iff M_{\rho^{-1}(a),\sigma^{-1}(b)} = 1 \\ &\iff \alpha_M\rho^{-1}(a) \in \mathcal{B}(\beta_M\sigma^{-1}(b)). \end{aligned}$$

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Figure 4: A 2-(6,3,2) design

On the other hand (ρ^{-1}, σ^{-1}) is an isomorphism from M' to M. So, if $(\alpha_M \rho^{-1}, \beta_M \sigma^{-1}, M')$ is an incidence matrix representation of $(\mathcal{B}, I, V)$, we have $(\alpha_M \rho^{-1}(\rho^{-1})^{-1}, \beta_M \sigma^{-1}(\sigma^{-1})^{-1}, M) = (\alpha_M \rho^{-1}\rho, \beta_M \sigma^{-1}\sigma, M) = (\alpha_M, \beta_M)$ is an incidence matrix representation of $(\mathcal{B}, I, V)$. $\square$

Property 2.3.2 says that if two matrices M and M' are isomorphic, then any collection that can be represented by one can also be represented by the other. Now we will present an example of an isomorphism of incidence matrix representations in detail, but because of the equivalence through these two properties, we will later only be focusing on isomorphisms of the incidence matrices themselves. That is, we will not specify the mappings α and β.

Suppose that M in Figure 4 represents the collection $(\mathcal{B}, I, V)$ with the mappings (α, β) : $\{0, \ldots, |V|-1\} \times \{0, \ldots, |I|-1\} \to V \times I$. Consider the entry of the matrix:

$$M_{(2,1)} = 0.$$

$\alpha(2)$ is an element of V and $\beta(1)$ is an element of I. And, since $M_{(2,1)} = 0$,

$$\alpha(2) \notin \mathcal{B}(\beta(1)).$$

Similarly, since

$$M_{(3,1)} = 1,$$

we conclude,

$$\alpha(3) \in \mathcal{B}(\beta(1)).$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Figure 5: $M\sigma$ (on the right), a column permutation of M

Now, suppose that we interchange the first and second columns of M as in Figure 5. Let $\sigma = (0\ 1)$ represent this column permutation.
Notice that,

$$(M\sigma)_{(2,1)} = 1.$$

But, recall that,

$$\alpha(2) \notin \mathcal{B}(\beta(1)).$$

However, $M\sigma$ can also be made to represent the collection $(\mathcal{B}, I, V)$ with an appropriate choice of indexing, (α', β'). In particular, if

$$\begin{aligned} \alpha' &= \alpha \qquad \text{and} \\ \beta' &= \beta\sigma^{-1} \end{aligned}$$

as suggested by Property 2.3.2. Then, with this choice of indexing,

$$(M\sigma)_{(2,1)} = 1$$

is correct, since

$$\alpha'(2) = \alpha(2) \in \mathcal{B}(\beta(0)) = \mathcal{B}(\beta\sigma^{-1}(1)) = \mathcal{B}(\beta'(1)).$$

Similarly, if we apply a row permutation ρ to $M\sigma$, we have another incidence matrix representation of $(\mathcal{B}, I, V)$:

$$\begin{bmatrix} 1&1&1&1&1&0&0&0&0&0 \\ 1&1&0&0&0&1&1&1&0&0 \\ 0&1&1&0&0&1&0&0&1&1 \\ 1&0&0&1&0&0&1&0&1&1 \\ 0&0&1&0&1&0&1&1&1&0 \\ 0&0&0&1&1&1&0&1&0&1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1&1&1&1&1&0&0&0&0&0 \\ 1&1&0&0&0&1&1&1&0&0 \\ 1&0&0&1&0&0&1&0&1&1 \\ 0&1&1&0&0&1&0&0&1&1 \\ 0&0&1&0&1&0&1&1&1&0 \\ 0&0&0&1&1&1&0&1&0&1 \end{bmatrix}.$$

If $\rho = (2\ 3)$ represents this row permutation, then the appropriate choice of α'' and β'' to make $\rho M\sigma$ an incidence matrix representation of $(\mathcal{B}, I, V)$ is:

$$\begin{aligned} \alpha'' &= \alpha\rho^{-1} \text{ and} \\ \beta'' &= \beta\sigma^{-1}. \end{aligned}$$

2.4 Isomorph Rejection

In the previous section we explained that an incidence matrix representation of a collection $(\mathcal{B}, I, V)$ includes a choice of numbering of I and a choice of numbering of V. Given a matrix representing a collection, any permutation of columns is a renumbering of the elements of I and any permutation of rows is a renumbering of V.

Recall the example in Section 2.3 of three matrices that all can be made to represent the same collection:

$$\begin{bmatrix} 1&1&1&1&1&0&0&0&0&0\\ 1&1&0&0&0&1&1&1&0&0\\ 1&0&1&0&0&1&0&0&1&1\\ 0&1&0&1&0&0&1&0&1&1\\ 0&0&1&0&1&0&1&1&1&0\\ 0&0&0&1&1&1&0&1&0&1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1&1&1&1&1&0&0&0&0&0\\ 1&1&0&0&0&1&1&1&0&0\\ 0&1&1&0&0&1&0&0&1&1\\ 1&0&0&1&0&0&1&0&1&1\\ 0&0&1&0&1&0&1&1&1&0\\ 0&0&0&1&1&1&0&1&0&1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1&1&1&1&1&0&0&0&0&0\\ 1&1&0&0&0&1&1&1&0&0\\ 1&0&0&1&0&0&1&0&1&1\\ 0&1&1&0&0&1&0&0&1&1\\ 0&0&1&0&1&0&1&1&1&0\\ 0&0&0&1&1&1&0&1&0&1 \end{bmatrix}.$$

If we wish to use $(0, 1)$-matrices to represent all collections with given parameters, it would not be correct to list these three matrices as different collections; the difference in the matrices is an artifact of the choices of numbering. The typical way to overcome this is to define a canonical form for incidence matrices.

Definition 2.4.1. A *canonical form with respect to a permutation group* $G \leq \mathcal{S}_v \times \mathcal{S}_b$ for $\mathcal{M}_{v,b}(L)$ is a mapping $\Psi : \mathcal{M}_{v,b}(L) \to \mathcal{M}_{v,b}(L)$ such that for any $A, B \in \mathcal{M}_{v,b}(L)$

$$\Psi(A) = \Psi(B) \text{ if and only if there exists } (\sigma_r, \sigma_c) \in G \text{ such that } \sigma_r A \sigma_c = B.$$

This definition states that a canonical form for a matrix is the same for any matrix that can be obtained from it through the action of G, that is, any matrix in the same orbit. If the permutation group is not specified, the *canonical form* is assumed to be with respect to $\mathcal{S}_v \times \mathcal{S}_b$.

When the permutation group in question is $\mathcal{S}_v \times \mathcal{S}_b$, recall how Property 2.3.2 states that if (M, α, β) is an incidence matrix representation of the collection $(\mathcal{B}, I, V)$, then for any element M' in the orbit of M, there exist mappings α' and β' such that (M', α', β') is an incidence matrix representation of $(\mathcal{B}, I, V)$. Therefore, the canonical form of a matrix can represent any collection that any matrix in its orbit can represent. And, by its definition, is the canonical form of any matrix in its orbit.

Given any total ordering of $\mathcal{M}_{v,b}(L)$ and permutation group G we can define a canonical form with respect to G:

$$\Psi(A) = \rho' A \sigma' \text{ such that for any } (\rho, \sigma) \in G,\ \rho A \sigma \leq \rho' A \sigma'.$$

In other words, $\Psi(A)$ is the matrix in the orbit of A under the action of G with the greatest value in the ordering. For $\mathcal{M}_{v,b}(L)$, matrices of size $v \times b$ with entries from L, given a value $f : L \to \overline{|L|}$, the row-major or column major ordering with respect to f is a total ordering and can thus be used to define a canonical form. But, other more efficiently calculated canonical forms exist. The canonical forms that were used in the implementations of the algorithms are those returned by Brendan Mckay's *nauty* [MP14].

2.5 t-designs

We will now begin considering collections with more structure. The algorithms that will be presented in later sections are for the generation of 2-designs, a t-design with $t = 2$. Chapter 3 relates t-designs with our additional property to EKR. As such we need to introduce t-designs for readers that may not be familiar with them.

Definition 2.5.1. For $0 \leq t \leq k \leq v$, a collection $(\mathcal{B}, I, V)$ is a t-(v,k,λ) *design* if:

1. $|V| = v$,
2. $\mathcal{B}$ is k-uniform, and
3. For any t-subset $T \subseteq V$, $|\{i \in I; T \subseteq \mathcal{B}(i)\}| = \lambda$.

The term *t-design* is used to refer to a t-(v,k,λ) design for some parameters v, k, and λ. The members of a t-(v,k,λ) design are commonly referred to as *blocks* and the elements of the base set are commonly referred to as *varieties.* In the literature, t-(v,k,λ) designs are characterized with five parameters (along with t). But, given t, any three of the five parameters determine the other two.

$$\begin{aligned}
v &= \text{The number of varieties.} \\
b &= \text{The number of blocks.} \\
k &= \text{The number of varieties per block.} \\
r &= \text{The number of blocks containing each variety.} \\
\lambda &= \text{The number of blocks containing any given } t\text{-subset of varieties.}
\end{aligned}$$

Establishing that b is a function of t,v, k, and λ can be done through a simple argument. Since it must be k-uniform, each block of a t-design includes $\binom{k}{t}$ t-subsets. Each t-subset of V must occur in exactly λ blocks. Therefore, $b\binom{k}{t} = \lambda\binom{v}{t}$, or, $b = \lambda\frac{\binom{v}{t}}{\binom{k}{t}}$.

Property 2.5.1. If $(\mathcal{B}, I, V)$ is a t-(v,k,λ) design, then $|\mathcal{B}| = \lambda\frac{\binom{v}{t}}{\binom{k}{t}}$.

That the parameter r is a function of t,v, k, and λ follows from observing that b blocks each with k elements must be the same number as v elements each in r blocks, i.e. $bk = vr$, or $r = \frac{bk}{v}$.

Note that t-designs can be concatenated to yield t-designs.

Property 2.5.2. Suppose that $(\mathcal{B}, I, V)$ is a t-(v,k,λ) designs and $(\mathcal{B}', I', V)$ is a t-(v,k,λ') designs with $I \cap I' = \emptyset$. Then, the collection:

$$\mathcal{B}'' : I \cup I' \to 2^V : i \mapsto \begin{cases} \mathcal{B}(i) & \text{if } i \in I, \text{ and} \\ \mathcal{B}'(i) & \text{if } i \in I', \end{cases}$$

is a t-$(v, k, \lambda + \lambda')$ designs.

Proof. V hasn't changed and $\mathcal{B}''$ is clearly k-uniform. Given a t-subset T of V, there are λ indices $i \in I$ for which $T \subseteq \mathcal{B}(i)$ and λ' indices $i \in I'$ for which $T \subseteq \mathcal{B}(i')$. Therefore there are $\lambda + \lambda'$ indices $i \in I \cup I'$ such that $T \subseteq \mathcal{B}''(i)$. □

Our first examples of t-designs are Steiner systems. These are the class of t-designs where $\lambda = 1$.

Definition 2.5.2. An $\mathcal{S}(t, k, v)$-*Steiner System* is a t-$(v, k, 1)$ design for some parameters $t \leq k \leq v$.

Steiner systems are interesting to us because we will see that they all satisfy the additional condition that we will impose on t-designs. In this sense, the designs that we isolate can also be seen as generalizations of Steiner systems. Steiner systems are well studied. One can find examples in Chapter 4 of the CRC Handbook [CD06].

2.6 (λ, t)-Disjointness

As there does not appear to already be a term for the property we need to discuss, we are naming it here. This property is the focus of this work. At the end of this section, it will become clear why t and λ were used as the "parameters" of this property.

Definition 2.6.1. For $\lambda, t \in \mathbb{N}$, a collection $(\mathcal{B}, I, V)$ is (λ, t)-*disjoint* if all t-intersecting sub-collections are of size no more than λ.

Our first example of (λ, t)-disjoint collections are Steiner Systems. That they are t-designs is not coincidental. In fact, Katona pointed out that Steiner systems can be used in the same way that we will use (λ, t)-disjoint t-designs in Chapter 3.

Property 2.6.1. An $\mathcal{S}(t, k, v)$-Steiner system is $(1, t)$-disjoint.

Proof. Suppose that a pair of members in an $\mathcal{S}(t, k, v)$-Steiner system is t-intersecting. Then, then there is a t-subset contained in the intersection of these. But this t-subset is thus contained in more than one block, contradicting the definition of a Steiner system. □

The argument of this example can be extended to show an upper bound on the size of k-uniform (λ, t)-disjoint collections. The following result was provided by Vašek Chvátal in a private communication [Chv12].

Property 2.6.2. For any (λ, t)-disjoint k-uniform collection $(\mathcal{B}, I, V)$,

$$|\mathcal{B}| \leq \lambda \frac{\binom{v}{t}}{\binom{k}{t}}.$$

Proof. We count the number of instances where a t-subset of V is a subset of a member of the collection $\mathcal{B}$. This is given by the double summation:

$$\sum_{B\in\mathcal{B}}\sum_{T\in\binom{V}{t}} \mathbb{1}_B(T).$$

Since $\mathcal{B}$ is k-uniform, each member includes $\binom{k}{t}$ t-subsets as a subset:

$$\begin{aligned}\sum_{B\in\mathcal{B}}\sum_{T\in\binom{V}{t}} \mathbb{1}_B(T) &= \sum_{B\in\mathcal{B}}\binom{k}{t}\\ &= |\mathcal{B}|\binom{k}{t}.\end{aligned}$$

If we reverse the order of summation we see the implication of (λ, t)-disjointness. Since a given t-subset can be a subset of no more than λ members of $\mathcal{B}$,

$$\begin{aligned}\sum_{T\in\binom{V}{t}}\sum_{B\in\mathcal{B}} \mathbb{1}_B(T) &\le \sum_{T\in\binom{V}{t}}\lambda\\ &= \lambda\binom{v}{t}.\end{aligned}$$

Therefore,

$$|\mathcal{B}|\binom{k}{t} = \sum_{B\in\mathcal{B}}\sum_{T\in\binom{V}{t}} \mathbb{1}_B(T) = \sum_{T\in\binom{V}{t}}\sum_{B\in\mathcal{B}} \mathbb{1}_B(T) \le \lambda\binom{v}{t}.$$

□

Property 2.6.2 says that an upper bound on the size of a k-uniform (λ, t)-disjoint collection is the size of a t-(v, k, λ) design. Steiner systems were our first example of (λ, t)-disjoint collections, and, being t-designs, they attain this bound by Property 2.5.1 . One might wonder if k-uniform (λ, t)-disjoint collections that attain this maximum size are *always* t-(v, k, λ) designs. We introduce the term *Katona Sieve* to encompass these properties:

Definition 2.6.2. A (λ, t)-disjoint collection $(\mathcal{B}, I, V)$ is defined to be a *Katona Sieve* for parameters (t, v, k, λ), where $t \le k \le v$, if $|V| = v$, it is k-uniform, and

$$|\mathcal{B}| = \lambda\frac{\binom{v}{t}}{\binom{k}{t}}.$$

To our knowledge no name exists for such collections. The name chosen derives from the possibility of using such a collection in an analogue of a proof discovered by Katona and presented in Section 3.3.

Theorem 2.6.3. Suppose for $t \le k \le v$ and λ, $(\mathcal{B}, I, V)$ is a Katona sieve, then $\mathcal{B}$ is a t-design with corresponding parameters.

Proof. By the hypothesis, $|\mathcal{B}|\frac{\binom{k}{t}}{\binom{v}{t}} = \lambda$. So, continuing from the conclusion of the proof of Property 2.6.2 we have:

$$\lambda\binom{v}{t} = |\mathcal{B}|\binom{k}{t} = \sum_{B\in\mathcal{B}}\sum_{T\in\binom{V}{t}} \mathbb{1}_B(T) = \sum_{T\in\binom{V}{t}}\sum_{B\in\mathcal{B}} \mathbb{1}_B(T) \leq \lambda\binom{v}{t}.$$

Thus, they must all be equal and we conclude:

$$\frac{\sum_{T\in\binom{V}{t}}\sum_{B\in\mathcal{B}} \mathbb{1}_B(T)}{\binom{v}{t}} = \lambda. \tag{2}$$

The expression on the left is the average number of $B \in \mathcal{B}$ that a t-subset of V is contained in. But, since $\mathcal{B}$ is (λ, t)-disjoint, any given t-subset of V can be contained in *no more* than λ members of $\mathcal{B}$. So, the average value attains an upper bound on each of the constituent values. This is only possible if each of the values equals the upper bound as well. That is, we can conclude that for each $T \in \binom{V}{t}$,

$$\sum_{B\in\mathcal{B}} \mathbb{1}_B(T) = \lambda.$$

□

Like t-designs, Katona sieves can be concatenated to yield Katona sieves.

Property 2.6.4. Suppose that $(\mathcal{B}, I, V)$ and $(\mathcal{B}', I', V)$ are Katona sieves where $I \cap I' = \emptyset$. Then, the collection:

$$\mathcal{B}'' : I \cup I' \to 2^V : i \mapsto \begin{cases} \mathcal{B}(i) & \text{if } i \in I, \text{ and} \\ \mathcal{B}'(i) & \text{if } i \in I'. \end{cases}$$

is also a Katona sieve.

Proof. V hasn't changed and $\mathcal{B}''$ is k-uniform.

$$|\mathcal{B}''| = |\mathcal{B}| + |\mathcal{B}'| = \lambda\frac{\binom{v}{t}}{\binom{k}{t}} + \lambda'\frac{\binom{v}{t}}{\binom{k}{t}} = (\lambda+\lambda')\frac{\binom{v}{t}}{\binom{k}{t}}.$$

Therefore, we need only establish (λ, t)-disjointness. Suppose that $J \subseteq I \cup I'$ such that $(\mathcal{B}''\restriction_J, J, V)$ is t-intersecting. Then, if $J_I = J \cap I$ and $J_{I'} = J \cap I'$, both $(\mathcal{B}\restriction_{J_I}, J_I, V)$ and $(\mathcal{B}'\restriction_{J_{I'}}, J_{I'}, V)$ are t-intersecting subcollections of $\mathcal{B}$ and $\mathcal{B}'$ respectively. So, we have

$$|J| = |J_I| + |J_{I'}| \leq \lambda + \lambda'.$$

□

Finally, we have a negative result for the case $t = 2$. We need the following definition.

Definition 2.6.3. Given a collection $(\mathcal{B}, I, V)$ the *dual* of the collection $(\mathcal{B}^*, V, I)$ is the collection:

$$\mathcal{B}^* : V \to 2^I : v \mapsto \{i \in I; v \in \mathcal{B}(i)\}.$$

The dual of a symmetric 2-(v, k, λ) design is also a 2-(v, k, λ) design [CD06]. As a result, we have the following theorem.

Theorem 2.6.5. If $v > \lambda > 1$ and $(\mathcal{B}, V, I)$ is a 2-(v, k, λ) design with $v = b$, then $(\mathcal{B}, V, I)$ is not a Katona Sieve.

Proof. Because the dual of a symmetric 2-(v, k, λ) design is also a 2-(v, k, λ) design, $(\mathcal{B}^*, I, V)$ is also a 2-(v, k, λ) design. Condition 3 of Definition 2.5.1 for $\mathcal{B}^*$ says that given any two elements $i_1, i_2 \in I$, there exist exactly λ elements $v \in V$ such that $\{\mathcal{B}(i_1), \mathcal{B}(i_1)\} \subseteq \mathcal{B}^*(v)$. But,

$$\{v \in V; \{\mathcal{B}(i_1), \mathcal{B}(i_1)\} \subseteq \mathcal{B}^*(v)\} = \mathcal{B}(i_1) \cap \mathcal{B}(i_1).$$

Since $\lambda \geq 2$, this means that *any* two blocks of $\mathcal{B}$ are 2-intersecting. $\square$

Chapter 3

Motivation

Up to this point, we have been discussing terminology and notation, and some basic observations on collections of subsets. Now we will discuss the results motivating the study of (λ, t)-disjoint collections.

3.1 Erdös-Ko-Rado Theorem

The Erdos-Ko-Rado Theorem is a famous result in extremal set systems. It establishes an upper bound on the size of a t-intersecting family of k-subsets of a set V, provided that V is sufficiently large in relation to t and v.

Theorem 3.1.1. Given $k \in \mathbb{N}, k \geq t \geq 0$, there exists $v_0(k,t) \in \mathbb{N}$ such that the size of any k-uniform t-intersecting family, $(\mathscr{F}, I, V)$ with $|V| = v \geq v_0(k,t)$ is bounded as:

$$|\mathscr{F}| \leq \binom{v-t}{k-t}.$$

Note that for convenience, we will parametrize the theorem as EKR-t. So, EKR-2 states that for $t = 2$, given any $k \geq 2$, $v_0(k,2)$ exists. We will refer to EKR-1 simply as EKR. And, we will use the term EKR-t when referring to the entire set of theorems. For the case where $t = 1$ the EKR paper established that $v_0(k,1) = 2k$. EKR-t was also proven by establishing an upper-bound of $t + (k-t)\binom{k}{t}^3$ for $v_0(k,t)$. The original paper actually considers a class of families of subsets larger than k-uniform families, that is, Sperner families. Since we consider here only k-uniform families we have stated the theorem as such. Today, the exact value of $v_0(k,t)$ is known to be $(t+1)(k-t+1)$, a result generally credited to both Frankl [Fra78] and Wilson [Wil84]. The proofs contained in the EKR paper will not be presented here. Instead, in the following section, a proof of EKR-1, similar to Katona's original proof that $v_0(k,1) = 2k$, will be presented [Kat72]. This proof inspires the name and consideration of the t-designs discussed here. Katona's proof will be extended to show that the conclusion of EKR-t holds for v and k *assuming* the existence of a Katona sieve with these parameters. Otherwise no other proof of EKR-t, nor the exact value of $v_0(k,t)$ for $t > 1$, will be given.

First, we want to establish that the bound on the size of the t-intersecting family in the theorem is "tight". For EKR-t, what this means is that, given k and t, there actually is a family of t-intersecting subsets of V whose size attains the bound of $\binom{v-t}{k-t}$. It is formed by choosing t elements of V to be included in every member of the family and selecting every possible $k-t$ subset of the remaining elements. That is, we take every k-subset of V containing a given t-subset.

For instance, since, $6 \geq (1+1)(3-1+1) = 2 * 3$, EKR-1 along with the known value of $v_0(3,1) = 6$ establishes that no 1-intersecting family of 3-subsets of a set of size 6 is larger than the following:

$$\begin{bmatrix} 1&1&1&1&1&1&1&1&1&1 \\ 1&1&1&1&0&0&0&0&0&0 \\ 1&0&0&0&1&1&1&0&0&0 \\ 0&1&0&0&1&0&0&1&1&0 \\ 0&0&1&0&0&1&0&1&0&1 \\ 0&0&0&1&1&0&1&0&1&1 \end{bmatrix},$$

with $10 = \binom{6-1}{3-1}$ members and formed from selecting one element of V to be in all members and completing with all possible selections of 2 elements from the remaining. On the other hand, consider the family formed from all 4-subsets of a set of size 6 containing a given 2-subset. This is an incidence matrix of such a family:

$$\begin{bmatrix} 1&1&1&1&1&1 \\ 1&1&1&1&1&1 \\ 1&1&1&0&0&0 \\ 1&0&0&1&1&0 \\ 0&1&0&1&0&1 \\ 0&0&1&0&1&1 \end{bmatrix}.$$

Remember that the exact value of $v_0(4,2)$ has been proven to be $(2+1)(4-2+1) = 9$. So, EKR-2 does not apply to this case. Indeed, in this case we can find a larger family of 2-intersecting 4-subsets:

$$\begin{bmatrix} 1&1&1&1&1&1&0&0&0 \\ 1&1&1&0&0&0&1&1&1 \\ 0&0&0&1&1&1&1&1&1 \\ 1&1&0&1&1&0&1&1&0 \\ 1&0&1&1&0&1&1&0&1 \\ 0&1&1&0&1&1&0&1&1 \end{bmatrix}.$$

3.2 Katona's Circle Family

Katona found a very elegant double counting proof of EKR-1 and its bound $v_0(k, 1)$. In the proof, a $(\lambda, 1)$-disjoint family of subsets k-uniform subsets is used. Katona's proof relies on a lemma, establishing that a family of sets is $(\lambda, 1)$-disjoint. He then uses this family as a sieve to count all the images of members of an arbitrary k-uniform 1-intersecting family, $(\mathscr{F}, I, V)$ under a set of permutations. We begin by defining his family and providing an example.

Definition 3.2.1. Given parameters k and v where $2k \leq v$, a *Katona circle family* for these parameters is the family of subsets:

$$\mathscr{R} = \big\{\{i \bmod v, i+1 \bmod v, \ldots, i+(k-1) \bmod v\};\ i \in [0, v-1]\big\}.$$

For the parameters $k = 4$ and $v = 8$ the incidence matrix of such a family is shown in Figure 6.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 6: Katona's Circle Family for $v = 8$

It is the orbit of the 4-subset, $\{0, 1, 2, 3\}$, of the 8 element set, $\{0, 1, \ldots, 7\}$, under the action of the cyclic group of order 8. The important thing to note is that any selection of greater than $k = 4$ members (columns) includes at least one pair of members that does not intersect. That is, this family is $(4, 1)$-disjoint. Katona's Lemma is that Katona's cycle family is $(k, 1)$-disjoint for any v.

Lemma 3.2.1. (*Katona's Lemma* [Kat72]). If $v \geq 2k$, then the family of subsets of $\bar{v}$,

$$\mathscr{R} = \big\{\{i \bmod v, i+1 \bmod v, \ldots, i+(k-1) \bmod v\};\ i \in [0, v-1]\big\},$$

is $(k, 1)$-disjoint.

Proof. We need to show that if $v \geq 2k$ for any selection of $k+1$ members of the family, at least one pair does not intersect. To see this, suppose that a subfamily $\mathscr{F} \subseteq \mathscr{R}$ is intersecting and that $F = \{i \bmod v, i+1 \bmod v, \ldots, i+(k-1) \bmod v\} \in \mathscr{F}$. There are $2k-2$ members of $\mathscr{R}$ that intersect F and they can be separated into two classes: those that intersect F on the "lower" end and those that intersect F on the "higher" end:

$$\begin{aligned}\mathscr{A} &= \big\{\{i+m \bmod v, i+m+1 \bmod v, \ldots, i+m+(k-1) \bmod v\};\ m \in [1,k-1]\big\},\ \text{and}\\ \mathscr{B} &= \big\{\{i-m \bmod v, i-m+1 \bmod v, \ldots, i-m+(k-1) \bmod v\};\ m \in [1,k-1]\big\}.\end{aligned}$$

Since all members of $\mathscr{F}$ intersect F, $\mathscr{F}$ must be a subset of $\{F\} \cup \mathscr{A} \cup \mathscr{B}$. Since $2k \leq v$, $\mathscr{A}$ and $\mathscr{B}$ are disjoint. Moreover, given $m \in [1,k-1]$,

$$\begin{aligned}&\{i+m \bmod v, i+m+1 \bmod v, \ldots, i+m+(k-1) \bmod v\\ &\quad \cap \{i-m \bmod v, i-m+1 \bmod v, \ldots, i-m+(k-1) \bmod v\} = \emptyset.\end{aligned}$$

So, these two members of $\mathscr{R}$ cannot both be in $\mathscr{F}$ (since $\mathscr{F}$ is 1-intersecting). That is each selection of an element in $\mathscr{A}$ eliminates a possible selection of a member of $\mathscr{B}$ and *vice versa*. Therefore,

$$|\mathscr{F}| \leq 1 + \max\{|\mathscr{A}|, |\mathscr{B}|\} = 1 + (k-1) = k.$$

□

Katona's Lemma for $\mathscr{R}$ is simply the statement that $\mathscr{R}$ is $(k,1)$-disjoint if $v \geq 2k$. Since, $\mathscr{R}$ is k-uniform, $\lambda = k$, and $t = 1$,

$$\lambda \frac{\binom{v}{t}}{\binom{k}{t}} = k\frac{v}{k} = v = |\mathscr{R}|.$$

This means that $\mathscr{R}$ is a Katona sieve, as defined in Definition 2.6.2.

The requirement that $v \geq 2k$ cannot be removed because the value of $v_0(k,1) = 2k$. A construction equivalent to Katona's for $v = 7$ and $k = 4$, with $v < 2k$, looks like this:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1\\ 1 & 1 & 0 & 0 & 0 & 1 & 1\\ 1 & 1 & 1 & 0 & 0 & 0 & 1\\ 1 & 1 & 1 & 1 & 0 & 0 & 0\\ 0 & 1 & 1 & 1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1 & 1 & 1 & 0\\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

All of this family's members intersect. Indeed, this example extends to any k and v where $2k > v$. All members of the corresponding family will 1-intersect because v is too small to contain two disjoint k-subsets.

3.3 Katona's Circle Argument

Now, equipped with Katona's lemma, we can move on to his proof. Recall the statement of EKR-1.

Theorem 3.3.1. Given $k \in \mathbb{N}, k \geq 1$, there exists $v_0(k,1) \in \mathbb{N}$ such that the size of any k-uniform 1-intersecting family, $(\mathscr{F}, I, V)$ with $|V| = v \geq v_0(k,1)$ is bounded as:

$$|\mathscr{F}| \leq \binom{v-1}{k-1}.$$

Katona's proof of EKR-1 [Kat72]. Assume that $v \geq 2k$ and consider the summation:

$$\sum_{\rho \in \mathcal{S}_V} \sum_{F \in \mathscr{F}} \sum_{R \in \mathscr{R}} \mathbb{1}_{\rho F}(R).$$

Here $\mathcal{S}_V$ denotes the set of all permutations of the elements of V and $\mathscr{R}$ is the Katona circle family for k and v. This summation is the number of instances where a permutation maps an element of $\mathscr{F}$ to an element of $\mathscr{R}$. Since $\mathscr{F}$ is intersecting, so is the family $\rho\mathscr{F} = \{\rho F; F \in \mathscr{F}\}$. Therefore, since the largest intersecting subfamily of $\mathscr{R}$ is of size k, a given permutation can map at most k elements of $\mathscr{F}$ to elements of $\mathscr{R}$.

$$\sum_{\rho \in S_V} \sum_{F \in \mathscr{F}} \sum_{R \in \mathscr{R}} \mathbb{1}_{\rho F}(R) \quad \leq \quad \sum_{\rho \in S_V} k = kv!. \tag{3}$$

On the other hand,

$$\sum_{\rho \in S_V} \sum_{F \in \mathscr{F}} \sum_{R \in \mathscr{R}} \mathbb{1}_{\rho F}(R) \quad = \quad \sum_{F \in \mathscr{F}} \sum_{R \in \mathscr{R}} \sum_{\rho \in S_V} \mathbb{1}_{\rho F}(R).$$

Given two k-subsets of V, there are $(v-k)!k!$ permutations that map one to the other. Using this fact and evaluating the summations:

$$\begin{aligned}
\sum_{F \in \mathscr{F}} \sum_{R \in \mathscr{R}} \sum_{\rho \in S_V} \mathbb{1}_{\rho F}(R) \quad &= \quad \sum_{F \in \mathscr{F}} \sum_{R \in \mathscr{R}} (v-k)!k! \\
&= \quad \sum_{F \in \mathscr{F}} |\mathscr{R}|(v-k)!k! \\
&= \quad |\mathscr{F}||\mathscr{R}|(v-k)!k! \\
&= \quad |\mathscr{F}|v(v-k)!k!.
\end{aligned}$$

Combining these results,

$$|\mathscr{F}|v(v-k)!k! = \sum_{F \in \mathscr{F}} \sum_{R \in \mathscr{R}} \sum_{\rho \in S_V} \mathbb{1}_{\rho F}(R) = \sum_{\rho \in S_V} \sum_{F \in \mathscr{F}} \sum_{R \in \mathscr{R}} \mathbb{1}_{\rho F}(R) \leq kv!.$$

Finally, if $|\mathscr{F}|$ is isolated, we have:

$$|\mathscr{F}| \leq \frac{k}{v}\binom{v}{k} = \binom{v-1}{k-1}.$$

□

It is in Equation (3) of the proof that the family $\mathscr{R}$ is used to sieve no more than k elements out of $\rho\mathscr{F}$ for each permutation ρ. Now, observe that although $\mathscr{R}$ was a family of sets, the fact that the blocks are distinct was used only to establish Katona's Lemma and not in the proof itself. Indeed, given v and k, not only could any Katona sieve with parameter $t = 1$ substitute for $\mathscr{R}$ in this proof of the conclusion of EKR-1 for v and k, but any Katona sieve with parameters t, k, and v could be substituted yielding a proof that the conclusion of EKR-t holds for these values. The following theorem is not a full generalization of Katona's proof of EKR-1 to a proof of EKR-t. The existence of a Katona sieve would be required for this to prove EKR-t.

Theorem 3.3.2. If there exists a Katona sieve for parameters t, k, v, $0 \le t \le k \le v$ and some $\lambda > 0$, then, for any t-intersecting family $\mathscr{F}$ of k-subsets of V,

$$|\mathscr{F}| \le \binom{v-t}{k-t}.$$

Proof. Let $\mathcal{R}$ be a Katona sieve. Given a t-intersecting family $\mathscr{F}$, we again we consider the summation:

$$\sum_{\rho\in\mathcal{S}_V}\sum_{R\in\mathcal{R}}\sum_{F\in\mathscr{F}} \mathbb{1}_R(\rho F).$$

For any permutation $\rho \in \mathcal{S}_V$, $\rho\mathscr{F} = \{\rho F; F \in \mathscr{F}\}$ is again a t-intersecting family. Since $\mathcal{R}$ is (λ, t)-disjoint and k-uniform, and since $\rho\mathscr{F}$ is t-intersecting, there are at most λ members $F \in \mathscr{F}$ such that $\rho F = R$ for some $R \in \mathcal{R}$. That is, given, ρ:

$$\sum_{R\in\mathcal{R}}\sum_{F\in\mathscr{F}} \mathbb{1}_R(\rho F) \le \lambda.$$

And, we have

$$\sum_{\rho\in\mathcal{S}_V}\sum_{R\in\mathcal{R}}\sum_{F\in\mathscr{F}} \mathbb{1}_R(\rho F) \le \sum_{\rho\in\mathcal{S}_V} \lambda = \lambda v!. \tag{4}$$

On the other hand,

$$\sum_{\rho\in\mathcal{S}_V}\sum_{R\in\mathcal{R}}\sum_{F\in\mathscr{F}} \mathbb{1}_R(\rho F) = \sum_{F\in\mathscr{F}}\sum_{R\in\mathcal{R}}\sum_{\rho\in\mathcal{S}_V} \mathbb{1}_R(\rho F).$$

And, given $F \in \mathscr{F}$, for each $R \in \mathcal{R}$ there are $(v-k)!k!$ permutations that map F to R:

$$\sum_{R\in\mathcal{R}}\sum_{\rho\in\mathcal{S}_V} \mathbb{1}_R(\rho F) = |\mathcal{R}|(v-k)!k!.$$

Summing over all of $\mathscr{F}$ we have

$$\sum_{F\in\mathscr{F}}\sum_{R\in\mathcal{R}}\sum_{\rho\in\mathcal{S}_V} \mathbb{1}_R(\rho F) = |\mathscr{F}|(v-k)!k!.$$

Combining this with Inequality (4) gives

$$|\mathscr{F}| \le \frac{\lambda v!}{|\mathcal{R}|(v-k)!k!} = \frac{\lambda}{|\mathcal{R}|}\binom{v}{k}.$$

By the definition of a Katona sieve,

$$\frac{\lambda}{|\mathcal{R}|}\binom{v}{k} = \frac{\binom{k}{t}}{\binom{v}{t}}\binom{v}{k}.$$

But,

$$\frac{\binom{k}{t}}{\binom{v}{t}}\binom{v}{k} = \binom{v-t}{k-t}.$$

Therefore, we can conclude:

$$|\mathscr{F}| \leq \binom{v-t}{k-t}.$$

□

Katona himself had noted that a Steiner system can be used for such a proof [Kat00]. This is the case where $\lambda = 1$ in Theorem 3.3.2. And, EKR-t itself is equivalent to the statement that for v sufficiently large $\binom{\bar{v}}{k}$ is a Katona sieve with $\lambda = \binom{v-t}{k-t}$. By Theorem 3.3.2, regardless of the value of λ and regardless of whether there are repeated blocks, a Katona sieve implies EKR-t for v and k. This is a new result as Katona sieves themselves have not previously been identified.

As mentioned in Section 3.1, the exact value of $v_0(k,t)$, as defined by EKR-t, has been shown to be $v_0(k,t) = (t+1)(k-t+1)$ [Wil84, Fra78]. We do not prove this here, but this means that for $v \geq (t+1)(k-t+1)$ there exists a Katona sieve for some λ, namely the t-design $\binom{V}{k}$ with $\lambda = \binom{v-t}{k-t}$. We will see in Corollary 3.3.4, the value $(t+1)(k-t+1)$ is also a lower bound on the size of V for a Katona sieve to exist and we will later use this to eliminate possibilities. We present the proof that Wilson cites as due to Frankl [Fra78, Wil84] of the lower bound of $v_0(k,t)$ beginning with an explanation of the construction used.

Let $Y \in \binom{V}{t+2}$ and consider the family of k-subsets of V:

$$\mathscr{F} = \{F \in \binom{V}{k}; |F \cap Y| \geq t+1\}.$$

Frankl's proof that $(t+1)(k-t+1)$ is a lower bound for $v_0(k,t)$ compares the size of $\mathscr{F}$ with the family, $\mathscr{F}'$, of k-subsets of V containing a fixed t-subset X of V. The size of $\mathscr{F}'$ is given by $\binom{v-t}{k-t}$. Let $k = 3$, $t = 1$, and consider how the sizes of these families changes as $|V|$ varies from 5 to 7:

$$|V| = 5: \quad \mathscr{F} = \left.\left[\begin{array}{ccccccc} 1&1&1&1&1&0&0\\ 1&1&1&0&0&1&1\\ 1&0&0&1&1&1&1\\ 0&1&0&1&0&1&0\\ 0&0&1&0&1&0&1 \end{array}\right]\begin{array}{l} \\ Y \\ \\ \\ \\ \end{array}\right. . \qquad \mathscr{F}' = \left[\begin{array}{cccccc} 1&1&1&1&1&1\\ 1&1&1&0&0&0\\ 1&0&0&1&1&0\\ 0&1&0&1&0&1\\ 0&0&1&0&1&1 \end{array}\right]\begin{array}{l} \} X \\ \\ \\ \\ \\ \end{array} .$$

$$|V|=6:\quad \mathscr{F}=\begin{bmatrix}1&1&1&1&1&1&1&0&0&0\\1&1&1&1&0&0&0&1&1&1\\1&0&0&0&1&1&1&1&1&1\\0&1&0&0&1&0&0&1&0&0\\0&0&1&0&0&1&0&0&1&0\\0&0&0&1&0&0&1&0&0&1\end{bmatrix}\begin{matrix}\left.\begin{matrix}\\\\\end{matrix}\right\}Y\\\\\\\end{matrix}. \qquad \mathscr{F}'=\begin{bmatrix}1&1&1&1&1&1&1&1&1&1\\1&1&1&1&0&0&0&0&0&0\\1&0&0&0&1&1&1&0&0&0\\0&1&0&0&1&0&0&1&1&0\\0&0&1&0&0&1&0&1&0&1\\0&0&0&1&0&0&1&0&1&1\end{bmatrix}\begin{matrix}\}\ X\\\\\\\\\\\end{matrix}.$$

$$|V|=7:\quad \mathscr{F}=\begin{bmatrix}1&1&1&1&1&1&1&1&1&0&0&0&0\\1&1&1&1&1&0&0&0&0&1&1&1&1\\1&0&0&0&0&1&1&1&1&1&1&1&1\\0&1&0&0&0&1&0&0&0&1&0&0&0\\0&0&1&0&0&0&1&0&0&0&1&0&0\\0&0&0&1&0&0&0&1&0&0&0&1&0\\0&0&0&0&1&0&0&0&1&0&0&0&1\end{bmatrix}\begin{matrix}\left.\begin{matrix}\\\\\end{matrix}\right\}Y\\\\\\\\\end{matrix}.$$

$$\mathscr{F}'=\begin{bmatrix}1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\1&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\0&1&0&0&0&1&0&0&0&1&1&1&0&0&0\\0&0&1&0&0&0&1&0&0&1&0&0&1&1&0\\0&0&0&1&0&0&0&1&0&0&1&0&1&0&1\\0&0&0&0&1&0&0&0&1&0&0&1&0&1&1\end{bmatrix}\begin{matrix}\}\ X\\\\\\\\\\\\\end{matrix}.$$

Although $\mathscr{F}$ begins larger than $\mathscr{F}'$, $\mathscr{F}'$ grows faster as $|V|$ is increased. Regardless of k and t, this will be the case. The size of $\mathscr{F}'$ eventually surpasses the size of $\mathscr{F}$ as v grows. To be exact, this occurs when $|V|=(t+1)(k-t+1)$.

Theorem 3.3.3 (*Extension of Tit's Bound*)**.** If $v<(t+1)(k-t+1)$ and $|V|=v$ there exists a family of t-intersecting k-subsets of V such that

$$|\mathscr{F}|>\binom{v-t}{k-t}.$$

Proof. (due to Frankl [Fra78, Wil84])

Let $Y\in\binom{V}{t+2}$ and consider the family of k-subsets of V:

$$\mathscr{F}=\{F\in\binom{V}{k};|F\cap Y|\geq t+1\}.$$

Any two sets in $\mathscr{F}$ are t-intersecting, and

$$|\mathscr{F}|=\binom{t+2}{t+1}\binom{v-(t+2)}{k-(t+1)}+\binom{v-(t+2)}{k-(t+2)}.$$

The first term in this sum is the number k-subsets of V whose intersection with Y is exactly of size $t+1$, and the second the number of k-subsets whose intersection with Y is exactly of size $t+2$. Rearranging terms in the previous equation, we have:

$$|\mathscr{F}| = \frac{(t+2)(v-k)(k-t)-(k-t-1)(k-t)}{(v-t-1)(v-t)}\binom{v-t}{k-t}.$$

If $|\mathscr{F}| \leq \binom{v-t}{k-t}$, we must have:

$$\frac{(t+2)(v-k)(k-t)-(k-t-1)(k-t)}{(v-t-1)(v-t)} \leq 1.$$

Letting $x = v-t-1$ and $y = (k-t)$ and rearranging terms, this becomes

$$x^2 - (t+2)xy - (t+1)y^2 = (x-(t+1)y)(x-y) \geq -(x-(t+1)y).$$

Since $x - y = v-k-1$ is greater than or equal to zero unless $v = k$, if $v \neq k$, $x-(t+1)y \geq 0$, or $v \geq (t+1)(k-t+1)$. □

Because of this lower bound on the value of $v_0(k,t)$, we also have a lower bound for the existence of a Katona sieve with parameters v,k,t, and *any* λ.

Corollary 3.3.4. If $(\mathcal{R}, I, V)$ is a Katona sieve for parameters t, k, v, and λ, then

$$v \geq (t+1)(k-t+1).$$

Proof. We combine Theorem 3.3.2 and Theorem 3.3.3 and argue as follows: suppose that $(\mathcal{R}, I, V)$ is a Katona sieve where $v < (t+1)(k-t+1)$. Then, it can be used via Theorem 3.3.2 to prove that the conclusion of EKR-t holds for the particular values t,k, and v . But, the conclusion of EKR-t cannot hold when $v < (t+1)(k-t+1)$, because of Theorem 3.3.3. □

This lower bound on the value of v for a Katona sieve to exist is a direct extension of the Tit's lower bound on the value of v for a Steiner system to exist [Tit64]. The question that remains is if $v \geq (t+1)(k-t+1)$, for which values of λ does there exist a Katona sieve. We know that one exists for $\lambda = \binom{v-t}{k-t}$, but what are the smaller values of λ for which there exists a Katona sieve? This work is a computer-based search to begin answering this question.

Chapter 4

Computer Search

Up until this point, we have been explaining our problem and its origin. Now, we will describe programs that can be used to search for Katona Sieves. The reader is invited to recall that the *incidence matrix* representing the collection $\binom{6}{3}$ is:

$$\begin{bmatrix} 1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \\ 1&1&1&1&0&0&0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 1&0&0&0&1&1&1&0&0&0&1&1&1&0&0&0&1&1&1&0 \\ 0&1&0&0&1&0&0&1&1&0&1&0&0&1&1&0&1&1&0&1 \\ 0&0&1&0&0&1&0&1&0&1&0&1&0&1&0&1&1&0&1&1 \\ 0&0&0&1&0&0&1&0&1&1&0&0&1&0&1&1&0&1&1&1 \end{bmatrix}.$$

Since we have not yet carefully looked at an incidence matrix representing a Katona sieve, we will now use this as an example. We will confirm that this matrix satisfies the definition of a Katona sieve, or rather, that this matrix satisfies the equivalent conditions for a matrix to represent a Katona sieve.

The collection is clearly 3-uniform. We want to show that it is $(4, 2)$-disjoint. That is, we are choosing $\lambda = 4$ and $t = 2$. So we will argue that and choice of $\lambda + 1 = 5$ columns includes a pair that does not 2-intersect. Note that this matrix is special because it represents *all* 3-subsets of a set of size 6. What this means is that for any choice of two rows, the rows and the columns can be rearranged to yield the matrix above with these two rows moved to the first two positions. Now suppose that we select 5 columns pairwise 2-intersecting and that we have rearranged the matrix so that the first two columns in the above matrix are included in the selection of 5 columns. Only the third, fourth, fifth, and eleventh columns 2-intersect both of the first two columns. Of these, no selection of three are pairwise 2-intersecting. This contradicts the existence of these 5 pairwise 2-intersecting columns.

Finally, we check the size condition of Definition 2.6.2:

$$20 = 4 * \binom{6}{1}\binom{3}{1}.$$

So this is indeed a Katona sieve.

We will present some programs to search for Katona sieves of given parameters in Section 4.2. We begin with a discussion of general techniques for testing the conditions that in modified form will apply to all the programs.

4.1 Condition Testing

In this section, we will review the conditions for a collection to represent a Katona Sieve and explain how these conditions will be tested.

Property 4.1.1. A $v \times b$ matrix, M, represents a Katona sieve if it satisfies the following conditions:

1. M is k-uniform, that is each column contains k 1's.
2. M has $b = \lambda \frac{\binom{v}{t}}{\binom{k}{t}}$ columns.
3. For any $(\lambda+1)$-subset $J \subseteq \bar{b}$, there exists a pair $j, j' \in J$ where the inner product, $M_j \cdot M_{j'}$, of the columns M_j and $M_{j'}$ is less than t.

Note that the third condition of Property 4.1.1 is stronger than the following, which together with the first and second conditions imply only that M is a t-design.

3'. For any t-subset $T \subseteq \bar{v}$, the number of columns containing T is λ, $|\{j \in \bar{v}; \mathbb{1}_T \cdot M_j = t\}| = \lambda$.

Indeed, that Conditions 1, 2 and 3 imply 3' is the result of Theorem 2.6.3. However, for convenience, we will still refer to 3' as Condition 3' of Property 4.1.1. For the example above, an incidence matrix of $\binom{\bar{6}}{3}$, Conditions 1 and 2 were easily verifiable by the reader. Condition 3 was more difficult to verify, and we relied on specific properties that this matrix has. We now explain a more general technique to test this property. To illustrate, we use the following, smaller, 2-design with $t = 2$, $k = 3$, $v = 6$, and $\lambda = 2$:

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

We show that this also represents a Katona sieve. Conditions 1 and 2 are readily verifiable. In order to verify Condition 3, we construct an auxilliary graph with the the blocks as vertices and an edge between two vertices if and only if the two blocks are 2-intersecting. First we can compute the matrix $A^T A$. The entry (i, j) of $A^T A$ is the inner product of column i and column j of the matrix A:

$$A^T A = \begin{bmatrix} 3 & 2 & 2 & 1 & 1 & 2 & 1 & 1 & 1 & 1 \\ 2 & 3 & 1 & 2 & 1 & 1 & 2 & 1 & 1 & 1 \\ 2 & 1 & 3 & 1 & 2 & 1 & 1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 3 & 2 & 1 & 1 & 1 & 1 & 2 \\ 1 & 1 & 2 & 2 & 3 & 1 & 1 & 2 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 & 3 & 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 1 & 1 & 1 & 3 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 & 1 & 2 & 1 & 3 & 2 \\ 1 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 2 & 3 \end{bmatrix}.$$

This matrix is known as the *block intersection matrix* of A. Since we are interested in t-intersecting columns of A, we will be considering a graph that we will denote $\phi(A)$ throughout the remainder of this work. The adjacency matrix of $\phi(A)$ is derived from $A^T A$, with an edge between two vertices if the dot product of the corresponding columns is greater than or equal to t. In this case, $t = 2$, so we have the adjacency matrix shown in Figure 7.

$$\phi(A) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 7: $\phi(A)$, the t-intersection graph of A

Suppose that this graph has a clique of size $\lambda + 1$. Then, there would be $\lambda + 1$ columns of the original collection, A, such that the inner product of any pair of them is at least t. So, A would not be (λ, t)-disjoint. On the other hand, if no such clique exists, then in any selection of $\lambda + 1$ columns of A at least one pair has dot product less than t. That is, A is the incidence matrix of a (λ, t)-disjoint collection if and only if $\phi(A)$ has no cliques of size greater than λ. A picture of the graph $\phi(A)$ can be seen in Figure 8. Using this picture, the reader can easily see that there are no cliques of size $\lambda + 1 = 3$, and, therefore, our collection is (λ, t)-disjoint. This together with Conditions 1 and 2 of Property 4.1.1 makes the collection a Katona sieve.

Note that any algorithm can be used for finding the maximum size of a clique in $\phi(A)$. Initially, the program used for finding the maximum clique size was Sampo Niskanen and Patric Östergård's *Cliquer* [Ns03]. Later, we used a simple routine that saves all cliques of size less than or equal to λ, extending and retracting cliques as columns were filled. This is not included in the programs and we will simply refer to an arbitrary algorithm *MaxClique* for determining the clique-number of $\phi(A)$.

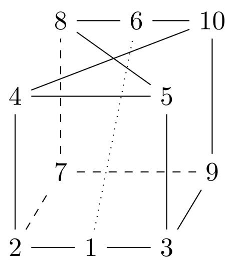

Figure 8: Picture of $\phi(A)$

4.2 Programs

The programs used are all backtracking algorithms which progressively fill an initially empty matrix. For demonstration, we begin with a very general, though impractical, program that can fill the entries in any order. As we proceed to later programs, we will use specific orders to fill the entries. The first program would be very slow and was not implemented and the second was not implemented because the third program is a minor modification of the second which was implemented. The fourth and fifth programs were implemented.

For the remainder of this chapter, we assume that $t = 2$ unless otherwise noted.

4.2.1 Program 1

The concepts are in place to define an initial backtracking program that finds a (0,1)-matrix to represent each unique nonisomorphic Katona sieve given parameters t,v,k, and λ. We write the first program in two functions. The first tests to see if a completed matrix satisfies the conditions to represent a Katona sieve, the conditions of Property 4.1.1. The second function is recursive. Over the course of the recursion, an initially empty matrix is filled in an arbitrary order with 0's and 1's. The recursion serves to consider all possible completions of the matrix.

Condition 2 of Property 4.1.1 says that we can assume *a priori* the value of b. Condition 3', which followed from Theorem 2.6.3, tells us that the matrix must also represent a 2-design. Since we are generating a 2-design we can use the fact that all rows must have $r = \frac{bk}{v}$ 1's. Moreover, since we are now assuming that $t = 2$, we use the fact that the inner product of any pair of rows must be λ. Since checking these properties is computationally less intensive than clique testing, it is advantageous to test these prior to performing clique testing. The condition testing is presented in Algorithm 1.

Algorithm 1 Program 1 (initial version)

```
    Prerequisites:
      • A maximum clique-testing algorithm, MaxClique.
      • A canonical form $\Psi$ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
      • A data-structure to hold solutions, $\mathfrak{S}$.

 1: function ConditionsPass($A$)
 2:     For each $j$ do
 3:         if $\sum_{i=0}^{v-1} A_{(i,j)} \neq k$ then return false.
 4:     For each $i$ do
 5:         if $\sum_{j=0}^{b-1} A_{(i,j)} \neq r$ then return false.
 6:     For each $\{i, i'\} \in \binom{v}{2}$ do
 7:         if not $\sum_{j=0}^{b-1} A_{(i,j)} A_{(i',j)} = \lambda$ then return false.
 8:     if MaxClique($\phi(A)$) $> \lambda$ then return false.
 9:     return true.
10: end function
```

The second function is the backtracking recursion that fills the matrix with 0's and 1's. At a given node of the recursion at depth $m \leq vb$, the recursive function begins with the matrix $\mathbf{H}$ having entries $(i_1, j_1), (i_2, j_2), \ldots (i_{m-1}, j_{m-1})$ filled. The two children of this node begin with a 0 as entry (i_m, j_m) and a 1 as entry (i_m, j_m). When $m = vb + 1$, $\mathbf{H}$ is completely filled and the conditions are tested. If the matrix satisfies the conditions, its canonical form is then compared to the canonical forms of all previously seen matrices that satisfied the conditions which were stored in $\mathfrak{S}$. If it is new, in that its canonical form was not previously seen, its canonical form is also stored in $\mathfrak{S}$. Algorithm 2 presents this recursion.

As it is, this program generates all 2^{vb}, $v \times b$ matrices, we would like to prune this search tree by eliminating some of the partially completed matrices which are impossible to complete. This will be done by moving the condition testing as specified in ConditionsPass to the nodes of the search tree. In terms of the pseudocode, we wish to move the test, ConditionsPass, before the statement "**If** $m = vb + 1$" of Algorithm 2. However, the actual routines encompassing ConditionsPass will need to be modified to accomplish this. Let us consider the conditions specified in ConditionsPass in relation to a matrix only partially filled with 0's and 1's:

Algorithm 2 Program 1 (initial version, continued)

```
11: function RECURSE($m$)
12:     if $m = vb + 1$ then
13:         if not CONDITIONSPASS($\mathbf{H}$) then return
14:         For each $A \in \mathfrak{S}$ do
15:             if $\Psi(\mathbf{H}) = A$ then return
16:         Copy $\Psi(\mathbf{H})$ into $\mathfrak{S}$
17:     $\mathbf{H}_{(i_m, j_m)} \leftarrow 0$
18:     RECURSE($m + 1$)
19:     $\mathbf{H}_{(i_m, j_m)} \leftarrow 1$
20:     RECURSE($m + 1$)
21:     $\mathbf{H}_{(i_m, j_m)} \leftarrow ?$
22: end function
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & ? & 0 & 0 & 1 & ? & ? & ? & 0 \\ 0 & 1 & ? & ? & ? & ? & ? & ? & ? & ? \\ 1 & 0 & ? & 1 & ? & ? & ? & ? & ? & 1 \\ 0 & 0 & ? & 0 & 1 & 0 & ? & ? & ? & 0 \\ 0 & 0 & ? & 1 & 1 & 1 & ? & ? & ? & 1 \end{bmatrix}.$$

Here, “?” indicates values not yet set. This matrix corresponds to a possible internal node of the search tree. Since the matrix is not complete we need to determine whether it is *impossible* for any of the leaf nodes descended from this node to be a solution. If this is the case, then we do not need to continue the recursion to the children of this node. For this program, we will only illustrate this with the two conditions specified by:

```
2: For each $j$ do
3:     if $\sum_{i=0}^{v-1} A_{(i,j)} \neq k$ then return false.
4: For each $i$ do
5:     if $\sum_{j=0}^{b-1} A_{(i,j)} \neq r$ then return false.
```

since these are straightforward to modify. Condition 1 of 4.1.1 simply says that each column contains k 1’s and condition 2 that each row contains r 1’s. In the context of an incomplete matrix, Condition 1 becomes that the number of 1’s in each column doesn’t exceed k *and* the number of 0’s in each column doesn’t exceed $v - k$ as otherwise there could not be k 1’s in the completed matrix. Similarly, condition 2 becomes that the number of 1’s in each row cannot exceed r and the number of 0's cannot exceed $b - r$.

Algorithm 3 Test row sums (final version).

```
1: function RowSumsPass($A$)
2:     For each $i$ do
3:         if $\sum_{\substack{j=0 \\ A_{(i,j)}=1}}^{b-1} A_{(i,j)} > r$ then return false.
4:         if $\sum_{\substack{j=0 \\ A_{(i,j)}=0}}^{b-1} (1 - A_{(i,j)}) > b - r$ then return false.
5:         return true.
6: end function
```

Algorithm 4 Test column sums (final version).

```
1: function ColumnSumsPass($A$)
2:     For each $j$ do
3:         if $\sum_{\substack{i=0 \\ A_{(i,j)}=1}}^{v-1} A_{(i,j)} > k$ then return false.
4:         if $\sum_{\substack{i=0 \\ A_{(i,j)}=0}}^{v-1} (1 - A_{(i,j)}) > v - k$ then return false.
5:         return true.
6: end function
```

On the other hand, the other conditions tested by ConditionsPass require a fairly elaborate modification to be tested at internal nodes when the entries are being filled in any order. Since this program only serves for demonstration, we refrain from presenting them here. But we will do so in later programs, where it is more simple.

Before proceeding, we update the statement of the program to reflect the preceeding discussion. We introduce two separate functions, NodeConditionsPass and LeafConditionsPass, separating the conditions that can be tested only at the leaves from those that can now be tested at the internal nodes of the recursive tree. This updated program is presented in Algorithm 5.

Both the (λ, t)-disjointness testing and isomorphism testing are performed only at the leaf nodes. This will be addressed in the next program. In order to perform isomorphism testing at the internal nodes of the search tree, we will assume more about the order in which the entries are being filled. That is, in all subsequent programs, the order in which the entries of the matrix are filled will be more restrictive.

Algorithm 5 Program 1 (final version).

```
Prerequisites:
  • A maximum clique-testing algorithm, MaxClique.
  • A canonical form $\Psi$ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
  • A data-structure to hold solutions, $\mathfrak{S}$.

function NodeConditionsPass($A$)
    if not RowSumsPass($A$) then return false.
    if not ColumnSumsPass($A$) then return false.
    return true.
end function

function LeafConditionsPass($A$)
    For each $\{i, i'\} \in \binom{v}{2}$ do
        if not $\sum_{j=0}^{b-1} A_{(i,j)} A_{(i',j)} = \lambda$ then return false.
    if $MaxClique(\phi(A)) > \lambda$ then return false.
    return true.
end function.

function Recurse($m$)
    if not NodeConditionsPass($\mathbf{H}$) then return
    if $m = vb + 1$ then
        if not LeafConditionsPass($\mathbf{H}$) then return
        For each $A \in \mathfrak{S}$ do
            if $\Psi(\mathbf{H}) = A$ then return
        Copy $\Psi(\mathbf{H})$ into $\mathfrak{S}$
    $\mathbf{H}_{(i_m, j_m)} \leftarrow 0$
    Recurse($m + 1$)
    $\mathbf{H}_{(i_m, j_m)} \leftarrow 1$
    Recurse($m + 1$)
    $\mathbf{H}_{(i_m, j_m)} \leftarrow ?$
end function
```

4.2.2 Program 2

Whereas in Program 1 we only required a fixed ordering for filling the entries of the matrix, in this program, we will be filling the matrix an entire column at a time. Each column is therefore set to an element of $\binom{v}{k}$, k-subsets of a set of size v. That is, the elements of the set represent the entries of the column that are set to 1. We first state an initial version of our column-by-column program. Then, we improve it by first bringing the condition testing to the internal nodes of the search tree and then by bringing isomorphism testing to the internal nodes too. The initial version of this program is found in Algorithm 6. Note that since we are assuming that we are setting the columns of $\mathbf{H}$ to elements of $\binom{v}{k}$, we do not need to test that each column sums to k.

We now generalize the condition that the inner product of any pair of rows of an acceptable completed matrix is λ so that it can be tested at the internal nodes of the search tree. That is, we need a method DotProductsPass(n, A) to test for condition 3' of Property 4.1.1, when A is a matrix with only n columns filled. When completing the matrix column by column, we need to check that for any pair of rows, it is still possible for them to be completed so as to have inner product equal to λ. Since the number of 1's in each row of an acceptable completed matrix is r, this is not possible for a pair of rows A^i and $A^{i'}$ if:

1. the number of columns j such that $A_{(i,j)} = 1$ and $A_{(i',j)} = 1$ already exceeds λ,
2. the number of columns j such that $A_{(i,j)} = 1$ and $A_{(i',j)} = 0$ exceeds $r - \lambda$,
3. the number of columns j such that $A_{(i,j)} = 0$ and $A_{(i',j)} = 1$ exceeds $r - \lambda$, or
4. the number of columns j such that $A_{(i,j)} = 0$ and $A_{(i',j)} = 0$ exceeds $b - (2r - \lambda)$.

Therefore, our method DotProductsPass should be as shown in Algorithm 7.

Algorithm 7 Test row intersections (final version).

```
1: function DotProductsPass($n, A$)
2:     For each $\{i, i'\} \in \binom{v}{2}$ do
3:         if $\sum_{j=0}^{n-1} A_{(i,j)}A_{(i',j)} > \lambda$ then return false.
4:         if $\sum_{j=0}^{n-1}(1 - A_{(i,j)})A_{(i',j)} > r - \lambda$ then return false.
5:         if $\sum_{j=0}^{n-1} A_{(i,j)}(1 - A_{(i',j)}) > r - \lambda$ then return false.
6:         if $\sum_{j=0}^{n-1}(1 - A_{(i,j)})(1 - A_{(i',j)}) > b - (2r - \lambda)$ then return false.
7:     return true.
8: end function
```

Generalizing clique testing so that it can be performed at the internal nodes of the search tree is also greatly simplified when compared to the modification that would be needed when filling the entries in arbitrary order. Suppose that A is a matrix with n columns filled, and that A' is a completion of A to b columns, the upper left $n \times n$ submatrix of $\psi(A')$ is determined by the inner products of the first n columns of A' and any clique of size greater than λ in this

Algorithm 6 Program 2 (initial version).

Prerequisites:

- A maximum clique-testing algorithm, *MaxClique*.
- A canonical form Ψ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
- A data-structure to hold solutions, $\mathfrak{S}$.

```
1:  function NodeConditionsPass($A$)
2:      if not RowSumsPass($A$) then return false.
3:      return true.
4:  end function

5:  function LeafConditionsPass($A$)
6:      For each $\{i, i'\} \in \binom{\bar{v}}{2}$ do
7:          if not $\sum_{j=0}^{b-1} A_{(i,j)} A_{(i',j)} = \lambda$ then return false.
8:      if $MaxClique(\phi(A)) > \lambda$ then return false.
9:      return true.
10: end function.

11: function Recurse($n$)
12:     if not NodeConditionsPass($\mathbf{H}$) then return
13:     if $n = b$ then
14:         if not LeafConditionsPass($\mathbf{H}$) then return
15:         For each $A \in \mathfrak{S}$ do
16:             if $\Psi(\mathbf{H}) = A$ then return
17:         Copy $\Psi(\mathbf{H})$ into $\mathfrak{S}$
18:     For each $\vec{b} \in \binom{\bar{v}}{k}$ do
19:         $\mathbf{H}_n \leftarrow \vec{b}$
20:         Recurse(n+1)
21:     $\mathbf{H}_n \leftarrow ?? \ldots ??$
22: end function
```

submatrix is also a clique in $\psi(A')$. But, this submatrix can already be determined from the completed columns of A. That is, if we define an $n \times n$ matrix

$$\phi_n(A)_{(j,j')} = \begin{cases} 0 & \text{if } A_j \cdot A_{j'} < t \text{ and} \\ 1 & \text{if } A_j \cdot A_{j'} \geq t, \end{cases}$$

then any clique of size greater than λ in $\phi_n(A)$ would appear in $\phi(A')$ for any completion A' of A. Therefore, we do not need to continue completing A. This gives us Algorithm 8 as the final version of NodeConditionsPass which will be used in all subsequent programs.

Algorithm 8 Test Node Conditions (final version).

```
1: function NodeConditionsPass($n, A$)
2:     if not RowSumsPass($A$) then return false.
3:     if not DotProductsPass($n, A$) then return false.
4:     if MaxClique($\phi_n(A)$) > $\lambda$ then return false.
5:     return true.
6: end function
```

Now recall that isomorph rejection is still only being performed at the leaves of the search tree. Indeed, thus far we *only* have a canonical form for completed matrices, Ψ. To remedy this, we will require more. For precision, we need some additional definitions, but we will immediately simplify the notation afterwards:

Definition 4.2.1. For $n \in [0, b]$, define:

$$\mathcal{M}_{v,b}^{n} = \{v \times b \text{ matrices with entries from } \{0,1\} \text{ in columns } [0, n-1] \text{ and "?" otherwise}\}.$$

The elements of $M_{v,b}^{n}$ will be called matrices *truncated* to n columns.

We also need a way to represent truncations of matrices.

Definition 4.2.2. For $m, n \in [0, b]$ where $m \geq n$ and $M \in \mathcal{M}_{v,b}^{m}$, define $\pi_n^m(M)$ to be the matrix in $\mathcal{M}_{v,b}^{n}$ obtained from M with all entries from columns $[n, m-1]$ substituted with "?". The matrix $\pi_n^m(M)$ will also be called M *truncated* to n columns and M will be called a *completion* of $\pi_n^m(M)$ to m columns.

Note that $\mathcal{M}_{v,b}^{0}$, contains only the matrix with all entries "?", and $\pi_0^m(M)$ is this matrix for any M and $m \in [0, b]$. Also $\mathcal{M}_{v,b}^{b} = \mathcal{M}_{v,b}$. Throughout this section, we will make the following notational conventions for simplicity:

1. For any n, the subscripts v and b on $\mathcal{M}_{v,b}^{n}$ will usually be dropped.

2. Although the functions π_n^m had a superscript to represent the domain of the function, we will simply write π_n and the domain can be inferred from the context.
3. For any n, and $M \in \mathcal{M}_{v,b}^n$, $\|M\|$ will represent the row major lexicographical value with respect to the function $f : \{0, 1, ?\} \to \{0, 1, 2\}$ and the permutations fixing columns beyond the n^{th}. This was defined in Definition 2.1.15.

Finally, we will make use of a sequence of "pruning functions".

Definition 4.2.3. Given $n \in [0, b]$, let U be an arbitrary set and $\psi_n : \mathcal{M}_{v,b}^n \to U$. The function ψ_n is a *pruning function* for level n if it satisfies:

For any A, B in $\mathcal{M}_{v,b}^n$, if $\psi_n(A) = \psi_n(B)$ and $A' \in \mathcal{M}_{v,b}$ such that $\pi_n(A') = A$, then there exists $B' \in \mathcal{M}_{v,b}$ such that $\pi_n(B') = B$ and $A' \cong B'$.

In other words, ψ_n is a "pruning function" for level n if it only maps two matrices truncated to n columns to the same value if they can be completed to form isomorphic matrices. Despite the notational difficulty, the modification to the program and its justification are fairly intuitive. This is the modification and its motivation in point form:

- The current program recurses on all partial matrices that might be completed to yield a $(\lambda, 2)$-disjoint design.
- For each $n \in [0, b]$ we have a pruning function that, given two partial matrices $A, B \in \mathcal{M}^n$, tells us whether any completion of A is isomorphic to a completion of B.
- Modify the program so that if any completion of a partially completed matrix A must be isomorphic to a completion of partially completed matrix B that was already recursed upon, then A is not recursed upon.

Note that this implies recording the value of the pruning function already seen in some way, for example, keeping a representative for each value. We do so by introducing a data structure $\mathfrak{S}_n$ at each level for storage. Also note that we can always select ψ_n to be the identity mapping on $\mathcal{M}^n$; any completion of a partially complete matrix is obviously isomorphic to itself.

A toy example of a pruning function for level n is as follows: arbitrarily select a matrix $A \in \mathcal{M}^n$ for some $n \in [0, v]$. Let B be the matrix formed from A by interchanging the first two columns of A. Define $\psi_n : \mathcal{M}^n \to \mathcal{M}^n \cup \{\emptyset\}$:

$$\psi_n(M) = \begin{cases} \emptyset & \text{if } M = A \text{ or } M = B, \text{ and} \\ M & \text{otherwise.} \end{cases}$$

Then, if $\psi_n(M) = \psi_n(N)$, either:

1. Neither M nor N are equal to A or B, in which case $\psi_m(M) = \psi_m(N)$ implies $M = N$. So, any completion of M *is* a completion of N.
2. $M = A$ and $N = B$ or vice versa, in which case any completion of M can be formed from a completion of N by interchanging the first two collmns.
3. $M = N = A$ or $M = N = B$, and, again, any completion of M *is* a completion of N.

We will return to our choice of mappings ψ_n later in this section, but now we modify the program and prove its correctness using Definition 4.2.3. The modified program is displayed in Algorithm 9.

Moving isomorphism testing to the internal nodes of the search tree by means of the pruning functions ψ_n is a relatively significant change that warrants proving the correctness of the program. We need to show that $\mathfrak{S}$ contains all possible canonical forms of elements of $\mathcal{M}$ which satisfy the conditions of being a Katona sieve.

Claim 4.2.1. For any matrix $M \in \mathcal{M}$ that can represent a Katona sieve, the preceeding program generates a matrix $M' \in \mathcal{M}$ such that $M \cong M'$ and $\Psi(M') \in \mathfrak{S}$.

Proof. We will argue by induction. But first, we explicitly state the key property of NodeConditionsPass(n, A):

If A is a Katona sieve, then for any $n \in [0, b]$, NodeConditionsPass$(n, \pi_n(A))$ is true.

The correctness of this was explained earlier when defining NodeConditionsPass. We also use the fact that if $A, B \in \mathcal{M}$ and $A \cong B$, then if A is $(\lambda, 2)$-disjoint, so is B. Now, we can proceed with the induction.

First, note that if $\psi_b(\pi_b(A')) \in \mathfrak{S}_b$, then $\Psi(A') \in \mathfrak{S}$. Indeed, if $\psi_b(\pi_b(A')) \in \mathfrak{S}_b$ then a matrix $B \in \mathcal{M}^b = \mathcal{M}$ such that $\psi_b(B) = \psi_b(\pi_b(A'))$ was generated and recursed upon. But $B' = B$ is the unique matrix such that $B = \pi_b(B')$ and A' is the unique matrix such that $A' = \pi_b(A')$. Since $\psi_b(\pi_b(B')) = \psi_b(\pi_b(A'))$, it must be the case that $B' \cong A'$. Since A' is $(\lambda, 2)$-disjoint, so is B', and $\Psi(B') = \Psi(A')$ was added to $\mathfrak{S}$.

Suppose that a matrix $A'_0 \in \mathcal{M}$ is $(\lambda, 2)$-disjoint and $\Psi(A'_0) \notin \mathfrak{S}$. Let w_0 be the greatest index in $[0, b]$ such that $\psi_{w_0}(\pi_{w_0}(A'_0)) \in \mathfrak{S}_{w_0}$. Then, $\psi_{w_0+1}(\pi_{w_0+1}(A'_0)) \notin \mathfrak{S}_{w_0+1}$. So it must be the case that a matrix $A_1 \in \mathcal{M}^{w_0+1}$ was previously generated with $\psi_{w_0+1}(A_1) = \psi_{w_0+1}(\pi_{w_0+1}(A'))$. By the definition of a pruning function, there must be $A'_1 \in \mathcal{M}$ such that $A'_1 \cong A'_0$ and $\pi_{w_0+1}(A'_1) = A_1$. Since A'_1 is isomorphic to A'_0, A'_1 is also $(\lambda, 2)$-disjoint and for any $n \in [0, b]$, NodeConditionsPass$(n, \pi_n(A'_1))$ is true. If w_1 is the greatest index such that $\psi_{w_1}(\pi_{w_1}(A'_1)) \in \mathfrak{S}_{w_1}$ then and w_1 is strictly greater than w_0. This argument can be repeated with A'_1 to yield w_2 and $A'_2 \in \mathcal{M}^{w_2}$ such that $w_2 > w_1 > w_0$, $\psi_{w_2}(\pi_{w_2}) \in \mathfrak{S}_{w_2}$, and $A'_2 \cong A'_1 \cong A'_0$. Continuing in this way, eventually w_ℓ will be equal to b and we will have a matrix $A'_\ell \cong A'$ that

Algorithm 9 Program 2 (second version).

Prerequisites:

- A maximum clique-testing algorithm, *MaxClique*.
- A canonical form Ψ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
- A data-structure to hold solutions, $\mathfrak{S}$.
- Pruning functions ψ_n for $n \in [0, b]$ satisfying Definition 4.2.3.
- A sequence of data-structures to hold partial solutions, $\mathfrak{S}_n$, for $n \in [0, b]$.

```
 1: function NodePruningPasses($n, A$)
 2:     For each $M \in \mathfrak{S}_n$ do
 3:         if $\psi_n(A) = M$ then return false.
 4:     Copy $\psi_n(A)$ into $\mathfrak{S}_n$
 5:     return true.
 6: end function

 7: function Recurse($n$)
 8:     if not NodeConditionsPass($n, \mathbf{H}$) then return
 9:     if not NodePruningPasses($n, \mathbf{H}$) then return
10:     if $n = b$ then
11:         For each $A \in \mathfrak{S}$ do
12:             if $\Psi(\mathbf{H}) = A$ then return
13:         Copy $\Psi(\mathbf{H})$ into $\mathfrak{S}$
14:     For each $\vec{b} \in \binom{\bar{v}}{k}$ do
15:         $\mathbf{H}_n \leftarrow \vec{b}$
16:         Recurse(n+1)
17:     $\mathbf{H}_n \leftarrow ??\ldots??$
18: end function
```

was generated, and such that $\psi_b(\pi_b(A'_\ell)) \in \mathfrak{S}_b$. But this implies $\Psi(A'_\ell) = \Psi(A'_0)$ was added to $\mathfrak{S}$. $\square$

That only one matrix is generated isomorphic to a given matrix by the program is ensured by the final isomorphism test at level $n = b$. Since we have not defined the mappings ψ_n for $n \in [0, b]$ we essentially have a template for a program. The pruning functions that we would like to use are canonical forms with respect to $G_n = \{(\rho, \sigma) \in \mathcal{S}_v \times \mathcal{S}_b; j \geq n \Rightarrow \sigma j = j\}$, the permutations fixing all but the first n columns. So, for example we can define:

$$\psi_n(M) = \rho' M \sigma' \text{ such that } \|\rho' M \sigma'\| = \max_{(\rho,\sigma)\in G_n} \|\rho M \sigma\|.$$

We need to verify that this choice of ψ_n for $n \in [0, b]$ satisfies the definition of a pruning function. Suppose that $A, B \in \mathcal{M}^n$ such that $\psi_n(A) = \psi_n(B)$ and $A' \in \mathcal{M}$ is a completion of A. That is, $A = \pi_n(A')$. Since $\psi_n(A) = \psi_n(B)$, by the definition of ψ_n, there are permutations $(\rho_A, \sigma_A), (\rho_B, \sigma_B) \in G_n$ such that:

$$\rho_A A \sigma_A = \psi_n(A) = \psi_n(B) = \rho_B B \sigma_B.$$

By the definition of G_n, $\sigma_A \sigma_B^{-1}$ maps columns $[0, b-1]$ to columns $[0, b-1]$ and columns $[n, b-1]$ to columns $[n, b-1]$. But this means that:

$$\pi_n(\rho_B^{-1} \rho_A A' \sigma_A \sigma_B^{-1}) = \pi_n(\rho_B^{-1} \rho_A A') \sigma_A \sigma_B^{-1}.$$

By its definition π_n commutes with any permutation of rows. So,

$$\begin{aligned}
\pi_n(\rho_B^{-1} \rho_A A' \sigma_A \sigma_B^{-1}) &= \pi_n(\rho_B^{-1} \rho_A A') \sigma_A \sigma_B^{-1} \\
&= \rho_B^{-1} \rho_A \pi_n(A') \sigma_A \sigma_B^{-1} \\
&= \rho_B^{-1} \rho_A A \sigma_A \sigma_B^{-1} \\
&= B.
\end{aligned}$$

So for any $A, B \in \mathcal{M}^n$ such that $\psi_n(A) = \psi_n(B)$, and $A' \in \mathcal{M}$ such that $A = \pi_n(A')$ there exists a $B' \in \mathcal{M}$ such that $\pi_n(B') = B$ and $B' \cong A'$. Namely,

$$B' = \rho_B^{-1} \rho_A A' \sigma_A \sigma_B^{-1},$$

where $\rho_A A \sigma_A = \psi_n(A) = \psi_n(B) = \rho_B B \sigma_B$.

In this verification, the only property of G_n that was used is that each element of G_n commutes with π_n. As such, any subgroup of G_n could also be used to define ψ_n. For example, the subgroup of G_n that fixes the rows, $\{(e, \sigma) \in G_n; e \text{ is the identity permutation of } \mathcal{S}_v\}$, could have been used. The canonical form with respect to this subgroup is the matrix obtained by sorting the columns in lexicographical order.

It is important to note that for $n \in [0, b]$, ψ_n can be defined separately for *each* n. When n is larger, that is, when more columns are filled, it may be costly to compute a canonical form. Also, the number of different isomorphism classes may be quite large for some values of n, so using a canonical form as our pruning function may not always be appropriate. For this reason, we may define ψ_n to be the identity for some values of n and avoid this cost. For example, we can choose ψ_n as a canonical form over G_n for small values of n and the identity for larger values of n. If $I \in [0, b]$ is the level to which we wish to use a canonical form for ψ_n, then we can state our final selection of the sequence ψ_n.

Let $G_n = \{(\rho, \sigma) \in \mathcal{S}_v \times \mathcal{S}_b; j \geq n \Rightarrow \sigma j = j\}$. Then define:

$$\psi_n(M) = \begin{cases} \rho' M \sigma' \text{ such that } \|\rho' M \sigma'\| = \max_{(\rho,\sigma)\in G_n} \|\rho M \sigma\| & \text{if } 0 \leq n \leq I, \text{ and} \\ M & \text{if } I < n \leq b. \end{cases} \tag{5}$$

Note that choosing ψ_n to be the identity means that at level n, no pruning of the search tree is being performed at all. That is, any partial matrix generated at level n is recursed upon. For this reason, if ψ_n is the identity, it is not necessary to store the values in $\mathfrak{S}_n$ at all unless the partial matrices are otherwise needed. The final version of this program is thus presented in Algorithm 10.

4.2.3 Program 3

In this section we will make only a minor modication to Program 2. This program was implemented and its performance will be compared to Program 4 in Section 4.3. We still complete the matrix column by column, but *first* we set the values of the first two rows to be elements of $\binom{b}{r}$ whose inner product is λ. We then proceed to fill the remaining entries of each column in order so that each column contains k 1's.

Since any acceptable way to complete the first two rows is isomorphic to any other through a permutation of the columns, we can simply start at level $n = 0$ with a specific selection for these two rows. In particular, we begin with the matrix $\mathbf{H}$ in the form shown in Figure 9.

The column by column generation is then modified as follows:

1. Fill the remaining entries of the first $r - \lambda$ columns with $(k-1)$-subsets of $[2, v-1]$.
2. Fill the remaining entries in the next λ columns with $(k-2)$-subsets of $[2, v-1]$.
3. Fill the remaining entries of the next $r - \lambda$ columns with $(k-1)$-subsets of $[2, v-1]$.
4. Finally, the remaining entries in the last $b - 2r + \lambda$ with k-subsets of $[2, v-1]$.

We restate the program with this modification in Algorithm 11.

Algorithm 10 Program 2 (final version).

Prerequisites:

- A maximum clique-testing algorithm, *MaxClique*.
- A canonical form Ψ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
- A data-structure to hold solutions,$\mathfrak{S}$.
- Pruning functions ψ_n for $n \in [0, b]$ as in Equation 5 on Page 5.
- A sequence of data-structures to hold partial solutions, $\mathfrak{S}_n$, for $n \in [0, b]$.

```
1:  function NodePruningPasses($n, A$)
2:      For each $M \in \mathfrak{S}_n$ do
3:          if $\psi_n(A) = M$ then return false.
4:      Copy $\psi_n(A)$ into $\mathfrak{S}_n$
5:      return true.
6:  end function

7:  function Recurse($n$)
8:      if not NodeConditionsPass($n, \mathbf{H}$) then return
9:      if not NodePruningPasses($n, \mathbf{H}$) then return
10:     if $n = b$ then
11:         For each $A \in \mathfrak{S}$ do
12:             if $\Psi(\mathbf{H}) = A$ then return
13:         Copy $\Psi(\mathbf{H})$ into $\mathfrak{S}$
14:     For each $\vec{b} \in \binom{\bar{v}}{k}$ do
15:         $\mathbf{H}_n \leftarrow \vec{b}$
16:         Recurse(n+1)
17:     $\mathbf{H}_n \leftarrow ??\ldots??$
18: end function
```

Algorithm 11 Program 3 (final version).

```
    **Prerequisites:**
        • A maximum clique-testing algorithm, *MaxClique*.
        • A canonical form $\Psi$ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
        • A data-structure to hold solutions, $\mathfrak{S}$.
        • Pruning functions $\psi_n$ for $n \in [0, b]$ as in Equation 5 on Page 50.
        • A sequence of data-structures to hold partial solutions, $\mathfrak{S}_n$, for $n \in [0, b]$.

function NodePruningPasses($n, A$)
    For each $M \in \mathfrak{S}_n$ do
        if $\psi_n(A) = M$ then return false.
    Copy $\psi_n(A)$ into $\mathfrak{S}_n$
    return true.
end function

function Recurse($n$)
    if not NodeConditionsPass($n, \mathbf{H}$) then return
    if not NodePruningPasses($n, \mathbf{H}$) then return
    if $n = b$ then
        For each $A \in \mathfrak{S}$ do
            if $\Psi(\mathbf{H}) = A$ then return
        Copy $\Psi(\mathbf{H})$ into $\mathfrak{S}$
    if $n < r - \lambda$ then
        For each $\vec{b} \in \binom{\overline{v-2}}{k-1}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            Recurse(n+1)
    if $r - \lambda \leq n < r$ then
        For each $\vec{b} \in \binom{\overline{v-2}}{k-2}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            Recurse(n+1)
    if $r \leq n < 2r - \lambda$ then
        For each $\vec{b} \in \binom{\overline{v-2}}{k-1}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            Recurse(n+1)
    if $2r - \lambda \leq n < b$ then
        For each $\vec{b} \in \binom{\overline{v-2}}{k}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            Recurse(n+1)
    $\mathbf{H}_n \leftarrow ?? \ldots ??$
end function
```

$$
\mathbf{H} = \begin{array}{c}
\begin{array}{cccc}
\overbrace{\hphantom{1\ 1\ \dots\ 1}}^{r-\lambda} & \overbrace{\hphantom{1\ 1\ \dots\ 1}}^{\lambda} & \overbrace{\hphantom{0\ 0\ \dots\ 0}}^{r-\lambda} & \overbrace{\hphantom{0\ 0\ \dots\ 0}}^{b-2r+\lambda}
\end{array} \\
\left[\begin{array}{cccccccccccccccc}
1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\
0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 & 0 & 0 & \dots & 0 \\
? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ? \\
? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ? \\
? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ? \\
? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ? \\
? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ? & ? & ? & \dots & ?
\end{array}\right]
\end{array} .
$$

Figure 9: Completion of first two rows

4.2.4 Program 4

Filling the matrix in the manner of Program 3, beginning with the first two rows completed, opens the door to another possibility. This program was implemented and will be compared to Program 3 in Section 4.3. For illustration, we will use the construction of a 2-design with parameters $v = 7$, $k = 3$, $b = 14$, $r = 6$, and $\lambda = 2$. Suppose that with $n = r$ columns completed, the procedure begins with:

$$
\mathbf{H} = \left[\begin{array}{cccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & ? & ? & ? & ? & ? & ? & ? & ? \\
1 & 0 & 0 & 1 & 0 & 0 & ? & ? & ? & ? & ? & ? & ? & ? \\
1 & 1 & 0 & 0 & 0 & 0 & ? & ? & ? & ? & ? & ? & ? & ? \\
0 & 1 & 1 & 0 & 0 & 0 & ? & ? & ? & ? & ? & ? & ? & ? \\
0 & 0 & 1 & 1 & 0 & 0 & ? & ? & ? & ? & ? & ? & ? & ?
\end{array}\right] .
$$

After some searching, the program might reach this matrix at level $n = 2r - \lambda$:

$$
\mathbf{H} = \left[\begin{array}{cccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & ? & ? & ? & ? \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & ? & ? & ? & ? \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & ? & ? & ? & ? \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & ? & ? & ? & ? \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & ? & ? & ? & ?
\end{array}\right] .
$$

Notice the reappearance of the section indicated with dotted lines.

Notation: Let us use Σ_1, Λ, Σ_2, and Ω to denote the various sections of the matrix $\mathbf{H}$ as follows:

$$\mathbf{H} = \left[\begin{array}{cccc} \overbrace{1\ 1\ \dots\ 1}^{r-\lambda} & \overbrace{1\ 1\ \dots\ 1}^{\lambda} & \overbrace{0\ 0\ \dots\ 0}^{r-\lambda} & \overbrace{0\ 0\ \dots\ 0}^{b-2r+\lambda} \\ 0\ 0\ \dots\ 0 & 1\ 1\ \dots\ 1 & 1\ 1\ \dots\ 1 & 0\ 0\ \dots\ 0 \\ \Sigma_1(\mathbf{H}) & \Lambda(\mathbf{H}) & \Sigma_2(\mathbf{H}) & \Omega(\mathbf{H}) \end{array}\right].$$

The preceding example illustrates that for any valid completion of the matrix up to level $n = 2r - \lambda$:

$$\mathbf{H} = \left[\begin{array}{cccc} \overbrace{1\ 1\ \dots\ 1}^{r-\lambda} & \overbrace{1\ 1\ \dots\ 1}^{\lambda} & \overbrace{0\ 0\ \dots\ 0}^{r-\lambda} & \overbrace{0\ 0\ \dots\ 0}^{b-2r+\lambda} \\ 0\ 0\ \dots\ 0 & 1\ 1\ \dots\ 1 & 1\ 1\ \dots\ 1 & 0\ 0\ \dots\ 0 \\ & & & ?\ ?\ \dots\ ? \\ & & & ?\ ?\ \dots\ ? \\ \Sigma_1(\mathbf{H}) & \Lambda(\mathbf{H}) & \Sigma_2(\mathbf{H}) & ?\ ?\ \dots\ ? \\ & & & ?\ ?\ \dots\ ? \\ & & & ?\ ?\ \dots\ ? \end{array}\right],$$

the following matrix with $\Sigma_2(\mathbf{H})$ occupying the first columns must be a valid completion of r columns:

$$\mathbf{H}' = \left[\begin{array}{cccc} \overbrace{1\ 1\ \dots\ 1}^{r-\lambda} & \overbrace{1\ 1\ \dots\ 1}^{\lambda} & \overbrace{0\ 0\ \dots\ 0}^{r-\lambda} & \overbrace{0\ 0\ \dots\ 0}^{b-2r+\lambda} \\ 0\ 0\ \dots\ 0 & 1\ 1\ \dots\ 1 & 1\ 1\ \dots\ 1 & 0\ 0\ \dots\ 0 \\ & & ?\ ?\ ?\ ? & ?\ ?\ \dots\ ? \\ & & ?\ ?\ ?\ ? & ?\ ?\ \dots\ ? \\ \Sigma_2(\mathbf{H}) & \Lambda(\mathbf{H}) & ?\ ?\ ?\ ? & ?\ ?\ \dots\ ? \\ & & ?\ ?\ ?\ ? & ?\ ?\ \dots\ ? \\ & & ?\ ?\ ?\ ? & ?\ ?\ \dots\ ? \end{array}\right].$$

We will actually be using this observation in the reverse order. When the program has filled r columns of $\mathbf{H}$, we will fill the columns $[r, 2r - \lambda - 1]$ with previously seen $\Sigma_1(A)$, where A is a matrix the program generated with $r - \lambda$ columns completed. In order to keep things readable, we need to be able to represent a matrix constructed from putting together or concatenating smaller matrices together. In particular, we need the following notation:

Definition 4.2.4. If A is a $(v-2) \times x$ matrix, B is a $(v-2) \times y$ matrix, and $\mathbf{H}$ is the matrix of Figure 9 on Page 53, then we define $A|B$ to be the matrix:

$$(A|B)_{(i,j)} = \begin{cases} \mathbf{H}_{(i,j)} & \text{if } 0 \leq i < 2, \\ A_{(i-2,j)} & \text{if } 2 \leq i < v \text{ and } 0 \leq j < x, \text{ and} \\ B_{(i-2,j-x)} & \text{if } 2 \leq i < v \text{ and } a \leq j < x+y \; . \end{cases}$$

In other words, "|" concatenates the matrices A and B into an a $(v-2) \times (x+y)$ matrix and then inserts it into the left-hand side of the "?" portion of Figure 9. For example, if A is a $(v-2) \times (r-\lambda)$ matrix and B is a $(v-2) \times \lambda$ matrix, then

$$A|B = \begin{array}{cccc}
\overbrace{\hphantom{1\ 1\ \dots\ 1}}^{r-\lambda} & \overbrace{\hphantom{1\ 1\ \dots\ 1}}^{\lambda} & \overbrace{\hphantom{0\ 0\ \dots\ 0}}^{r-\lambda} & \overbrace{\hphantom{0\ 0\ \dots\ 0}}^{b-2r+\lambda} \\
1\ 1\ \dots\ 1 & 1\ 1\ \dots\ 1 & 0\ 0\ \dots\ 0 & 0\ 0\ \dots\ 0 \\
0\ 0\ \dots\ 0 & 1\ 1\ \dots\ 1 & 1\ 1\ \dots\ 1 & 0\ 0\ \dots\ 0 \\
 & & ?\ ?\ \ ?\ \ ? & ?\ ?\ \dots\ ? \\
 & & ?\ ?\ \ ?\ \ ? & ?\ ?\ \dots\ ? \\
A & B & ?\ ?\ \ ?\ \ ? & ?\ ?\ \dots\ ? \\
 & & ?\ ?\ \ ?\ \ ? & ?\ ?\ \dots\ ? \\
 & & ?\ ?\ \ ?\ \ ? & ?\ ?\ \dots\ ?
\end{array} .$$

We also extend this definition slightly using a matrix [?] consisting of all ? of the implied size. That is, if A is a $(v-2) \times x$ matrix and we want to indicate the $v \times b$ matrix with only the columns of A included in the submatrix of columns $[0, x-1]$, we will write $A|[?]$ as though [?] was a matrix with $v-2$ rows of all ?'s.

Now, we can introduce our strategy. For the moment, we will assume that isomorphism testing was not performed when filling the first $r-\lambda$ columns and that we are storing *every* acceptable matrix $\mathbf{H}$ generated at level $r-\lambda$ in $\mathfrak{S}_{r-\lambda}$. So, with $G_n = \{(\rho, \sigma) \in \mathcal{S}_v \times \mathcal{S}_b; j \geq n \Rightarrow \sigma j = j\}$ and I the level at which we stop performing partial isomorphism pruning, we are choosing the pruning function:

$$\psi_n(M) = \begin{cases} M & \text{if } n < r-\lambda, \\ \rho' M \sigma' \text{ such that } \|\rho' M \sigma'\| = \max_{(\rho,\sigma) \in G_n} \|\rho M \sigma\| & \text{if } r-\lambda \leq n \leq I, \text{ and} \\ M & \text{if } I < n \leq b. \end{cases} \qquad (6)$$

By choosing ψ_n as the identity for $n < r-\lambda$, we are not performing isomorphism pruning at all for these levels *and* we are storing every matrix generated in $\mathfrak{S}_n$. With this choice, we begin with initial program presented in Algorithm 12.

Algorithm 12 Program 4 (initial version).

```
Prerequisites:
  • A maximum clique-testing algorithm, MaxClique.
  • A canonical form $\Psi$ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
  • A data-structure to hold solutions, $\mathfrak{S}$.
  • Pruning functions $\psi_n$ for $n \in [0, b]$ as in Equation 6 on Page 55.
  • A sequence of data-structures to hold partial solutions, $\mathfrak{S}_n$, for $n \in [0, b]$.

function NodePruningPasses($n, A$)
    For each $M \in \mathfrak{S}_n$ do
        if $\psi_n(A) = M$ then return false.
    Copy $\psi_n(A)$ into $\mathfrak{S}_n$
    return true.
end function

function RecursePhase3($n$)
    if not NodeConditionsPass($n, \mathbf{H}$) then return
    if not NodePruningPasses($n, \mathbf{H}$) then return
    if $n = b$ then
        For each $A \in \mathfrak{S}$ do
            if $\Psi(\mathbf{H}) = A$ then return
        Copy $\Psi(\mathbf{H})$ into $\mathfrak{S}$
    if $2r - \lambda \leq n < b$ then
        For each $\vec{b} \in \binom{v-2}{k}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            RecursePhase3(n+1)
    $\mathbf{H}_n \leftarrow ?? \ldots ??$
end function
```

Algorithm 12 Program 4 (initial version, continued).

```
function FillPhase2
    For each $A \in \mathfrak{S}_{r-\lambda}$ do
        $\mathbf{H} \leftarrow \Sigma_1(\mathbf{H})|\Lambda(\mathbf{H})|\Sigma_1(A)$
        RecursePhase3($2r-\lambda$)
    $\mathbf{H} \leftarrow \Sigma_1(\mathbf{H})|\Lambda(\mathbf{H})|[?]$
end function

function RecursePhase1($n$)
    if not NodeConditionsPass($n, \mathbf{H}$) then return
    if not NodePruningPasses($n, \mathbf{H}$) then return
    if $n = r$ then
        FillPhase2
    if $n < r - \lambda$ then
        For each $\vec{b} \in \overline{\binom{v-2}{k-1}}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            RecursePhase1(n+1)
    if $r - \lambda \leq n < r$ then
        For each $\vec{b} \in \overline{\binom{v-2}{k-2}}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            RecursePhase1(n+1)
    $\mathbf{H}_n \leftarrow ??\ldots??$
end function
```

We repeat the differences between this initial version of Program 4 and the final version of Program 3:

1. The recursion filling columns $[0, r-1]$ and columns $[2r-\lambda, b-1]$ remains unchanged, but has been separated by the function FILLPHASE2(). FILLPHASE2() completes columns $[r, 2r-\lambda-1]$ of the matrix $\mathbf{H}$ with the previous completions of columns $[0, r-\lambda-1]$.
2. The isomorphs are not being pruned when columns $[0, r-\lambda-1]$ are being completed.

Foregoing isomorphism testing when completing columns $[0, r-\lambda-1]$ is not feasible in practice but is required to make this initial program correct. From the previous discussion, it is clear that any completion of $\Sigma_2(\mathbf{H})$ is contained in $\mathfrak{S}_{r-\lambda}$ when the program completes, since this will contains all possible completions of $\Sigma_1(\mathbf{H})$. The only potential issue is that the program only uses *previously generated* members of $\mathfrak{S}_{r-\lambda}$. However, this is justified because for any matrix $\mathbf{H}$ with $2r-\lambda$ rows completed, one of the matrices $\Sigma_1(\mathbf{H})|[?]$ or $\Sigma_2(\mathbf{H})|[?]$ was generated first or they are equal. So, considering only previously generated members would lead to the generation of either $\mathbf{H}$ or $\Sigma_2(\mathbf{H})|\Lambda(\mathbf{H})|\Sigma_1(\mathbf{H})$, which are isomorphic through a column permutation.

$$D_0 = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&?&?&?&?&?&?&?&?&?&? \\ 1&1&0&0&?&?&?&?&?&?&?&?&?&? \\ 1&1&0&0&?&?&?&?&?&?&?&?&?&? \\ 0&0&1&1&?&?&?&?&?&?&?&?&?&? \\ 0&0&1&1&?&?&?&?&?&?&?&?&?&? \end{bmatrix}.$$

$$D_1 = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&?&?&?&?&?&?&?&?&?&? \\ 1&1&0&0&?&?&?&?&?&?&?&?&?&? \\ 0&0&1&1&?&?&?&?&?&?&?&?&?&? \\ 1&1&0&0&?&?&?&?&?&?&?&?&?&? \\ 0&0&1&1&?&?&?&?&?&?&?&?&?&? \end{bmatrix}.$$

Figure 10: Two isomorphic completions of $r-\lambda$ columns

We would like to reintroduce isomorphism pruning when completing the first $r-\lambda$ columns with respect to the permutations fixing the first two rows and begin by explaining why it would have been incorrect to do this pruning without more changes. Any permutation mapping one $(v-2)\times(r-\lambda)$, A, to another, B, carries over to a permutation mapping $A|[?]$ to $B|[?]$. However, if C is a $(v-2)\times r$ matrix, a permutation mapping A to B will not generally carry over to a permutation mapping $C|A$ to $C|B$.

$$R|\Sigma_1(D_0) = \left[\begin{array}{cccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & ? & ? & ? & ? \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & ? & ? & ? & ? \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & ? & ? & ? & ? \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & ? & ? & ? & ? \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & ? & ? & ? & ?
\end{array}\right].$$

$$R|\Sigma_1(D_1) = \left[\begin{array}{cccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & ? & ? & ? & ? \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & ? & ? & ? & ? \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & ? & ? & ? & ? \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & ? & ? & ? & ? \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & ? & ? & ? & ?
\end{array}\right].$$

Figure 11: Two isomorphic completions of $r - \lambda$ columns appended to a matrix complete to r columns

To illustrate the problem, consider the matrices in Figure 10. These two are isomorphic since interchanging the fifth and sixth rows transforms one into the other. But, when we append $\Sigma_1(D_0)$ and $\Sigma_1(D_1)$ to the same matrix completed to level r, the resulting matrices are not necessarily isomorphic, as illustrated by appending these to a matrix R in Figure 11.

The reason that the two matrices fail to be isomorphic in the latter context is because interchanging the fifth and sixth rows is not an automorphism of the rest of the matrix; that is, there does not exist a column permutation that when combined with this row permutation fixes the rest of the matrix.

This shows that if we wish to fill the columns $[r, 2r - \lambda - 1]$ as a unit while still pruning isomorphs when filling the columns $[0, r - \lambda - 1]$, we need to recover some isomorphs that were eliminated. Now, let $\mathfrak{S}_{r-\lambda}$ be the set of matrices that would have been generated if we *had* performed isomorphism pruning while completing rows $[0, r - \lambda - 1]$. Note that it is not necessary to consider column permutations of these matrices since the columns $[r, 2r - \lambda - 1]$ are interchangeable for any partial matrix generated in this way. For a matrix A generated at level r, the possible completions to $2r - \lambda$ columns of a A must be contained in:

$$\{A|\Sigma_1(\rho M);\ M \in \mathfrak{S}_{r-\lambda},\ \rho \in \mathcal{S}_{[2,v-1]}\}.$$

Indeed, this recovers *all* isomorphs through a row permutation of elements of $\mathfrak{S}_{r-\lambda}$ and must therefore include at least one member of each isomorphism class of completions to $2r - \lambda$ of A. Returning to the example of Figures 10 and 11, if $\rho = (4\ 5)$ is the permutation interchanging the fifth and sixth rows, then we would have generated both $R|\Sigma_1(D_0)$ and $R|\Sigma_1(\rho D_0)$.

We state the changed functions in Algorithm 13 which includes isomorphism testing when filling the first $r-\lambda$ columns. With $G_n = \{(\rho,\sigma) \in \mathcal{S}_v \times \mathcal{S}_b; j \geq n \Rightarrow \sigma j = j\}$ and I the level at which we stop performing partial isomorphism pruning, we are returning to the pruning function:

$$\psi_n(M) = \begin{cases} \rho' M \sigma' \text{ such that } \|\rho' M \sigma'\| = \max_{(\rho,\sigma)\in G_n} \|\rho M \sigma\| & \text{if } n \leq I, \text{ and} \\ M & \text{if } I < n \leq b. \end{cases} \tag{7}$$

Algorithm 13 Program 4 (second version).

Prerequisites:

- A maximum clique-testing algorithm, *MaxClique*.
- A canonical form Ψ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
- A data-structure to hold solutions,$\mathfrak{S}$.
- Pruning functions ψ_n for $n \in [0,b]$ as in Equation 7 on Page 60.
- A sequence of data-structures to hold partial solutions, $\mathfrak{S}_n$, for $n \in [0,b]$.

```
1:  function NodePruningPasses($n, A$)
2:      For each $M \in \mathfrak{S}_n$ do
3:          if $\psi_n(A) = M$ then return false.
4:      Copy $\psi_n(A)$ into $\mathfrak{S}_n$
5:      return true.
6:  end function

7:  function FillPhase2
8:      For each $\rho \in \mathcal{S}_{[2,v-1]}$ do
9:          For each $A \in \mathfrak{S}_{r-\lambda}$ do
10:             $\mathbf{H} \leftarrow \Sigma_1(\mathbf{H})|\Lambda(\mathbf{H})|\Sigma_1(\rho A)$
11:             RecursePhase3($2r-\lambda$)
12:     $\mathbf{H} \leftarrow \Sigma_1(\mathbf{H})|\Lambda(\mathbf{H})|[?]$
13: end function
```

Despite restoring isomorphism testing when completing columns $[0, r-\lambda-1]$ this program is still not feasible in practice. The loop that we have introduced over all permutations of rows $[2, v-1]$ is very costly. The remainder of this discussion focuses on reducing this cost.

The principal way in which this will be reduced is based on the fact that the inner product of each of the rows $[2, v-1]$ with both the first and second rows must equal λ. Suppose that in FillPhase2 we are again considering the two matrices R and $\Sigma_1(D_0)$ from Figure 11. Now however, let ρ' be the permutation interchaging the third and fourth rows of the matrix being appended. Then the program generates the following:

$$R|\Sigma_1(\rho D_0) = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&1&1&1&1&0&0&?&?&?&? \\ 1&1&0&0&0&0&0&0&0&0&?&?&?&? \\ 1&1&0&0&0&0&1&1&0&0&?&?&?&? \\ 0&0&1&1&0&0&0&0&1&1&?&?&?&? \\ 0&0&1&1&0&0&0&0&1&1&?&?&?&? \end{bmatrix}.$$

The inner product of the third row with the second is not equal to λ. In fact, this will always be the case if the permutation ρ being applied to $\Sigma_1(A)$ in FILLPHASE2 does not map the rows of $\Sigma_1(A)$ to rows with an equal number of 1's in $\Sigma_1(\mathbf{H})$. For a valid $\mathbf{H}$ with $2r - \lambda$ columns completed, the number of 1's in each row of $\Sigma_1(\mathbf{H})$ must equal the number of 1's in the corresponding row of $\Sigma_2(\mathbf{H})$. If this was not the case for a given $\mathbf{H}$ and row i, then we would have:

$$\lambda = \mathbf{H}^i \cdot \mathbf{H}^0 = \sum_{j=0}^{\lambda-1} \mathbf{H}_{(i,j)} + \sum_{j=0}^{r-\lambda-1} \mathbf{H}_{(i,j)} \neq \sum_{j=0}^{\lambda-1} \mathbf{H}_{(i,j)} + \sum_{j=r}^{2r-\lambda-1} \mathbf{H}_{(i,j)} = \mathbf{H}^i \cdot \mathbf{H}^1 = \lambda.$$

So for each A in $\mathfrak{S}_{r-\lambda}$ we need only consider the permutations ρ that map a row i to ρi where the number of 1's in row i of $\Sigma_1(A)$ is equal to the number of 1's in row ρi of $\Sigma_1(\mathbf{H})$. As a result, if there is no such permutation, A itself should not be considered in FILLPHASE2.

The most convenient way to use this observation is to have rows $[2, v-1]$ of $\mathbf{H}$ sorted by the number of 1's in the corresponding row of $\Sigma_1(\mathbf{H})$ and to have each $A \in \mathfrak{S}_{r-\lambda}$ sorted similarly. Again, if the sequence of row sums of $\Sigma_1(\mathbf{H})$ is different then the sequence of sums for $\Sigma_1(A)$ for some $A \in \mathfrak{S}_{r-\lambda}$ we need not consider A for $\mathbf{H}$ in FILLPHASE2. If the sequence of row sums is the same, then the permutations that should be considered with A in FILLPHASE2 are exactly those permutations that map a row of $\Sigma_1(\mathbf{H})$ to another row with the same row sums. This leads to the following definitions:

Definition 4.2.5. Given A with $r-\lambda$ columns completed and rows $[2, v-1]$ sorted by row sums, define $\mathcal{C}(A) \subseteq \mathfrak{S}_{r-\lambda}$, the partial matrices *compatible* with A, to be the matrices $B \in \mathfrak{S}_{r-\lambda}$ such that for any row i,

$$\sum_{j=0}^{r-\lambda-1} A_{(i,j)} = \sum_{j=0}^{r-\lambda-1} B_{(i,j)}.$$

Since the compatibility lists are simply a function of the sequence of sorted row sums, in the program we maintain compatibility lists for each each such sequence seen and "Update" the compatibility lists with the matrices generated. And, we also have:

Definition 4.2.6. Given A with $r-\lambda$ columns completed and rows $[2, v-1]$ sorted by row sums, define $\mathcal{G}(A) \subseteq \mathcal{S}_{[2,v-1]}$, the *rowsum-fixing group* of A, to be the permutations $\rho \in \mathcal{S}_{[2,v-1]}$ such that for any row i,

$$\sum_{j=0}^{r-\lambda-1} A_{(i,j)} = \sum_{j=0}^{r-\lambda-1} A_{(\rho i,j)}.$$

That the rowsum-fixing group actually forms a subgroup of $\mathcal{S}_{[2,v-1]}$ follows by noting that if two permutations keep the number of 1's of A fixed, then the matrix resulting from applying one of these then the other to A must have the same number of 1's in each row as A. We can now state the next revision to this program. The groups can be stored as a set of generators known as a strong generating set. This permits efficient group traversal. For details on strong generating sets and how they can be used refer to Kaski and Ostergard [Ks06, 159-162].

In Algorithm 14, we have reduced the matrices considered in FillPhase2 to those compatible with the $\Sigma_1(\mathbf{H})$ and reduced the permutations to those that fix the rowsums of $\Sigma_1(\mathbf{H})$. The next modification that we would like to make is to eliminate the permutations from the rowsum-fixing group of $\Sigma_1(\mathbf{H})$ that are equivalent with respect to $\Sigma_1(\mathbf{H})$ to a previously considered permutation.

Suppose that R is taken from the example in Figure 11 on Page 59, and that D_2 is the matrix:

$$D_2 = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&?&?&?&?&?&?&?&?&?&? \\ 1&1&0&0&?&?&?&?&?&?&?&?&?&? \\ 0&1&1&0&?&?&?&?&?&?&?&?&?&? \\ 0&0&1&1&?&?&?&?&?&?&?&?&?&? \\ 1&0&0&1&?&?&?&?&?&?&?&?&?&? \end{bmatrix}.$$

Then, $D_2 \in \mathcal{C}(\Sigma_1(R))$. The permutation $\rho = (4\ 5)$ is in the rowsum-fixing group of $\Sigma_1(R)$. In FillPhase2 this generates:

$$R|\Sigma_1(\rho D_2) = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&1&1&0&0&0&0&?&?&?&? \\ 1&1&0&0&0&0&1&1&0&0&?&?&?&? \\ 1&1&0&0&0&0&0&0&1&1&?&?&?&? \\ 0&0&1&1&0&0&0&1&1&0&?&?&?&? \\ 0&0&1&1&0&0&1&0&0&1&?&?&?&? \end{bmatrix}.$$

The permutation $\rho' = (3\ 6)$ interchanging the fourth row with seventh row is also in the rowsum-fixing group of $\Sigma_1(R)$ and yields:

Algorithm 14 Program 4 (third version).

```
Prerequisites:
    • A maximum clique-testing algorithm, MaxClique.
    • A canonical form $\Psi$ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
    • A data-structure to hold solutions,$\mathfrak{S}$.
    • Pruning functions $\psi_n$ for $n \in [0, b]$ as in Equation 7 on Page 60.
    • For $A \in \mathcal{M}_{v-2,r-\lambda}$, a means to compute and traverse the group $\mathcal{G}(A)$.

function NodePruningPasses($n, A$)
    For each $M \in \mathfrak{S}_n$ do
        if $\psi_n(A) = M$ then return false.
    if $n = r - \lambda$ then
        Sort rows $[2, v-1]$ of $\psi_n(A)$ by rowsums.
        Update $\mathcal{C}(\psi_n(A))$
    Copy $\psi_n(A)$ into $\mathfrak{S}_n$
    return true.
end function

function FillPhase2
    For each $\rho \in \mathcal{G}(\Sigma_1(\mathbf{H}))$ do
        For each $A \in \mathcal{C}(\Sigma_1(\mathbf{H}))$ do
            $\mathbf{H} \leftarrow \Sigma_1(\mathbf{H})|\Lambda(\mathbf{H})|\Sigma_1(\rho A)$
            RecursePhase3($2r - \lambda$)
    $\mathbf{H} \leftarrow \Sigma_1(\mathbf{H})|\Lambda(\mathbf{H})|[?]$
end function
```

Algorithm 14 Program 4 (third version, continued).

```
function RecursePhase1($n$)
    if not NodeConditionsPass($n, \mathbf{H}$) then return
    if not NodePruningPasses($n, \mathbf{H}$) then return
    if $n = r$ then
        FillPhase2
    if $n < r - \lambda$ then
        For each $\vec{b} \in \binom{\overline{v-2}}{k-1}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            if $n = r - \lambda - 1$ then
                Sort rows $[2, v-1]$ of H by rowsums.
                Compute $\mathcal{G}(\Sigma_1(\mathbf{H}))$
            RecursePhase1(n+1)
            if $n = r - \lambda - 1$ then
                Restore H to the state prior to sorting its rows.
    if $r - \lambda \leq n < r$ then
        For each $\vec{b} \in \binom{\overline{v-2}}{k-2}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            RecursePhase1(n+1)
    $\mathbf{H}_n \leftarrow ?? \ldots ??$
end function
```

$$R|\Sigma_1(\rho' D_2) = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&1&1&0&0&0&0&?&?&?&? \\ 1&1&0&0&0&0&1&0&0&1&?&?&?&? \\ 1&1&0&0&0&0&0&1&1&0&?&?&?&? \\ 0&0&1&1&0&0&0&0&1&1&?&?&?&? \\ 0&0&1&1&0&0&1&1&0&0&?&?&?&? \end{bmatrix}.$$

Now, suppose that we apply the permutation $\rho'' = (3\ 6)(4\ 5)$ to this *entire* matrix:

$$\rho''(R|\Sigma_1(\rho' D_2)) = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&1&1&0&0&0&0&?&?&?&? \\ 0&0&1&1&0&0&1&1&0&0&?&?&?&? \\ 0&0&1&1&0&0&0&0&1&1&?&?&?&? \\ 1&1&0&0&0&0&0&1&1&0&?&?&?&? \\ 1&1&0&0&0&0&1&0&0&1&?&?&?&? \end{bmatrix},$$

followed by the *column* permutation $\sigma = (0\ 2)(1\ 3)$:

$$\rho''(R|\Sigma_1(\rho' D_2))\sigma = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&1&1&0&0&0&0&?&?&?&? \\ 1&1&0&0&0&0&1&1&0&0&?&?&?&? \\ 1&1&0&0&0&0&0&0&1&1&?&?&?&? \\ 0&0&1&1&0&0&0&1&1&0&?&?&?&? \\ 0&0&1&1&0&0&1&0&0&1&?&?&?&? \end{bmatrix}.$$

Then we have the same matrix as $R|\Sigma_1(\rho D_2)$. That is,

$$\rho''(R|\Sigma_1(\rho' D_2))\sigma = R|\Sigma_1(\rho D_2).$$

So the two matrices $R|\Sigma_1(\rho D_2)$ and $R|\Sigma_1(\rho' D_2)$ are actually isomorphic. There are two things to note about these three row permutations:

1. σ only permutes the columns $[0, r-1]$ amongst each other.

2. $\rho'' R \sigma = R$, that is (ρ'', σ) is an automorphism of R.

3. $\rho = \rho''\rho'$.

Since σ only permutes the columns $[0, r-1]$ amongst each other, it commutes with the concatenation operation "|". Any row permutation, does as well. Moreover, all the row permutations here commute with Σ_1 since they fix the first two rows. These properties together imply the observation that was made about these matrices. Indeed,

$$\begin{aligned}\rho''(R|\Sigma_1(\rho' D_2))\sigma &= \rho'' R\sigma|\rho''\Sigma_1(\rho' D_2)) \\ &= \rho'' R\sigma|\Sigma_1(\rho''\rho' D_2)) \\ &= R|\Sigma_1(\rho D_2).\end{aligned}$$

On the other hand, suppose that we have three permutations ρ, ρ' and ρ'' and a column permutation σ such that:

1. σ only permutes the columns $[r, 2r-1]$ amongst each other. So, σ is essentially a permutation of D_2.
2. $\rho'' D_2\sigma = D_2$, that is (ρ'', σ) is an automorphism of D_2.
3. $\rho = \rho'\rho''$.

Then, we can arrive at a similar conclusion,

$$\begin{aligned}R|\Sigma_1(\rho' D_2) &= R|\Sigma_1(\rho'(\rho'' D_2\sigma)) \\ &= (R|\Sigma_1(\rho D_2))\sigma.\end{aligned}$$

Again, ρ and ρ' generated isomorphic matrices. Now we incorporate these observations in a final version of this program. In the preceding discussion, we also discussed the column permutation corresponding to the automorphisms. However, in the program they are not really used. As such, for each $A \in \mathfrak{S}_{r-\lambda}$, let:

$$\mathrm{Aut}_R(A) = \{\rho \in \mathcal{S}_{[2,v-1]};\ \text{there exists } \sigma \in \mathcal{S}_{[0,r-\lambda-1]} \text{ such that } \rho A\sigma = A\}.$$

And, for each matrix R with r rows completed, let:

$$\mathrm{Aut}_R(R) = \{\rho \in \mathcal{S}_{[2,v-1]};\ \text{there exists } \sigma \in \mathcal{S}_{[0,r]} \text{ such that } \rho R\sigma = R\}.$$

These still form groups. If ρ_1 and ρ_2 are in $\mathrm{Aut}_R(A)$ there exists column permutations σ_1 and σ_2 such that $\rho_1\rho_2 A\sigma_2\sigma_1 = \rho_1 A\sigma_1 = A$. Therefore, since the column permutation $\sigma_2\sigma_1$ exists, $\rho_1\rho_2 \in \mathrm{Aut}_R(A)$. The same argument can be used to show that $\mathrm{Aut}_R(R)$ forms a group. These are also stored as strong generating sets computed from the generators provided by *nauty* [MP14] using the techniques described in Kaski and Ostergard [Ks06, 159-162].

Algorithm 15 Program 4 (final version).

Prerequisites:

- A maximum clique-testing algorithm, *MaxClique*.
- A canonical form Ψ for $\mathcal{M}_{v,b}$ over $\mathcal{S}_v \times \mathcal{S}_b$ as defined in Definition 2.4.1.
- A data-structure to hold solutions,$\mathfrak{S}$.
- Pruning functions ψ_n for $n \in [0, b]$ as in Equation 7 on Page 60.
- For $A \in \mathcal{M}_{v-2,r-\lambda}$, a means to compute and traverse the groups $\mathcal{G}(A)$ and $\mathrm{Aut}_R(A)$.
- A data-structure to hold previously seen permutations, $\mathcal{P}$.

```
function NodePruningPasses($n, A$)
    For each $M \in \mathfrak{S}_n$ do
        if $\psi_n(A) = M$ then return false.
    if $n = r - \lambda$ then
        Sort rows $[2, v-1]$ of $\psi_n(A)$ by rowsums.
        Update $\mathcal{C}(\psi_n(A))$
        Compute and store $\mathrm{Aut}_R(\psi_n(A))$
    Copy $\psi_n(A)$ into $\mathfrak{S}_n$
    return true.
end function

function RecursePhase3($n$)
    if not NodeConditionsPass($n, \mathbf{H}$) then return
    if not NodePruningPasses($n, \mathbf{H}$) then return
    if $n = b$ then
        For each $A \in \mathfrak{S}$ do
            if $\Psi(\mathbf{H}) = A$ then return
        Copy $\Psi(\mathbf{H})$ into $\mathfrak{S}$
    if $2r - \lambda \leq n < b$ then
        For each $\vec{b} \in \binom{v-2}{k}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            RecursePhase3(n+1)
    $\mathbf{H}_n \leftarrow ?? \ldots ??$
end function
```

Algorithm 15 Program 4 (final version, continued).

```
function FillPhase2
    Compute and store $\mathrm{Aut}_R(\Sigma_1(\mathbf{H})|\Lambda(\mathbf{H}))$
    Empty $\mathcal{P}$
    For each $\rho \in \mathcal{G}(\Sigma_1(\mathbf{H}))$ do
        For each $\rho' \in \mathrm{Aut}_R(\Sigma_1(\mathbf{H})|\Lambda(\mathbf{H}))$ do
            if $\rho'\rho \in \mathcal{P}$ then continue to next $\rho$
        Insert $\rho$ into $\mathcal{P}$.
        For each $A \in \mathcal{C}(\Sigma_1(\mathbf{H}))$ do
            For each $\rho'' \in \mathrm{Aut}_R(A)$ do
                if $\rho\rho'' \neq \rho$ and $\rho\rho'' \in \mathcal{P}$ then continue to next $A$
            $\mathbf{H} \leftarrow \Sigma_1(\mathbf{H})|\Lambda(\mathbf{H})|\Sigma_1(\rho A)$
            RecursePhase3$(2r - \lambda)$
    $\mathbf{H} \leftarrow \Sigma_1(\mathbf{H})|\Lambda(\mathbf{H})|[?]$
end function

function RecursePhase1$(n)$
    if not NodeConditionsPass$(n, \mathbf{H})$ then return
    if not NodePruningPasses$(n, \mathbf{H})$ then return
    if $n = r$ then
        FillPhase2
    if $n < r - \lambda$ then
        For each $\vec{b} \in \binom{\overline{v-2}}{k-1}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            if $n = r - \lambda - 1$ then
                Sort rows $[2, v-1]$ of $\mathbf{H}$ by rowsums.
                Compute $\mathcal{G}(\Sigma_1(\mathbf{H}))$
            RecursePhase1(n+1)
            if $n = r - \lambda - 1$ then
                Restore $\mathbf{H}$ to the state prior to sorting its rows.
    if $r - \lambda \leq n < r$ then
        For each $\vec{b} \in \binom{\overline{v-2}}{k-2}$ do
            $\mathbf{H}_n \leftarrow \vec{b}$
            RecursePhase1(n+1)
    $\mathbf{H}_n \leftarrow ??\ldots??$
end function
```

4.2.5 Program 5

In this final algorithm, we address a weakness of Program 3. The definition of ψ_n in Equation 5 used the entire group of row permutations. However, the program begins with the matrix **H** shown in Figure 9. This actually reduces the amount of pruning done by ψ_n. We explain this now with an example. When r columns are completed, the following two are not isomorphic:

$$A = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&1&1&?&?&?&?&?&?&?&? \\ 1&1&0&0&0&0&?&?&?&?&?&?&?&? \\ 0&0&1&1&0&0&?&?&?&?&?&?&?&? \\ 1&0&1&0&0&0&?&?&?&?&?&?&?&? \\ 0&1&0&1&0&0&?&?&?&?&?&?&?&? \end{bmatrix},$$

$$B = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&1&1&0&0&0&0 \\ 1&1&0&0&0&0&?&?&?&?&?&?&?&? \\ 1&1&0&0&0&0&?&?&?&?&?&?&?&? \\ 0&0&1&1&0&0&?&?&?&?&?&?&?&? \\ 0&0&1&0&1&0&?&?&?&?&?&?&?&? \\ 0&0&0&1&0&1&?&?&?&?&?&?&?&? \end{bmatrix}.$$

However, if we apply column permutation (0 4)(1 5) and row permutation (1 3) to A we obtain the matrix:

$$C = \begin{bmatrix} 1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&?&?&?&?&?&?&?&? \\ 1&1&0&0&0&0&?&?&?&?&?&?&?&? \\ 1&1&0&0&0&0&1&1&1&1&0&0&0&0 \\ 0&0&1&1&0&0&?&?&?&?&?&?&?&? \\ 0&0&1&0&1&0&?&?&?&?&?&?&?&? \\ 0&0&0&1&0&1&?&?&?&?&?&?&?&? \end{bmatrix}.$$

The first r columns of C and B are the same, and this would also be true if the permutation was applied to any completion, A', of A to b rows. After rearranging the columns beyond r of A', we would have a completion of B to b columns.

The issue is that the completed rows are preventing the row permutation from being considered for isomorphism pruning. If $n < b$, then only the first two rows of A and B are completed and the row permutation ρ can only interchange these rows between each other. In fact, if $n < 2r - \lambda$ even these two rows cannot be interchanged with each other. Therefore, for $n < 2r - \lambda$, fixing the first two rows at the onset has increased the number of matrices considered non-isomorphic by a factor of up to $(v-1)$ and, for $n \geq 2r-\lambda$, by a factor of up to $\frac{v-1}{2}$.

But this inadvertantly imposed reduction can the remedied by truncating the matrix in the original selection of ψ_n, Equation 5. That is, using the truncation to n columns, π_n as in Definition 4.2.2, we can redefine ψ_n as:

$$\psi_n(M) = \begin{cases} \rho'\pi_n(M)\sigma' \text{ such that } \|\rho'\pi_n(M)\sigma'\| = \max_{(\rho,\sigma)\in G_n} \|\rho\pi_n(M)\sigma\| & \text{if } 0 \leq n \leq I, \text{ and} \\ M & \text{if } I < n \leq v. \end{cases}$$

and restore isomorphism testing over the group $G_n = \{(\rho,\sigma) \in \mathcal{S}_v \times \mathcal{S}_b; j \geq n \Rightarrow \sigma j = j\}$.

As a final note, the weakness of Program 3 that is pointed out above also applies to Program 4. It would also be possible to remedy this problem in Program 4, but doing so would be far more complicated. We do not include pseudocode for Program 5 as it is the same as Program 4 with a different choice of pruning function.

4.3 Performance

In this section, we will be compare the performance of the three programs that we have implemented. Since the code for Programs 3, 4, and 5 converge once $2r - \lambda$ columns are completed, we begin with a comparison of all the three while completing the first $2r - \lambda$ columns. Programs 3 and 4 generate the same number of solutions to $2r - \lambda$ columns, so we will only compare the performance of Programs 4 and 5 for completing all b columns.

Table 1 summarizes the performance of the three programs when filling the first $2r - \lambda$ columns. The first column of the table indicates the case number as they are listed in the CRC Handbook of Combinatorial Designs [CD06]. The runtimes are recorded as t_3, t_4, and t_5 where the subscript indicates the program number. Note how this comparison was performed. Isomorphism pruning was performed when each of the first columns $[0, r-1]$ were filled and when completing column $2r - \lambda - 1$. N_r represents the number of non-isomorphic partial matrices generated by Programs 3 and 4 with r columns completed. N_r^* is the number of non-isomorphic partial matrices generated by Program 5 with r columns completed.

We can see that Program 4 is sometimes faster than Programs 3. However, Program 5 is always at least as fast. The cases in which Program 4 performed reasonably well tended to be those cases where many solutions to $2r-\lambda$ were found. This is because the procedure FILLPHASE2 of Program 4 essentially performs some isomorphism testing prior to using the pruning function. Programs 3 and 5 rely entirely on the pruning function for isomorphism testing at level $2r - \lambda$, and it is consequently called more often by these programs in these cases. On the other hand, we refrained from performing isomorphism testing with the pruning function when filling columns $[r - \lambda, 2r - \lambda - 1]$ and it is possible that performing isomorphism testing at these levels would further reduce the runtimes of Programs 3 and 5.

Program 4 performs poorly if many solutions can be eliminated early in Programs 3 and 5 when completing columns $[r - \lambda, 2r - \lambda - 1]$. Indeed, because of the way that these columns are being completed by Program 4, clique-testing is not performed until the entire section is completed. Also note that, in the worst case, the rowsum-fixing group of a completion to $r - \lambda$ columns can be as large as $(v - 2)!$. Although Property 2.6.1 says that any Steiner system is a Katona sieve, we included CRC case 5, the last result in Table 1, to illustrate the effect of the size of the rowsum-fixing group on the performance of Program 4. This case is a Steiner system and there is a unique solution to $r - \lambda$ columns, so the poor performance cannot be due to the number of compatible pairs. Indeed, it can be shown that when $\lambda = 1$ (Steiner systems) there is always a unique solution to $r - \lambda$ columns. For $v = 9$ and $k = 3$ the unique completion to $r - \lambda$

Table 1: Performance results for completing the first $2r - \lambda$ columns.

#	v	b	r	k	λ	N_r	N_r^*	$N_{2r-\lambda}$	$N_{2r-\lambda}^*$	t_3	t_4	t_5
67	7	28	12	3	4	41	20	234	156	0.03	0.02	0.02
117	7	35	15	3	5	119	46	1251	739	0.2	0.1	0.08
191	7	42	18	3	6	332	108	6262	3278	1.36	0.67	0.46
276	7	49	21	3	7	829	231	26969	12626	8.17	4.03	2.35
357	7	56	24	3	8	1966	494	103678	44098	44.09	21.95	11.2
477	7	63	27	3	9	4322	995	355979	140517	207.69	106.37	47.41
275	8	56	21	3	6	4122	727	5511585	1765683	2248.18	1101.84	396.91
21	9	24	8	3	2	7	5	89	71	0.02	0.01	0.02
66	9	36	12	3	3	67	27	17792	10713	2.28	0.9	0.85
150	9	36	16	4	6	88253	11440	3	1	7.99	382.62	7.43
145	9	48	16	3	4	818	191	2658399	996768	560.87	242.01	125.01
30	10	30	9	3	2	7	4	484	424	0.15	0.08	0.07
71	10	30	12	4	4	9811	1309	75	5	1.58	201.55	0.8
55	12	44	11	3	2	14	7	33094	30282	28.47	9.7	9.39
56	12	33	11	4	3	6371	700	6761779	1348772	2049.31	22056.95	224.9
23	13	26	8	4	2	39	14	3853	2199	2.21	7.62	0.77
65	13	52	12	3	2	23	13	338577	314455	544.11	183.99	183.06
16	15	21	7	5	2	19	6	0	0	0	7.81	0
5	16	20	5	4	1	1	1	1	1	0.06	675.66	0.06

columns is:

$$\begin{bmatrix} 1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&1&1&1&1&0&0&0&0&0 \\ 0&0&0&?&?&?&?&?&?&?&?&? \\ 1&0&0&?&?&?&?&?&?&?&?&? \\ 1&0&0&?&?&?&?&?&?&?&?&? \\ 0&1&0&?&?&?&?&?&?&?&?&? \\ 0&1&0&?&?&?&?&?&?&?&?&? \\ 0&0&1&?&?&?&?&?&?&?&?&? \\ 0&0&1&?&?&?&?&?&?&?&?&? \end{bmatrix}.$$

Its rowsum-fixing group is of size 6! and its automorphism group is of size $3! * 2^3$. In general, given v, k, and $\lambda = 1$, the unique solution to $r - \lambda$ will have a rowsum-fixing group of size $(k-2)!(v-k)!$ and automorphism group of size $(k-2)!(r-1)!((k-1)!)^{r-1}$. So, taking into account the permutations eliminated for these cases, we will be applying roughly $\frac{(v-k)!}{(r-1)!((k-1)!)^{r-1}}$ permutations in FillPhase2 of Program 4. And, indeed, this is reflected in the performance results for Case 5. In this case, $\frac{(v-k)!}{(r-1)!((k-1)!)^{r-1}} = 15400$ and the performance difference from Programs 3 and 5 is a factor of roughly 10^5.

Another factor impacting the performance of Program 4 is the loop over all compatible completions to $r - \lambda$ columns. The runtime of algorithm FILLPHASE2 is $\Omega(m^2)$, where m is the length of the longest compatibility list. So, a particularly long list leads to poor runtime. This occurs for CRC case 150.

We explained in Section 4.2.5 that the partial isomorphism testing for Programs 3 and 4, when compared to Program 5, can increase the number of completions to r columns by a factor of up to $(v-1)$ and the number of completions to $2r-\lambda$ columns by a factor of up to $\frac{v-1}{2}$. Some cases come very close to these factors. For example, for case 275 with $v-1=7$, $N_r^*/N_r = 5.66$ and $N_{2r-\lambda}^*/N_{2r-\lambda} = 3.2$. And, for case 150 with $v-1=8$, $N_r^*/N_r = 7.71$. If we compare the runtimes of Programs 4 and 5 to complete all columns, the impact of this reduced number of completions to level $2r-\lambda$ for Programs 5 becomes clear. The increased number of solutions with $2r-\lambda$ blocks completed is proportional to the increased runtime. This is seen in Table 2 where cases with no solutions up to $2r-\lambda$ or with runtime 0 have been omitted.

Table 2: Performance Results for completing all columns.

#	v	b	r	k	λ	$N_{2r-\lambda}$	$N^*_{2r-\lambda}$	t_4	t_5	$\frac{N_{2r-\lambda}/N^*_{2r-\lambda}}{t_4/t_5}$
43	6	20	10	3	4	6	4	0.01	0.01	1.5
118	6	30	15	3	6	13	8	0.03	0.02	1.1
236	6	40	20	3	8	33	17	0.1	0.07	1.4
31	7	21	9	3	3	37	28	0.02	0.02	1.3
67	7	28	12	3	4	234	156	0.14	0.11	1.5
117	7	35	15	3	5	1251	739	1.02	0.66	1.1
191	7	42	18	3	6	6262	3278	7.24	4.06	1.1
276	7	49	21	3	7	26969	12626	41.92	20.69	1.1
357	7	56	24	3	8	103678	44098	209.74	93.98	1.1
477	7	63	27	3	9	355979	140517	925.47	384.86	1.1
21	9	24	8	3	2	89	71	0.07	0.07	1.3
66	9	36	12	3	3	17792	10713	176.03	110.86	1.0
275	8	56	21	3	6	5511585	1765683	41917.77	13860.55	1.0

The performance results indicate that the simpler method was more effective for this problem. It should be noted, however, that it would be possible to improve the isomorphism testing of of the technique used in Program 4. Program 4 could be modifed so that during the procedure FillPhase2 isomorphisms with respect to the row-group $\mathcal{S}_V$ are pruned as in Program 5. To do so one would need to track the row to which such a permutation maps the second row, extract the $r - \lambda$ columns not including this row, compute its canonical form and traverse the compatibility list of this matrix instead. However, doings so would be unlikely to lead to performance improvements as can be seen comparing Programs 3 and 4.

Chapter 5

Results

Before presenting the results of the computer search, we present the conclusions that follow from the theory presented in earlier chapters. In particular,

1. If $v < (t+1)(k-t+1) = 3(k-1)$, because of Corollary 3.3.4, there cannot exist a Katona sieve for any value of λ.

2. For $\lambda = 1$, by Property 2.6.1, any Steiner system is a Katona sieve.

3. If $\lambda > 1$, then by Theorem 2.6.5, a symmetric 2-design cannot be a Katona sieve.

These results, combined with the results for 2 designs published in the CRC Handbook [CD06] completely resolve the problem for 71 out of the 142 cases with $b \leq 64$ and $r \leq 21$. Since Katona sieves can be concatenated to yield Katona sieves by Property 2.6.4, this resolves the existence problem for an additional 21 cases.

Table 3 lists the results of the computer search achieved with the programs in Chapter 4. The first column indicates the case number as enumerated in the CRC Handbook. The column labelled N_K is the number of Katona sieves that were found, and N_d is the number of known designs as published in the CRC Handbook. Cases with a * are the 17 cases for which the existence problem was settled through computation. For one of these the existence problem was not established directly by computation, but can be deduced through concatenation of a computed case and Property 2.6.4. In particular, for CRC case 190, a 2-(10,3,4) Katona sieve was not computed, but one can be constructed by concatenating two computed 2-$(10, 3, 2)$ Katona sieves. Cases with † are cases for which a greater number of designs than previously listed in the CRC Handbook were found.

Table 4 lists the 33 cases with $r \leq 21$ and $b \leq 64$ for which the existence of 2-(v, k, λ) Katona sieves was not determined by this work. For cases that were attempted on a 3.0 Ghz system with

Table 3: Computed Results.

CRC #	**v**	**b**	**r**	**k**	λ	N_K	N_d
*4	6	10	5	3	2	1	1
9	7	14	6	3	2	4	4
*10	10	15	6	4	2	0	3
21	9	24	8	3	2	13	36
23	13	26	8	4	2	184	2461
*24	9	18	8	4	3	0	11
*30	10	30	9	3	2	111	960
31	7	21	9	3	3	10	10
*35	16	24	9	6	3	0	18920
*36	28	36	9	7	2	0	8
*43	6	20	10	3	4	4	4
44	16	40	10	4	2	≥ 321	$\geq 2.2 \times 10^6$
46	21	42	10	5	2	≥ 105	≥ 22998
*49	21	30	10	7	3	0	3809
*55	12	44	11	3	2	≥ 10230000	242995846
*56	12	33	11	4	3	0	≥ 17172470
†65	13	52	12	3	2	≥ 10230000	≥ 1897386
66	9	36	12	3	3	4215	22521
67	7	28	12	3	4	35	35
†70	13	39	12	4	3	≥ 157091	≥ 3702
*71	10	30	12	4	4	0	13769944
74	31	62	12	6	2	≥ 15	≥ 72
*†116	11	55	15	3	3	≥ 10230000	≥ 436800
117	7	35	15	3	5	109	109
*118	6	30	15	3	6	6	6
145	9	48	16	3	4	4061937	16585031
*150	9	36	16	4	6	1	270474142
*190	10	60	18	3	4	≥ 1	≥ 961
191	7	42	18	3	6	418	418
*236	6	40	20	3	8	13	13
*275	8	56	21	3	6	773919	3077244
276	7	49	21	3	7	1508	1508

64 GB of RAM, some comments are included. The programs print the runtime when a multiple of 2500 of partial solutions to $r - \lambda$ or $2r - \lambda$ columns are found and when complete solutions are found. As such, in some cases, no value for the runtime was printed and we only say "after a long time". In all cases, this indicates at least 8 hours but generally many days.

Finally, the fourteen cases listed in Table 5 are the cases with $r \leq 21$ and $b \leq 64$ for which at least one Katona sieve exists but not all have been enumerated. Case 32 is a Steiner system and N_K is taken directly from the CRC Handbook. For cases 116, 55, 65, 70, 44, and 46, N_K is the number of designs generated by the programs of Chapter 4. The remaining cases must have at least one Katona sieve by concatenating smaller Katona sieves.

We conclude this section with the result of the very special case which started our research. Of over 270 million 2-$(9, 4, 6)$ designs enumerated by Östergård [Ös01], Figure 12 is the only Katona sieve. Its automorphism group is of order 54. Since none of the 11 2-$(9, 4, 3)$ designs are Katona sieves, a 2-$(9, 4, 6)$ sieve could not have been constructed using Property 2.6.4. When $v \geq (t + 1)(k - t + 1)$, if we fix both k and v, and let λ increase, the Erdős-Ko-Rado theorem guarantees that sooner or later, a Katona sieve exists. So it is interesting to find the minimum value of λ for which a Katona sieve exists. In the case $t = 2$, $v = 9$, and $k = 4$, the value $\lambda = 3$ was not sufficient and the minimum λ is 6.

$$\begin{bmatrix}
111111 & 111111 & 1111 & 000000 & 0000 & 0000 & 000000 \\
111111 & 000000 & 0000 & 111111 & 1111 & 0000 & 000000 \\
000000 & 111111 & 0000 & 111111 & 0000 & 1111 & 000000 \\[1ex]
111000 & 001000 & 1100 & 010001 & 1000 & 0111 & 111100 \\
000011 & 000111 & 0001 & 000010 & 1110 & 0101 & 111010 \\[1ex]
001001 & 011100 & 1000 & 001000 & 0111 & 1010 & 001111 \\
010110 & 000010 & 0011 & 000101 & 0001 & 1011 & 010111 \\[1ex]
100000 & 110000 & 0111 & 111000 & 1100 & 1000 & 110011 \\
000100 & 100001 & 1110 & 100110 & 0011 & 0100 & 101101
\end{bmatrix}$$

Figure 12: The unique 2-$(9, 4, 6)$ Katona sieve

Table 4: Unsolved Existence Cases for $r \leq 21$ and $b \leq 64$.

CRC #	**v**	**b**	**r**	**k**	λ	**Comment:**
59	45	55	11	9	2	
69	19	57	12	4	2	No solutions after a long time.
75	21	42	12	6	3	
76	16	32	12	6	4	
79	33	44	12	9	3	No solutions after a long time.
90	27	39	13	9	4	No solutions after a long time.
91	40	52	13	10	3	
102	15	42	14	5	4	No solutions when out of memory.
104	15	35	14	6	5	No solutions after 2.5×10^5 sec.
107	29	58	14	7	3	No solutions after 2.4×10^5 sec.
108	22	44	14	7	4	No solutions after 7.4×10^4 sec.
123	16	48	15	5	4	No solutions after a long time.
124	13	39	15	5	5	No solutions when out of memory.
128	16	40	15	6	5	
134	28	42	15	10	5	No solutions after a long time.
153	21	56	16	6	4	
157	29	58	16	8	4	
161	33	48	16	11	5	No solutions after 7.4×10^5 sec.
163	45	60	16	12	4	
176	18	51	17	6	5	No solutions when out of memory (4.1×10^6 sec).
193	10	45	18	4	6	No solutions when out of memory (10^5 sec).
199	19	57	18	6	5	No solutions after 1.2×10^6 sec.
200	16	48	18	6	6	
207	25	50	18	9	6	No solutions after 1.2×10^6 sec.
211	34	51	18	12	6	
225	39	57	19	13	6	
242	11	55	20	4	6	No solutions when out of memory (3×10^6 sec).
250	21	60	20	7	6	No solutions after 1.2×10^6 sec.
258	31	62	20	10	6	No solutions after a long time.
280	15	63	21	5	6	
284	16	56	21	6	7	
289	19	57	21	7	7	
297	40	60	21	14	7	

Table 5: Unsolved Enumeration Cases for $r \leq 21$ and $b \leq 64$.

CRC #	**v**	**b**	**r**	**k**	λ	N_K
32	28	63	9	4	1	≥ 4747
44	16	40	10	4	2	≥ 321
46	21	42	10	5	2	≥ 105
55	12	44	11	3	2	≥ 10230000
65	13	52	12	3	2	≥ 10230000
70	13	39	12	4	3	≥ 157091
72	25	60	12	5	2	≥ 1
74	31	62	12	6	2	≥ 15
116	11	55	15	3	3	≥ 10230000
119	16	60	15	4	3	≥ 1
122	21	63	15	5	3	≥ 1
149	13	52	16	4	4	≥ 1
190	10	60	18	3	4	≥ 1
235	9	60	20	3	5	≥ 1

Chapter 6

Conclusion

Katona sieves form a class of combinatorial objects that have not been studied previously. We have shown how they are closely related to the Erdős-Ko-Rado theorem. In a strong sense they are generalizations of Steiner systems. In fact, Katona himself observed that the existence of a Steiner system for a given set of parameters implies EKR-t, stating that he never published this previously since existence of Steiner systems is itself a hard problem [Kat00]. It's therefore unlikely that he brought it a step further to consider (λ, t)-disjoint designs very deeply even if he may have known the results herein. That is, not only a Steiner systems, but also (λ, t)-disjoint designs can be used to generalize his proof. Moreover, Tit's lower bound on the existence of Steiner systems[Tit64] is directly extendable to a lower bound on the existence of (λ, t)-disjoint designs. For this reason we consider these as extensions of Steiner systems and name them Katona sieves in honor of his proof.

An extensive computer search across many cases was performed. The results represent over 200 CPU days of computation. In some cases, after weeks of generation, no solutions were generated. But, in other cases, very many solutions were found. In fact, for some cases we generated more designs than published in the CRC Handbook [CD06]. One case of particular interest is the case of 2-$(9, 36, 6)$ designs. Chvátal noted that any Katona sieves for this case must have been generated by Östergård, and, indeed, a *unique* Katona sieve exists amongst the 270 million t-designs that exist for this case[Ös01].

There are still 33 unsolved cases amongst for the existence of Katona sieves amongst 2-designs with $b \leq 64$ and $r \leq 21$ and an additional 14 for which not all sieves have been enumerated. Improvements in isomorphism testing or on clique finding may enable one to solve the problem for some of these designs. For our implementation of the programs herein, cases with $b > 64$ would require a complete rewrite of the program because designs are represented as a fixed number of 64-bit words and this representation is assumed throughout for bitwise operations.

Bibliography

[Art91] M. Artin. *Algebra.* Prentice Hall, 1991.

[CD06] C. J. Colbourn and J. H. Dinitz. *Handbook of Combinatorial Designs, Second Edition.* Chapman & Hall/CRC, 2006.

[Chv12] V. Chvátal. Private communication, April 2012.

[EFK92] P. L. Erdős, U. Faigle, and W. Kern. A group-theoretic setting for some intersecting Sperner families. *Combinatorics, Probability and Computing*, 1:323–334, 1992.

[EKR61] P. Erdős, C. Ko, and R. Rado. Intersection theorems for systems of finite sets. *Quarterly Journal of Mathematics, Oxford*, pages 313–318, 1961.

[Fra78] P. Frankl. The Erdős-Ko-Rado theorem is true for n = ckt. In *Combinatorics, Proc. Fifth Hungarian Colloq. Combin*, pages 365–375. North-Holland, 1978.

[Kat72] G. O. H. Katona. A simple proof of the Erdős-Chao Ko-Rado theorem. *Journal of Combinatorial Theory, Series B*, 13(2):183–184, 1972.

[Kat00] G. O. H. Katona. The cycle method and its limits. In I. Althöfer, N. Cai, G. Dueck, L. Khachatrian, M.S. Pinsker, A. Sárközy, I. Wegener, and Z. Zhang, editors, *Numbers, Information and Complexity.*, pages 129–141. Kluwer Academic Publishers, Norwell, MA, 2000.

[Ks06] P. Kaski and P. R. J. Östergård. *Classification Algorithms for Codes and Designs.* Number 15 in Algorithms and Computation in Mathematics. Springer-Verlag, Berlin Heidelberg, 2006.

[MP14] B. D. McKay and A. Piperno. Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014.

[Ns03] S. Niskanen and P. R. J. Östergård. Cliquer user's guide, version 1.0. Technical Report T48, Communications Laboratory, Helsinki University of Technology, Espoo, Finland, 2003.

[Ös01] P. R. J. Östergård. There are 270,474,142 nonisomorphic 2-(9, 4, 6) designs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 37:173–176, 2001.

[Tit64] J. Tits. Sur les systemes de steiner associes aux trois "grands" groupes de mathieu. *Rendic. Math.*, 23:166–184, 1964.

[Wil84] R. M. Wilson. The exact bound in the Erdős-Ko-Rado theorem. *Combinatorica*, 4(2):247–257, 1984.